



Hosting Guide

Amazon GameLift Servers



Amazon GameLift Servers: Hosting Guide

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

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is Amazon GameLift Servers?	1
Key features	1
How to work with Amazon GameLift Servers	4
Game hosting options	4
Managed EC2	5
Managed containers	6
Anywhere hosting	6
Hybrid hosting	7
How it works	8
Core components	8
How components work together	14
Player experience	16
Provide fast gameplay to players globally	16
Help players join games quickly	16
Build creative ways to group players for game sessions	17
Deliver reliable connections throughout gameplay	17
Improve player experience based on real-world data	18
Integrate enhanced game features with other AWS services	18
Game hosting in production	19
Monitoring hosting health and performance	19
Managing game server updates and patches	20
Optimizing performance and scaling	20
Troubleshooting and live operations	21
Tools and integration	21
Related resources	22
Key terms	22
Getting started	30
Before you start	30
Choose your path	30
I want to explore Amazon GameLift Servers quickly	30
I want to build a custom hosting solution	31
Recommended learning path	31
Set up an AWS account	32
Sign up for an AWS account	33

Set user permissions for Amazon GameLift Servers	33
Set up programmatic access for users	34
Set up programmatic access for your game	36
IAM permission examples	37
Set up an IAM service role	42
Get development tools	46
For game servers	46
For game clients	49
For game backend services	49
For resource management	50
For Amazon GameLift Servers Realtime	51
Tutorial: Quick onboarding with the Amazon GameLift Servers wrapper	52
Prerequisites	52
Overview	53
Step 1: Get and build the game server wrapper	53
Step 2: Prepare your game server build	54
Step 3: Configure the wrapper for your fleet	54
Step 4: Upload the game server build	56
Step 5: Create the managed EC2 fleet	56
Step 6: Create and connect to a game session	58
Step 7: Manage and monitor your fleet	59
Step 8: Scale your game servers	59
Troubleshoot common issues	60
Next steps	62
Amazon GameLift Servers examples	62
Custom game server example	62
Building a game hosting solution	64
Development roadmaps	64
Managed EC2 development roadmap	65
Managed containers development roadmap	71
Anywhere development roadmap	78
Hybrid hosting development roadmap	83
Prepare a game for hosting	91
Prepare with the game engine plugin	91
Integrate a game server	176
Package a game server build	204

Amazon GameLift Servers interactions	217
Build a backend service	223
Authenticating your players	223
Integrate game client functionality	224
Serverless backend	229
WebSocket-based backend	231
Deploy hosting fleets	233
Fleet characteristics	233
Choose a hosting option	234
Managed EC2 fleets	236
Managed container fleets	248
Anywhere fleets	271
Build a hybrid hosting solution	281
Configure game session placement	284
Queue characteristics	285
Best practices	286
Create a queue	287
Set up event notification	290
Customize your game hosting solution	294
Game server builds	295
Player sessions and matchmaking	308
Game session placements	310
Hosting resources	325
Managing a game hosting solution	352
Manage game hosting resources	352
Amazon GameLift Servers console features	352
Console dashboard	353
Builds	354
Fleets	357
Aliases	378
Game session queues	381
Prepare for game launch	384
Getting ready	384
Prepare for testing	385
Prepare for launch	385
Plan for post-launch	386

Manage game hosting resources with CloudFormation	386
Best practices	387
Using CloudFormation stacks	388
Updating builds	391
DDoS protection	394
Key benefits	394
How player gateway works	394
Traffic flow	395
Core concepts	395
Monitoring player gateway	396
IPv4 and IPv6 compatibility	396
Enable player gateway on fleets	397
Prerequisites	397
Player gateway modes	397
IPv4 and IPv6 compatibility	398
Integrate player gateway into a game	398
Backend integration	399
Client integration	400
Test your integration	402
Best practices	402
Testing and troubleshooting	404
Set up for iterative development	404
Build a cloud-based test environment	406
Set up local testing	409
Set up local testing (legacy)	415
Debug fleet issues	423
Fleet creation issues	423
Server process issues	425
Fleet deletion issues	425
Remotely access fleet resources	425
Connect to fleet instances	426
Connect to containers	435
Scaling	442
To manage fleet capacity in the console	442
Set hosting capacity limits	443
To set capacity limits	443

Manually set fleet capacity	445
Suspend auto scaling	445
To manually set fleet capacity	445
Auto scale fleet capacity	447
Target-based auto scaling	447
Rule-based auto scaling	449
Manage Scaling a Fleet To/From Zero	453
Scale-in behavior	454
Scale-out behavior	454
Configuring Scale To/From Zero	455
Best practices	456
Scale container fleets	456
Monitoring	459
Metrics comparison	460
Metrics availability by source	460
Topics	469
Monitor with CloudWatch	469
Metrics dimensions	470
Fleet metrics	471
DDoS protection (player gateway) metrics	487
Queue metrics	489
FlexMatch metrics	493
FleetIQ metrics	497
Monitor with server telemetry metrics	499
Benefits of telemetry metrics	500
Before you begin	501
Implementation	502
Available metrics	503
How it works	507
Dashboard organization and usage	508
Common monitoring scenarios	519
Troubleshooting guide	524
Logging API calls	526
Amazon GameLift Servers information in CloudTrail	527
Understanding Amazon GameLift Servers log file entries	528
Logging server messages	530

EC2 vs Container logging	530
Logging for custom servers	533
Security	537
Data protection	538
Encryption at rest	539
Encryption in transit	540
Internetwork traffic privacy	541
Identity and access management	541
Audience	541
Authenticating with identities	542
Managing access using policies	543
How Amazon GameLift Servers works with IAM	545
Identity-based policy examples	551
Troubleshooting	556
AWS managed policies	558
Logging and monitoring with Amazon GameLift Servers	560
Compliance validation	561
Resilience	561
Infrastructure security	562
Configuration and vulnerability analysis	563
Security best practices	564
Keep fleet runtime environments up to date	564
Secure your port configurations	565
Additional security resources	566
Pricing	567
Pricing models	567
Generate pricing estimates	568
Estimate managed hosting	568
Cost optimization strategies	571
Best practices	571
Resource cost and utilization tools	573
Performance monitoring	574
Reference guides	575
Service API (AWS SDK)	575
Manage Amazon GameLift Servers hosting resources	576
Start game sessions and join players	579

Server SDK 5.x	581
Server SDK 5.x updates	581
Migrate to server SDK 5.x	583
C++ server SDK	585
C# server SDK	639
Go server SDK	687
C++ (Unreal) server SDK: Actions	717
Server SDK 4 and earlier	768
Server SDK for C++: Actions	769
Server SDK for C#: Actions	792
Server SDK for Unreal: Actions	813
Game session placement events	828
Placement event syntax	829
PlacementFulfilled	829
PlacementCancelled	832
PlacementTimedOut	833
PlacementFailed	834
AMI versions	835
Service locations	837
Supported AWS locations	837
Locations for managed hosting	842
Locations for Amazon GameLift Servers Anywhere	843
Locations for Amazon GameLift Servers FlexMatch	844
Amazon GameLift Servers in China	844
Service endpoints and quotas	844
Amazon GameLift Servers API limits	845
API limits reference	845
Resource-level throttling	854
Amazon GameLift Servers SDK API limits	854
UDP ping beacons	857
How UDP ping beacons work	858
Common use cases for UDP ping beacons	858
Getting beacon endpoints	859
Implementing latency measurements	860
Code examples	861
FAQs on supported protocols	869

What are the AWS JSON and Smithy RPC v2 CBOR protocols, and how do they differ?	869
How do I get started with the Smithy RPC v2 CBOR protocol?	869
What are the risks of enabling the Smithy RPC v2 CBOR protocol?	870
What if my solution does not support Smithy RPC v2 CBOR?	870
What AWS Regions support the Smithy RPC v2 CBOR protocol?	870
What latency improvements can I expect when upgrading to SDK versions that support Smithy RPC v2 CBOR?	870
Will the AWS JSON protocol be deprecated?	870
Where can I find more information about Smithy RPC v2 CBOR?	871
Release notes and SDK versions	872
SDK versions	872
Release notes	882

What is Amazon GameLift Servers?

Use Amazon GameLift Servers to deploy, operate, and scale dedicated, low-cost servers in the cloud for session-based multiplayer games. Built on AWS global computing infrastructure, Amazon GameLift Servers helps deliver high-performance, high-reliability game servers while dynamically scaling your resource usage to meet worldwide player demand.

Key features

Global reach and low latency

- Deploy hosting in AWS Regions and Local Zones worldwide to bring your games to new players. See a complete list of [locations supported by Amazon GameLift Servers](#).
- Deliver lowest possible latency and great player experience, wherever your players are located.
- Use multi-location placement queues to dynamically place players into game sessions with optimal player experiences.

Automatic scaling for managed fleets

- Optimize fleet capacity for cost efficiency while maintaining quality of service.
- Scale capacity based on player demand using target-based auto scaling
- Maintain capacity buffer to handle sudden player influxes.

Flexible hosting options

See more details at [Amazon GameLift Servers game hosting options](#).

- Game hosting in the cloud, managed by Amazon GameLift Servers
 - Host game servers on Amazon Elastic Compute Cloud (Amazon EC2) infrastructure. Choose from resource types including compute-optimized (C-family), memory-optimized (R-family), and cost-efficient ARM-based Graviton instances.
 - Customize how Amazon GameLift Servers manages all aspects of your game hosting solution, from deploying game servers and starting game sessions to matching players and getting them into games.
 - Use multi-location fleets to streamline global game server deployment.

- Take advantage of low-cost hosting with [Amazon EC2 Spot Instances](#).
- Deploy Windows or Linux game servers with managed EC2 hosting.
- Deploy Linux-based containers with your game server with managed container hosting.
- Game hosting managed by you, with Amazon GameLift Servers Anywhere
 - Leverage Amazon GameLift Servers game session placement and session management capabilities.
 - Self-manage game server deployments and scaling on your own hardware, on-premises infrastructure, or other cloud providers.
- Hybrid game hosting
 - Create a game hosting solution that uses a combination of Amazon GameLift Servers managed fleets and Anywhere fleets to work with existing solutions or support migration.

Game session management

- Let Amazon GameLift Servers manage game server processes for you, track game server status and availability, and place new game sessions on request.
- Use game session placement features to help players find and join game sessions that will provide the best possible player experience. Rely on Amazon GameLift Servers decision-making, or customize around criteria such as hosting cost, player latency, and geographic location.
- Create game session placement queues to efficiently process new game session requests. Use features to set a balance between placement quality and speed that fits your game and player expectations.
- Take advantage of optional player session features to reserve player slots in game sessions, validate players on connection, and monitor slot availability. Search or filter existing game sessions to find available slots.

Advanced matchmaking with FlexMatch

- Add robust matchmaking that's fully integrated with your Amazon GameLift Servers hosting solution.
- Build custom matchmaking rules based on player attributes and preferences, such as skill level or character choice, and use complex logic as needed.
- Set up team-based matches for competitive or cooperative gameplay. Balance players across team and/or match composition.

- Optimize the matchmaking algorithm to balance match quality and wait times.
- Support matches from 2 to 200 concurrent players.
- Support match backfilling to fill available player slots and keep game sessions full.

Comprehensive monitoring

- Get real-time metrics on usage and performance for game sessions and player activity, as well as hosting performance and health for managed fleets.
- Integrate with Amazon CloudWatch for alerts and dashboards.
- Access server logs and event tracking for troubleshooting.
- Analyze performance and utilization data to optimize scaling (managed fleets) or resource allocation (self-managed fleets).

Customization and extensibility

- Customize game session placement logic, including prioritizing for latency, cost, and location.
- For managed hosting, choose your game hosting resources and provide instructions for launching and running game servers on each instance. For managed containers, fine-tune resource allocation across containers.
- Define your own matchmaking rules.
- Implement custom strategies for capacity scaling.
- Add game server code to integrate with your existing AWS services, such as databases or content storage tools.
- Create a backend service with custom features to manage player join requests (such as player parties), handle authentication, or add custom logic and persistence.

Integration with AWS ecosystem

- Use services such as Amazon DynamoDB, Amazon Simple Storage Service, and Amazon Aurora DSQL for game state persistence.
- Implement Amazon Cognito for player authentication.
- Process game analytics with Amazon Kinesis and Amazon S3.
- Add voice chat with Amazon Chime SDK.

- Create custom game features using AWS Lambda and Amazon API Gateway.

How to work with Amazon GameLift Servers

Use these tools to work with Amazon GameLift Servers.

AWS CLI

Use the AWS Command Line Interface (AWS CLI) to make calls to the AWS SDK, including the service API for Amazon GameLift Servers. See [Getting started with the AWS CLI](#) in the *AWS Command Line Interface User Guide*.

Amazon GameLift Servers console

Use the [AWS Management Console for Amazon GameLift Servers](#) to configure resources, manage your game server deployments, and track performance and usage metrics. The Amazon GameLift Servers console is a GUI alternative to managing resources programmatically or with the AWS CLI.

Amazon GameLift Servers SDKs

The Amazon GameLift Servers SDKs contain the libraries required to establish communication between your game clients, game servers, and game services and the Amazon GameLift Servers service. For more information, see [Get Amazon GameLift Servers development tools](#).

AWS CloudFormation

Use AWS CloudFormation to model, provision, and manage AWS resources for your game hosting solution by treating infrastructure as code. Create templates that describe the resources, and CloudFormation automates the tasks of configuring and deploying resources to the locations you specify.

Amazon GameLift Servers game hosting options

Amazon GameLift Servers offers a range of options for hosting game servers for session-based multiplayer games. Whether you want to set up cloud-based hosting managed by Amazon GameLift Servers or incorporate hosting resources that you already have, you can work with Amazon GameLift Servers to build the hosting solution you need for your players. For more details about what a game hosting solution looks like, see [How hosting with Amazon GameLift Servers works](#).

Topics

- [Managed EC2](#)
- [Managed containers](#)
- [Anywhere hosting](#)
- [Hybrid hosting](#)

Managed EC2

Offload the work of managing your production game servers onto Amazon GameLift Servers with managed EC2 hosting. Take advantage of optimizations for multiplayer game servers while still relying on the high performance and reliability of Amazon Elastic Compute Cloud (Amazon EC2) and AWS global computing infrastructure.

[Start building an Amazon GameLift Servers managed EC2 hosting solution](#)

Characteristics

- Host multiplayer game servers that run on Amazon Linux or Windows Server operating systems.
- Use Amazon GameLift Servers to deploy and manage your custom game servers to hosting resources AWS Cloud. Choose the hardware type and where to deploy it, and configure additional details only where you need to.
- Configure the runtime process manager to automatically maintain game server processes as needed to host game sessions.
- Set up custom health tracking to help Amazon GameLift Servers detect and resolve poor-performing game servers.
- Take advantage of Amazon GameLift Servers performance monitoring. Track metrics such as hardware performance and server process life cycles. You can also download and store game session logs.
- Use Amazon GameLift Servers to manage access between game server processes and other AWS resources.
- For production hosting, automate game hosting deployments and management with AWS CloudFormation templates and the AWS Cloud Development Kit (AWS CDK). Take advantage of continuous integration and continuous delivery (CI/CD) tools and services such as AWS CodePipeline.

Managed containers

Amazon GameLift Servers provides a complete cloud hosting solution for containerized game servers, so you can take advantage of the core benefits of container usage, such as portability, agility, and fault tolerance. As with managed EC2 hosting, managed container hosting deploys and runs your containers on Amazon EC2 instances.

[Start developing an Amazon GameLift Servers managed hosting solution for your containerized game server.](#)

Characteristics

- Develop a custom architecture with lightweight containers to run your game server build, dependencies and auxiliary software.
- Use Docker tools to create a Linux-based container image. Store images in an Amazon Elastic Container Registry (Amazon ECR) repository for deployment.
- Use Amazon GameLift Servers to deploy and manage your custom game servers to hosting resources AWS Cloud. Choose the hardware type and where to deploy it, and configure additional details only where you need to.
- Manage hosting fleet life-cycle with tools to model game server versions. Deploy game server updates and other configuration changes as needed.
- Track game hosting performance with container-specific performance metrics. Monitor the health of your fleet resources using hardware metrics.
- Manage container fleet resources using AWS CloudFormation templates for Amazon GameLift Servers.

Anywhere hosting

With Anywhere hosting, you can take advantage of Amazon GameLift Servers game session management features, including matchmaking and game session metrics, to host game servers wherever you want to. Self-manage your game server deployments, game server health monitoring, and capacity scaling.

Note

Anywhere fleets are particularly useful as test environments for rapid, iterative game development. Set up an Anywhere fleet for your own local workstation or a cloud-based

hosting resource. For more information about testing with Anywhere, see [Set up local testing with Amazon GameLift Servers Anywhere](#).

[Start developing an Amazon GameLift Servers Anywhere hosting solution for your game.](#)

Characteristics

- Deploy game servers on your own hardware, on-premises infrastructure, or other cloud providers. Make use of available hardware close to your players, anywhere.
- Use Amazon GameLift Servers session management features to monitor game server availability, start new game sessions, and join players to game session slots.

Hybrid hosting

For production hosting, you might choose a hybrid approach, with Anywhere fleets for the hosting resources you supply and manage, and Amazon GameLift Servers managed fleets filling in where needed. A hybrid solution uses the same processes to start game sessions and connect players to deliver a seamless player experience. You can monitor and collect usage metrics for all hosting, regardless of resource type.

Common scenarios where a hybrid solution makes sense:

- **Expand your hosting solution to the AWS Cloud.** Supplement the capabilities of your existing hosting solution (on-premises hardware or other cloud-based hosting) by adding Amazon GameLift Servers managed hosting. With managed hosting, you can increase your hosting capacity or add "burst" capacity to rapidly scale up and pay only for resources when you need them. You can also take advantage of the Amazon GameLift Servers service's global footprint to reach more players around the world and provide the low-latency multiplayer experience they expect.
- **Prepare for migration to cloud-based hosting.** If you're considering or planning to migrate to the AWS Cloud, a hybrid hosting solution is a viable way for you to make the transition as gradually as you need to. It lets you build the solution you need right now while preparing for where you want to be in the future.
- **Lower latency for players in locations beyond those serviced by Amazon GameLift Servers.** If you're already using Amazon GameLift Servers managed hosting, you might need to support players in unusual situations. For example, you might want to better reach players in unusually

remote locations or support players with special access needs. Add custom hosting locations and use Amazon GameLift Servers Anywhere to support those locations along with your managed hosting resources.

-

[Start developing an Amazon GameLift Servers hybrid hosting solution for your game.](#)

How hosting with Amazon GameLift Servers works

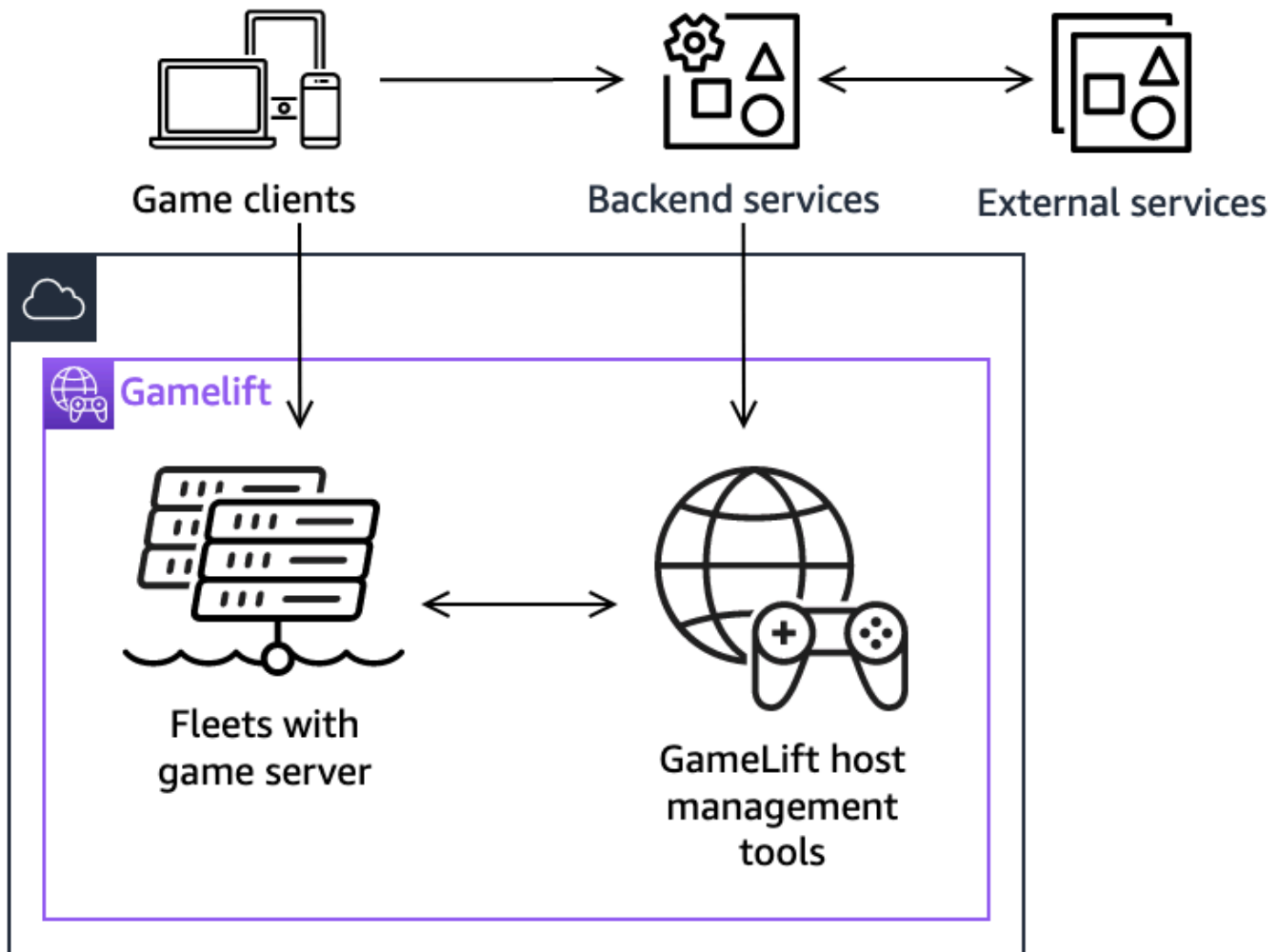
Amazon GameLift Servers is a service that provides dedicated, low-cost servers, infrastructure provisioning, scaling, and session management for your session-based multiplayer games. The service offers flexible tools and features that you can customize for your games or use in collaboration with your own systems. Take advantage of Amazon GameLift Servers managed hosting so you can focus on creating gameplay experiences rather than managing server operations.

This topic describes the core components of a complete Amazon GameLift Servers hosting solution and how they work together to supply multiplayer game sessions to players.

Core components of a game hosting solution

A complete Amazon GameLift Servers hosting solution consists of several essential components that work together to deliver multiplayer game experiences. Some of these components are built and managed by you, while others are provided by Amazon GameLift Servers and managed based on your configuration choices. Understanding how these components interact and their respective roles is crucial for developing an effective game hosting solution. The components include:

- Game client runs on player devices.
- Backend service enables game clients to communicate with Amazon GameLift Servers to get game session information and join games.
- Game server software runs on game hosting resources and hosts game sessions for a group of players.
- Placement system initiates game sessions and matches players to games.
- Game hosting fleets provide game servers in one or more geographic locations.
- Game hosting management system monitors game hosting status and manages capacity.



Game client

The game client is your game software that runs on a player's device. It initiates player placement into a game session by communicating with a backend service and connects directly to a game server to participate in gameplay.

Key functions for game hosting

- Send join requests to the backend service. Include relevant player data and game session data as part of the join request.
- Collect latency data for the game client and include that information in the join request.
- Receive game session connection information from the backend service and use it to connect to a game server.

- Handle session connection scenarios such as player verification, match acceptance for Amazon GameLift Servers FlexMatch, session interruptions, or disconnects.

Who builds it

You add game hosting functionality to your game client software and set up communication with the backend service.

Backend service

The backend service is a coordination layer between game clients and the Amazon GameLift Servers service. It controls all communication with the service to make requests for game session placements and to retrieve game session and player session information. Use of a backend service is a best practice that maintains secure communication with the service on behalf of game clients and avoids having to share sensitive AWS credentials and permissions with game clients.

Your backend service implements a player grouping strategy for your game sessions. You have a lot of flexibility in how players end up in a game session. You might set up a game session browser for players or support player-defined parties. Or you might pool game session requests as they come in, form player groups (possibly with pre-sorting), and start a game session for each group. With FlexMatch matchmaking, you can form player matches and backfill existing matches.

Key functions for game hosting

- Authenticate communications from game clients.
- Make requests to Amazon GameLift Servers to start new game sessions, get game session information, or join existing game sessions. Include relevant game session and player data in requests as needed.
- Optionally create requests for player sessions to more closely monitor game session availability. Use player sessions to reserve game session slots, validate players when they connect, and track player disconnects.
- Respond to game clients with game session connection details and other information as requested.

Who builds it

You build the backend service in an environment that's managed by you to support your game requirements and implement how players get into games. Integrate the AWS SDK to make calls

to the Amazon GameLift Servers service API. Optionally build your backend service on AWS using services such as AWS Lambda, Amazon Simple Storage Service (Amazon S3), and Amazon Cognito.

Game server

The game server is your custom server software that manages game state, processes player actions, and synchronizes gameplay across multiple connected players. The game server maintains communication with the Amazon GameLift Servers service to manage game session hosting.

Key functions for game hosting

- Communicate with Amazon GameLift Servers to:
 - report status (ready to host sessions, ready to accept players, health status).
 - respond to service calls (start or end game sessions).
- Manage game session lifecycle to host one game session at a time per process.
- Coordinate with other AWS services for added functionality.
- Optionally validate new player connections.

Who builds it

You build your game server software. You integrate the server SDK for Amazon GameLift Servers and add functionality to establish a connection with the service and support game session management.

Game hosting fleet

The hosting fleet is a collection of computing resources that run your game servers. Fleet resources can be distributed across multiple geographic locations to provide low-latency gameplay to players wherever they are. Each fleet resource runs one or more game server processes, which communicates directly with Amazon GameLift Servers. Each game server process can host one game session at a time.

Fleet characteristics and functionality vary based on the fleet's hosting type. Managed fleets deploy resources to the AWS Cloud and are managed by Amazon GameLift Servers. Anywhere fleets are customer-provided compute resources that are managed outside of Amazon GameLift Servers.

Key functions for game hosting

- Provision game server hosting resources.
 - Managed fleets deploy cloud-based Amazon EC2 instances with a wide range of configuration settings. A fleet configuration determines the computing power of each instance in the fleet, the physical location of instances, and other details. Managed fleets add or remove EC2 instances in response to capacity scaling
 - Anywhere fleet deployments are self-managed. The fleet can consist of physical hardware or other cloud resources, and be configured as needed.
- Install runtime environment and game server software.
 - Managed Amazon EC2 fleet instances are deployed with your game server build and an Amazon Machine Image (AMI) with a compatible runtime environment.
 - Managed container fleet instances are deployed with your game server build and a container-optimized AMI with Docker tools and other components to work with Amazon ECS.
 - Anywhere fleets are deployed with your chosen game server software and operating system.
- Manage game server process life cycle. Apply pre-configured runtime instructions to start and stop processes on each fleet resource.

Who builds it

This depends on the fleet's hosting type:

- Managed Amazon EC2 fleets: You upload your game server build to Amazon GameLift Servers, which stores it for deployment to fleet instances. Amazon GameLift Servers provides the fleet's AWS Cloud infrastructure. You configure the fleet and instruct it on how to run game servers on each fleet instance.
- Managed container fleets: You package your game server build and runtime instructions into a container image to store in Amazon ECR for deployment. Amazon GameLift Servers provides the fleet's AWS Cloud infrastructure. You provide a container architecture and configure the fleet to host your containers.
- Anywhere fleets: You provision all infrastructure and manage server software deployment. You create Anywhere fleets to connect your active game hosting resources with Amazon GameLift Servers.

Game session placement system

The game session placement system locates available game servers to host new game sessions. The system uses real-time information about game server availability to make optimal placement decisions.

In Amazon GameLift Servers, the primary game session placement mechanism is the queue. A game session queue uses algorithms, which you can configure, to place game sessions for the best possible outcome. You can prioritize placements based on factors such as lowest hosting cost and lowest player latency, and you can configure a queue to search across multiple geographic locations. As an alternative to queues, you can designate a specific fleet to host your game sessions.

Key functions for game hosting

- Process game session placement requests received from the backend service.
- Make placements based on real-time information about hosting resource availability.
- Use player latency data and other data to prioritize placement options (queues only)
- Prompt game server processes to start new game sessions.
- Update game session connection information after the game session is ready to accept players.
- Optionally set up FlexMatch matchmaking to create player matches and request game session placement for matches.

Who builds it

Amazon GameLift Servers supplies the placement system. You configure placement behavior through how your backend service makes placement requests and optionally by setting up game session queues and FlexMatch matchmakers. Use the Amazon GameLift Servers console, the AWS SDK, or the AWS CLI to create and configure queues and matchmakers for your game.

Game hosting management system

The game hosting management system is the operational backbone that coordinates and monitors all aspects of your game hosting solution. This system provides the intelligence and automation that makes Amazon GameLift Servers' placement and scaling capabilities possible.

Key functions for game hosting

- Track the real-time status and availability of game server processes, game sessions, and player sessions across all fleets to support game session placement and automatic capacity scaling.
- Monitor fleet health and performance.
- Collect and analyze game hosting activity metrics.
- Provide capacity scaling tools, including automatic scaling based on player demand and fleet utilization.
- Manage updates to game server software and runtime instructions.

Who builds it

- **Managed fleets:** Amazon GameLift Servers provides availability tracking, metrics on game hosting activity and hardware performance, and capacity scaling tools. You use available AWS tools to manage game server software updates, modify runtime instructions, and customize your use of metrics (such as setting up CloudWatch for monitoring). You configure a custom scaling policy and modify as needed.
- **Anywhere fleets:** Amazon GameLift Servers provides availability tracking and game hosting activity metrics. You manage fleet configuration changes, including updates to game server software and runtime instructions. You create systems to monitor fleet performance and manage fleet capacity scaling.

How the components work together

When a game hosting compute is deployed

- **Game server software installed:** The compute is installed with a runtime environment and your game server build
- **Game server launches:** At least one instance of the game server executable is launched, with optional launch parameters, on the compute.
- **Game server connects to Amazon GameLift Servers:** As part of its startup actions, the game server process calls the server SDK to initialize a connection to the service.
- **Game server reports ready to host a game session:** The game server process completes startup actions, then calls the server SDK to report readiness. It begins reporting health status based on its configuration.

- **Amazon GameLift Servers tracks availability:** The service records the game server process's availability to game session placement. It also updates metrics for fleets, instances, and game sessions to track usage and capacity.
- **Game server waits for game session assignment:** The game server process maintains its idle status while it waits for a prompt to start a game session.

When a player wants to join a game

- **Player initiates a game:** A player launches your game client, authenticates with your backend service, and asks to join a game.
- **Backend service requests a game session:** Your backend service calls Amazon GameLift Servers to find or create a suitable game session, based on its player grouping strategy. The request might include player or game data for use in the game session.
- **Amazon GameLift Servers places the game session:** When starting a new game session, the placement system identifies an optimal location to host the session and selects an available game server process. The service prompts the selected process to start a new game session and passes any player or game data. If FlexMatch matchmaking is in use, the matchmaker first creates a match, then requests placement for the match.
- **Game server starts the game session:** The game server initiates steps to start a game session. When complete, it reports to Amazon GameLift Servers that it's ready to accept player connections.
- **Connection information delivered to game client:** After the game server updates its status, Amazon GameLift Servers provides the game session connection information. The backend service receives this information and delivers it to the game client.
- **Player connects to the game session:** The game client uses the connection information to connect directly to the game server and begin gameplay.
- **Amazon GameLift Servers monitors game session status:** The game server process reports health status, optional player connection status, and game session status to track ongoing game session availability.
- **Game server process shuts down:** The game server process ends the game session, reports status, and then shuts itself down.

Amazon GameLift Servers and the player experience

Players expect fast, fair, and reliable multiplayer gaming experiences. Amazon GameLift Servers delivers these experiences by providing game developers with tools that directly enhance how players connect and engage with multiplayer games. This topic outlines the key player benefits Amazon GameLift Servers features provide.

Provide fast gameplay to players globally

Players want to connect to nearby servers with minimal lag. Amazon GameLift Servers ensures optimal connection experiences through intelligent placement and global infrastructure:

- **Players experience consistent performance globally** – Host games in AWS Regions and Local Zones available globally to minimize the distance between players and game servers (see [Amazon GameLift Servers service locations](#)). You can choose from a wide range of Amazon EC2 instance types in these locations to find the best computing resources to host your game workloads.
- **Players connect to the best available servers** – Amazon GameLift Servers places players in game sessions with the lowest possible latency by gathering game client network data and searching for available servers across multiple locations. Use UDP ping beacons to collect accurate latency data. Configure latency policies to balance connection quality with acceptable wait times for players.
- **Players from different regions can play together** – Game session placements can balance game sessions with players from regional player pools while maintaining acceptable latency. Players from less-populated regions can find matches without sacrificing connection quality.

Help players join games quickly

Players want fast access to games regardless of demand. Amazon GameLift Servers provides a range of capacity scaling tools that help you ensure that players can find games during peak and quiet periods:

- **Players can easily find available servers** – With automatic scaling tools, you can ensure that hosting resources are ready when they're needed. Target tracking manages a flexible buffer of capacity to anticipate player demand patterns for your game and scale proactively.

- **Players experience smooth performance during special events** – Combine auto-scaling with manual scaling techniques to handle sudden influxes after planned events, such as game updates, marketing campaigns, or tournaments.
- **Players don't have to wait for servers to start up** – On every game hosting resource, game servers are pre-warmed so that they're always ready to host new sessions as soon as players join.

Build creative ways to group players for game sessions

Players want to compete or collaborate with others in fair, balanced ways. Amazon GameLift Servers offers a number of options for grouping players into game sessions:

- **Players are grouped to optimize player experience** – You can manage player grouping to best suit your game using a range of features. Let players search or browse game sessions or group player requests when they arrive based on skill level, latency, and other attributes. You can manage player grouping on your own or you can implement FlexMatch matchmaking with customizable match rules.
- **Players can join active matches** – Build player groups to start new game sessions, or use player sessions to track available slots in existing sessions and add new players. With FlexMatch, automatically backfill sessions with well-matched players.
- **Friends can play together as a team** – Player party support lets groups of friends play together. Game sessions can fill remaining slots with other players or restrict access. With FlexMatch, friends can form teams.
- **Players can bring custom data to game sessions** – Include game and player information in requests for game sessions and player sessions. The information gets passed on to the game server for use in a game session.
- **Players don't wait indefinitely for perfect matches** – With FlexMatch, you can relax match requirements over time to prevent player frustration from excessive queuing. Create progressive relaxation rules that prioritize the most important aspects of match quality.
- **Players can choose to accept or decline matches** – With FlexMatch, you can customize workflows for letting players accept or reject prospective matches.

Deliver reliable connections throughout gameplay

Players expect consistent gameplay without interruptions or disconnections. Amazon GameLift Servers provides comprehensive session management and protection:

- **Players enjoy high-quality performance on AWS Cloud** – With hosting managed by Amazon GameLift Servers, players get fast, reliable game servers that run on AWS computing infrastructure.
- **Validate players when they connect** – When you enable player sessions for your game, you can have Amazon GameLift Servers reserve player slots in game sessions and validate players on connection.
- **Players can reconnect and maintain game progress** – Amazon GameLift Servers supports reconnection for both server-side and client-side interruptions.
- **Players' games are protected from termination** – Optional game session protection feature prevents active sessions from being terminated during scaling events or other interruptions.

Improve player experience based on real-world data

Players expect games to work consistently and get better over time. Amazon GameLift Servers provides comprehensive monitoring and analytics that help optimize the player experience:

- **Players experience fewer disruptions** – Real-time monitoring of fleet performance, game sessions, and player activity can identify issues before they significantly impact gameplay. Customize graceful game session shutdowns and migrations.
- **Players benefit from proactive server health monitoring** – Amazon GameLift Servers provides continuous server health monitoring and automatically replaces unhealthy game servers to minimize hardware or software failure impact. Configure health check parameters to support different game requirements.
- **Players get data-driven improvements in gameplay** – Take advantage of game session logs and detailed analytics to reveal patterns in player behavior and server performance. Add custom logging support for game-specific events.

Integrate enhanced game features with other AWS services

Players want integrated features like voice chat, secure authentication, and persistent progress. Amazon GameLift Servers can be integrated with other AWS services to provide comprehensive gaming experiences:

- **Players get streamlined authentication** – Amazon Cognito integration supports various identity providers and authentication methods while maintaining secure player identities across game sessions, with streamlined login processes for returning players.

- **Players can persist their game progress across sessions** – Amazon DynamoDB integration can be used to store player progression, inventories, and persistent data with high-performance access that ensures minimal gameplay impact while supporting cross-session continuity.
- **Players benefit from analytic insights** – Amazon Kinesis and Amazon Simple Storage Service (Amazon S3) integration processes game analytics to gather insights on player behavior and preferences, enabling real-time analytics that help developers adapt to changing player patterns and keep games fresh and engaging.

Managing game hosting in production with Amazon GameLift Servers

After you've launched your game with Amazon GameLift Servers, you'll need to manage your game hosting infrastructure to ensure optimal performance, reliability, and player experience. This topic covers the key features and tools for managing game hosting lifecycles during production.

Monitoring hosting health and performance

Amazon GameLift Servers provides comprehensive monitoring tools to help you track the health and performance of your game hosting infrastructure:

- **Performance metrics** – Monitor key performance indicators for managed fleets, such as instance utilization, player sessions, and game session placements. Track these metrics in Amazon GameLift Servers or in Amazon CloudWatch, where you can also set up alarms to get notifications when metrics exceed thresholds.
- **Fleet metrics and events** – Track fleet-specific metrics including active server processes, available game sessions, and active player sessions to ensure optimal capacity.
- **Game session placement metrics and events** – Track metrics and events to monitor the health and performance of your game session placement system.
- **Game session logs** – Access and analyze game session logs to identify issues and understand player behavior patterns.
- **Realtime script logs** – For games using Real-time Servers, monitor script execution and performance through detailed logs.
- **AWS Health Dashboard** – Stay informed about AWS service health that might affect your Amazon GameLift Servers deployments.

You can access these monitoring tools through the Amazon GameLift Servers console, AWS CLI, or use the AWS SDK for Amazon GameLift Servers to create custom dashboards and monitoring solutions.

Managing game server updates and patches

Keeping your game servers updated is critical for security, performance, and adding new features. Amazon GameLift Servers provides several approaches for managing updates:

- **Build management** - Upload and manage multiple versions of your game server builds. Each build is versioned and can be deployed to different fleets.
- **Fleet replacement** - Create new fleets with AMI versions and game server build updates, and gradually shift traffic from old fleets to new ones using game session placement queues and aliases.
- **Script updates** - For Real-time Servers, update server scripts without replacing the entire fleet by uploading new script versions.
- **Container updates** - For container-based deployments, update container images and definitions. Deploy new versions of your game servers to existing fleets. Replace fleets to update AMI versions.
- **Automated deployments** - Use AWS CodePipeline and AWS CodeDeploy to create CI/CD pipelines for automated game server updates.

When planning updates, consider using blue/green deployment strategies to minimize disruption to active players and allow for quick rollbacks if issues are detected.

Optimizing performance and scaling

As your player base evolves, you'll need to adjust your hosting configuration to maintain optimal performance and cost-efficiency:

- **Auto scaling** - Configure fleet scaling policies based on metrics like player count or game session utilization to automatically adjust capacity.
- **Regional deployment adjustments** - Add or remove regions from your multi-region deployments based on player demographics and latency requirements.
- **Queue management** - Optimize game session placement queue configurations to balance player experience and hosting costs.

- **Instance type selection** - Analyze performance metrics to determine the most cost-effective instance types for your game server requirements.
- **Spot Instance usage** - Leverage Spot Instances for non-critical workloads to reduce costs, with appropriate fallback strategies to On-Demand instances.
- **FlexMatch tuning** - Refine matchmaking rules and algorithms based on actual player data and feedback.

Regularly review CloudWatch metrics and cost reports to identify optimization opportunities and implement changes through the Amazon GameLift Servers console or API.

Troubleshooting and live operations

Effective troubleshooting and live operations management are essential for maintaining a positive player experience:

- **Fleet event notifications** - Set up Amazon Simple Notification Service notifications for fleet events such as scaling activities, instance terminations, or game session placement failures.
- **Game session placement debugging** - Use detailed placement logs to identify and resolve issues with game session placements.
- **Server process health checks** - Monitor server process health and automatically replace processes that fail health checks.
- **Remote access** - Connect to fleet instances for direct troubleshooting using AWS Systems Manager Session Manager.
- **Alias management** - Use aliases to quickly redirect player traffic away from problematic fleets without changing client configurations.
- **Backup and recovery** - Implement regular backups of critical game data and configuration to enable quick recovery from failures.

Establish clear incident response procedures and runbooks for common issues to minimize downtime and impact on players.

Tools and integration

Amazon GameLift Servers integrates with various AWS services and third-party tools to enhance your production management capabilities:

- **AWS CloudFormation** - Define and manage your GameLift resources as infrastructure as code for consistent deployments.
- **AWS Lambda** - Create serverless functions to automate routine management tasks and respond to events.
- **Amazon EventBridge** - Build event-driven architectures that respond automatically to changes in your Amazon GameLift Servers environment.
- **AWS SDK integration** - Use AWS SDKs to build custom management tools and dashboards tailored to your specific needs.
- **Third-party monitoring** - Integrate with third-party monitoring and analytics platforms using CloudWatch metrics export.

Leverage these integrations to create a comprehensive management solution that aligns with your team's workflows and requirements.

Related resources

- [Monitoring Amazon GameLift with Amazon CloudWatch](#)
- [Updating Your Amazon GameLift Fleets](#)
- [Scaling Game Hosting Capacity with Amazon GameLift](#)
- [Amazon GameLift Server SDK Reference](#)

Amazon GameLift Servers key terms

This topic provides definitions for key technical terms used in Amazon GameLift Servers documentation and services.

Active game session

A game session that is currently running and accepting or hosting players. Used as a metric to monitor fleet utilization and capacity planning. See [Monitor Amazon GameLift Servers with Amazon CloudWatch](#).

Alias

A friendly identifier for a fleet that provides an abstraction layer between your game client and fleet resources. Aliases allow you to update fleet configurations without changing your game client code. See [Abstract an Amazon GameLift Servers fleet designation with an alias](#).

AMI (Amazon Machine Image)

A pre-configured virtual machine image used to create Amazon EC2 instances for managed fleets. Contains the operating system, runtime libraries, and Amazon GameLift Servers agent software needed to run game servers. See [Amazon GameLift Servers managed EC2 fleets](#).

Anywhere fleet

A type of fleet that uses customer-provided compute resources managed outside of Amazon GameLift Servers. Anywhere fleets allow you to integrate existing infrastructure or hybrid cloud solutions with Amazon GameLift Servers game session management. See [Amazon GameLift Servers Anywhere fleets](#).

Authentication token

A security credential generated by Amazon GameLift Servers that authenticates game server processes running on Anywhere compute resources. Required for server processes to communicate with the Amazon GameLift Servers service. See [Amazon GameLift Servers Anywhere fleets](#).

Backend service

A coordination layer between game clients and the Amazon GameLift Servers service that controls communication for game session placement requests and retrieves game session information. Acts as a secure intermediary to protect hosting resources from malicious attacks. See [How hosting with Amazon GameLift Servers works](#).

Game server build

A packaged version of your game server software that includes all files needed to run your game server on Amazon GameLift Servers hosting resources. Contains the game server executable, dependencies, and assets. See [Package a game server build for deployment](#).

Compute

In container fleets, represents a copy of a game server container group on a fleet instance. Each compute can host one game session at a time. See [How containers work in Amazon GameLift Servers](#).

Container fleet

A type of managed fleet that runs containerized game servers using container images instead of traditional builds. Provides flexibility for deploying game servers with specific runtime environments and dependencies. See [Amazon GameLift Servers managed container fleets](#).

Container group

A collection of containers that work together to provide game server functionality. Similar to a container task or pod, defining how containers behave, dependencies, and resource sharing within a container fleet. See [How containers work in Amazon GameLift Servers](#).

Destination

In game session queues, a fleet or alias that can receive game session placement requests. Queues can have multiple destinations with different priorities and latency policies. See [Customize a game session queue](#).

Fleet

A collection of computing resources that run your game servers. Fleet resources can be distributed across multiple geographic locations and each resource runs one or more game server processes. See [Deploy hosting fleets for Amazon GameLift Servers](#).

Fleet location

A specific AWS Region or Local Zone where fleet instances are deployed. Multi-location fleets can span multiple locations to reduce latency and improve availability. See [Deploy hosting fleets for Amazon GameLift Servers](#).

FlexMatch

Amazon GameLift Servers matchmaking service that automatically forms player matches based on configurable rules and requests game session placement for completed matches. Supports skill-based matching, team balancing, and match backfilling. See [What is GameLift FlexMatch?](#)

Game client

Your game software that runs on a player's device. Initiates player placement into game sessions by communicating with a backend service and connects directly to game servers for gameplay. See [How hosting with Amazon GameLift Servers works](#).

Game property

Key-value pairs of custom data associated with a game session. Used to pass configuration information, game rules, or other metadata to game servers and clients. See [GameProperty](#) in the API Reference.

Game server

Your custom server software that manages game state, processes player actions, and synchronizes gameplay across multiple connected players. Integrates with the Amazon

GameLift Servers Server SDK to communicate with the service. See [Add Amazon GameLift Servers to your game server with the server SDK](#).

Game server activation

The process of starting and initializing a game server process on a hosting resource. Includes loading the game server executable, establishing communication with Amazon GameLift Servers, and preparing to host game sessions. See [Add Amazon GameLift Servers to your game server with the server SDK](#).

Game server container

A container within a game server container group that runs the game server application. Automatically designated as essential to the container group lifecycle. See [How containers work in Amazon GameLift Servers](#).

Game server container group

A type of container group that manages containers running game server applications and supporting software. Required for container fleets to host game sessions and can be replicated across fleet instances. See [How containers work in Amazon GameLift Servers](#).

Game server process

A running instance of your game server executable on a hosting resource. Each process can host one game session at a time and communicates directly with Amazon GameLift Servers to report status and receive instructions. See [Add Amazon GameLift Servers to your game server with the server SDK](#).

Game session

An instance of your multiplayer game running on a game server process. Represents a single gameplay experience that multiple players can join and participate in together. See [How hosting with Amazon GameLift Servers works](#) and [GameSession](#) in the API Reference.

Game session data

Information associated with a game session including player data, game state, session properties, and other runtime information used by game servers and clients. See [GameSession](#) in the API Reference.

Game session placement

The process of locating available game servers to host new game sessions. Uses real-time information about game server availability to make optimal placement decisions based on

factors like cost and latency. See [Integrate Amazon GameLift Servers game client functionality](#) and [StartGameSessionPlacement](#) in the API Reference.

Game session protection

A setting that prevents Amazon GameLift Servers from terminating game server processes that are hosting active game sessions during scaling events or fleet updates. Ensures ongoing games are not interrupted. See [Manually set capacity for an Amazon GameLift Servers fleet](#).

Game session queue

A placement mechanism that uses configurable algorithms to place game sessions for optimal outcomes. Can prioritize placements based on factors such as lowest hosting cost and lowest player latency across multiple geographic locations. See [Configure game session placement](#).

Idle instance

A fleet instance that is running but not currently hosting any active game sessions. Used as a metric for monitoring fleet efficiency and scaling decisions. See [Monitor Amazon GameLift Servers with Amazon CloudWatch](#).

Inbound permissions

Network access rules that define which IP addresses and port ranges can be used to connect to fleet instances. Controls access to game servers and ensures secure connections. Both managed EC2 and managed container fleets require inbound permissions configuration.

See [IpPermission](#) and [Configure network connections](#).

Latency policy

Rules in game session queues that define acceptable latency thresholds for player connections. Used to ensure players are placed in game sessions with acceptable network performance. See [Customize a game session queue](#).

Managed fleet

A type of fleet where Amazon GameLift Servers deploys and manages computing resources in the AWS Cloud. Provides automatic scaling, health monitoring, and infrastructure management. See [Amazon GameLift Servers managed EC2 fleets](#).

Matchmaking

The process of grouping players together for multiplayer gameplay based on criteria such as skill level, latency, or other game-specific factors. In Amazon GameLift Servers, primarily handled by FlexMatch. See [What is GameLift FlexMatch?](#)

Metric group

A collection of related performance metrics used for monitoring and auto-scaling decisions. Groups metrics like active game sessions, available game servers, and player utilization. See [Auto-scale fleet capacity with Amazon GameLift Servers](#).

Per-instance container group

An optional container group type that runs additional software on each fleet instance. Useful for background services or monitoring tools, with only one copy deployed per instance. See [How containers work in Amazon GameLift Servers](#).

Player session

A connection slot for a player in a game session. Used to reserve game session slots, validate players when they connect, and track player connection status and disconnects. See [Integrate Amazon GameLift Servers game client functionality](#) and [PlayerSession](#) in the API Reference.

Priority configuration

Settings that customize the prioritization order of a game session queue when selecting destinations for game session placement. Defines the overall queue behavior, in contrast to priority override lists which modify priority for individual placement requests. See [Prioritize game session placement](#).

Priority override list

An alternate priority list of locations that can be included in a game session placement request. Effectively replaces the queue's configured prioritization for locations for that one request only, without impacting other requests. See [Prioritize game session placement](#).

Process manager

A component that manages the lifecycle of multiple game server processes within a container or instance. Handles starting, stopping, and monitoring individual game server processes. See [Optimize game server runtime configuration on managed Amazon GameLift Servers](#).

Realtime script

JavaScript code that defines game logic for Amazon GameLift Servers Realtime Servers. Handles player connections, message routing, and game state management without requiring a custom game server. See [What is Amazon GameLift Realtime Servers?](#)

Routing strategy

Configuration that determines how aliases direct traffic to fleets. Can route to a specific fleet or distribute traffic across multiple fleets based on defined rules. See [Abstract an Amazon GameLift Servers fleet designation with an alias](#).

Runtime configuration

Settings that define how game server processes run on fleet instances, including the number of concurrent processes, launch parameters, and resource allocation. See [Optimize game server runtime configuration on managed Amazon GameLift Servers](#).

Scaling

The process of adjusting fleet capacity by adding or removing hosting resources based on player demand. Can be automatic (based on utilization metrics) or manual (administrator-controlled). See [Auto-scale fleet capacity with Amazon GameLift Servers](#).

Scaling policy

Rules that define when and how to automatically adjust fleet capacity. Can be target-based (maintain specific utilization) or rule-based (scale based on metric thresholds). See [Auto-scale fleet capacity with Amazon GameLift Servers](#).

Server SDK

Software development kit that provides the tools and libraries needed to integrate your game server with Amazon GameLift Servers. Enables communication between your game server and the service for session management, health reporting, and status updates. See [Add Amazon GameLift Servers to your game server with the server SDK](#).

Session-based multiplayer

A multiplayer game architecture where gameplay occurs within discrete, time-limited sessions. Each session has a defined start and end, with players joining and leaving as needed. See [How hosting with Amazon GameLift Servers works](#).

Spot instances

A type of Amazon EC2 instance that uses spare AWS capacity at reduced cost. Spot instances can be interrupted when AWS needs the capacity back, making them suitable for fault-tolerant game workloads. See [Build a queue for Spot Instances](#).

TLS certificate

A digital certificate used to enable encrypted communication between game clients and servers. Required for TLS-enabled fleets to ensure secure data transmission. See [Create an Amazon GameLift Servers managed EC2 fleet](#).

Getting started with Amazon GameLift Servers

Take advantage of these getting started resources to learn more about the Amazon GameLift Servers service and how you can start developing a custom hosting solution for your session-based multiplayer games.

Before you start

- Create an AWS account (or designate an existing one) to use with Amazon GameLift Servers.
- Set up users with permissions for Amazon GameLift Servers and related AWS services.
- Select an AWS Region to work in. For development, choose a Region that's close to your location. You can change Regions at any time.

[Set up an AWS user account](#)

Choose your path

I want to explore Amazon GameLift Servers quickly

Best for: learning, creating a proof of concept, rapid prototyping

Quick start options:

- [Game server wrapper for Amazon GameLift Servers](#) – See [Tutorial: Quick onboarding with the Amazon GameLift Servers wrapper](#). This tool is the quickest and easiest way to get your game server hosted and running game sessions with Amazon GameLift Servers, with no changes to game code required. The game server wrapper offers basic game session management functionality and streamlined game server deployment. It's ideal for doing a hands-on evaluation of Amazon GameLift Servers using your own game project or a sample project. When you're ready to build a custom game hosting solution, switch to one of the custom development options with full integration with the server SDK for Amazon GameLift Servers. If your game doesn't need a custom hosting solution, you can continue to use the game server wrapper to deploy and host your game servers in production.
- [Amazon GameLift Servers plugin for Unreal Engine or Unity](#) – The plugins give you GUI workflows and sample assets to guide you through the initial steps and deploy your game server

with a basic hosting solution. Use the plugin to set up hosting with self-managed Anywhere fleets, or deploy cloud-based, managed EC2 fleets or container fleets. When you're ready to develop a custom hosting solution, you can build on your plugin-built solutions.

- [Starter kit for Amazon GameLift Servers managed containers](#) – This kit streamlines tasks to integrate a game server, prepare a game server container image, and deploy a container fleet for hosting. For integration, the kit adds essential game session management features to your game server. The kit uses pre-configured templates to build a container fleet and an automated deployment pipeline for the game server. When you're ready to add full game session management features, follow one of the custom development roadmaps to integrate the server SDK for Amazon GameLift Servers.

I want to build a custom hosting solution

Best for: creating a production-level solution for a custom game server

Review the page [Amazon GameLift Servers game hosting options](#) and choose one of the options for your solution. Follow one of these development roadmaps to get started building a full-featured custom hosting solution for your game. The roadmaps provide detailed guidance on how to create, test, and customize each component in your hosting solution.

- [Development roadmap for hosting with Amazon GameLift Servers managed EC2](#)
- [Development roadmap for hosting with Amazon GameLift Servers managed containers](#)
- [Development roadmap for hosting with Amazon GameLift Servers Anywhere](#)
- [Development roadmap for hybrid hosting with Amazon GameLift Servers](#)

Recommended learning path

1. Start small. Try the game server wrapper or the plugin for Unreal.
2. Understand the concepts. Review the page [How hosting with Amazon GameLift Servers works](#).
3. Choose architecture. Select your hosting model based on your game's requirements.
4. Build and test. Follow the appropriate development roadmap. Build a basic version of each component and then iterate and customize.
5. Scale and optimize. Optimize your solution for production-level usage. Add advanced features such as matchmaking.

Set up an AWS user account

Tip

Use these topics to get help with these tasks:

- Get a new AWS account for use with Amazon GameLift Servers.
- Create a user or group with permissions to work with Amazon GameLift Servers resources.
- Set up security credentials (you need these to use the AWS CLI tools and the Amazon GameLift Servers plugins for Unreal and Unity)
-

As with all AWS services, you need an AWS account to use the Amazon GameLift Servers service and tools. An AWS account serves two primary functions: (1) it gives you a container for all the AWS resources that you create with the account; and (2) it lets you manage security for your AWS resources, including setting up user authentication and controlling user access permissions. There's no cost for creating an AWS account.

Explore Amazon GameLift Servers with or without an AWS account

You **don't** need an AWS account to:

- Discover AWS tools for building, running, and growing game experiences at [AWS for Games](#). Read the [Blog](#) and browse the [Solutions for Games library](#).
- Learn more about Amazon GameLift Servers in the [product overview, FAQs, and resources](#). **Ask AWS** to find answers to your product questions. (Try this one: "Looking for low-cost options to host my multiplayer game".)
- For a deeper dive, find out what makes Amazon GameLift Servers work in the [technical documentation](#), including developer guides for hosting and matchmaking, and the service API reference guide.
- Check out information on [Amazon GameLift Servers pricing](#) and cost optimization techniques. Try the [Pricing Calculator](#) to see how hosting costs are calculated based on peak concurrent player usage (CCU).
- Get downloads and see code repositories for Amazon GameLift Servers SDKs, plugins, and toolkits. See [Amazon GameLift Servers Getting started](#). (You need an AWS account to use them.)

You **do** need an AWS account to:

- Follow onboarding workflows with the Amazon GameLift Servers plugins for Unreal and Unity, or use the game server wrapper.
- Create and manage AWS resources using the AWS Management Console.
- Create and manage AWS resources using the AWS Command Line Interface.
- Use Amazon Q with the In the Amazon GameLift Servers technical documentation to find answers, guidance, and recommendations.

Topics

- [Sign up for an AWS account](#)
- [Set user permissions for Amazon GameLift Servers](#)
- [Set up programmatic access for users](#)
- [Set up programmatic access for your game](#)
- [IAM permission examples for Amazon GameLift Servers](#)
- [Set up an IAM service role for Amazon GameLift Servers](#)

Sign up for an AWS account

To get started with AWS, you need an AWS account. For information about creating an AWS account, see [Getting started with an AWS account](#) in the *AWS Account Management Reference Guide*.

Set user permissions for Amazon GameLift Servers

Create additional users or extend access permissions to existing users as needed for your Amazon GameLift Servers resources. As a best practice ([Security best practices in IAM](#)), apply least-privilege permissions for all users. For guidance on permissions syntax, see [IAM permission examples for Amazon GameLift Servers](#).

Use following instructions to set user permissions based on how you manage the users in your AWS account.

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

Create a permission set. Follow the instructions in [Create a permission set](#) in the *AWS IAM Identity Center User Guide*.

- Users managed in IAM through an identity provider:

Create a role for identity federation. Follow the instructions in [Create a role for a third-party identity provider \(federation\)](#) in the *IAM User Guide*.

- IAM users:
 - Create a role that your user can assume. Follow the instructions in [Create a role for an IAM user](#) in the *IAM User Guide*.
 - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in [Adding permissions to a user \(console\)](#) in the *IAM User Guide*.

When working with IAM users, as a best practice always attach permissions to roles or user groups, not individual users.

Set up programmatic access for users

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
IAM	(Recommended) Use console credentials as temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Login for AWS local development in the <i>AWS Command Line Interface User Guide</i>. • For AWS SDKs, see Login for AWS local development

Which user needs programmatic access?	To	By
		in the <i>AWS SDKs and Tools Reference Guide</i> .
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. - For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the <i>AWS Command Line Interface User Guide</i>. - For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the <i>AWS SDKs and Tools Reference Guide</i>.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the <i>IAM User Guide</i> .

Which user needs programmatic access?	To	By
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. - For the AWS CLI, see Authenticating using IAM user credentials in the <i>AWS Command Line Interface User Guide</i>. - For AWS SDKs and tools, see Authenticate using long-term credentials in the <i>AWS SDKs and Tools Reference Guide</i>. - For AWS APIs, see Managing access keys for IAM users in the <i>IAM User Guide</i>.

If you use access keys, see [Best practices for managing AWS access keys](#).

Set up programmatic access for your game

Most games use backend services to communicate with Amazon GameLift Servers using the AWS SDKs. Use a backend service (acting for a game client) to request game sessions, place players into games, and other tasks. These services need programmatic access and security credentials to authenticate calls to the service API for Amazon GameLift Servers.

For Amazon GameLift Servers, you manage this access by creating a player user in AWS Identity and Access Management (IAM). Manage player user permissions through one of the following options:

- Create an IAM role with player user permissions and allow the player user to assume the role when needed. The backend service must include code to assume this role before making requests

to Amazon GameLift Servers. In accordance with security best practices, roles provide limited, temporary access. You can use roles for workloads running on AWS resources ([IAM roles](#)) or outside of AWS ([IAM Roles Anywhere](#)).

- Create an IAM user group with player user permissions and add your player user to the group. This option gives your player user long-term credentials, which the backend service must store and use when communicating with Amazon GameLift Servers.

For permissions policy syntax, see [Player user permission examples](#).

For more information on managing permissions for use by a workload, see [IAM Identities: Temporary credentials in IAM](#).

IAM permission examples for Amazon GameLift Servers

Use the syntax in these examples to set AWS Identity and Access Management (IAM) permissions for users that need access to Amazon GameLift Servers resources. For more information on managing user permissions, see [Set user permissions for Amazon GameLift Servers](#). When managing permissions for users outside of the IAM Identity Center, as a best practice always attach permissions to IAM roles or user groups, not individual users.

If you're using Amazon GameLift Servers FleetIQ as a standalone solution, see [Set up your AWS account for Amazon GameLift Servers FleetIQ](#).

Administration permission examples

These examples give a hosting administrator or developer targeted access to manage Amazon GameLift Servers game hosting resources.

Example Syntax for Amazon GameLift Servers full access resource permissions

The following example extends full access to all Amazon GameLift Servers resources.

JSON

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

```
"Action": "gamelift:*",
"Resource": "*"
}
}
```

Example Syntax for Amazon GameLift Servers resource permissions with support for Regions that aren't enabled by default

The following example extends access to all Amazon GameLift Servers resources and AWS Regions that aren't enabled by default. For more information about Regions that aren't enabled by default and how to enable them, see [Managing AWS Regions](#) in the *AWS General Reference*.

JSON

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

Example Syntax for Amazon GameLift Servers resource to access container images in Amazon ECR

The following example extends access to Amazon Elastic Container Registry (Amazon ECR) actions that Amazon GameLift Servers users need when working with managed container fleets.

JSON

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

```

        "ecr:DescribeImages",
        "ecr:BatchGetImage",
        "ecr:GetDownloadUrlForLayer"
    ],
    "Resource": "*"
}
}

```

Example Syntax for Amazon GameLift Servers resource and PassRole permissions

The following example extends access to all Amazon GameLift Servers resources and allows a user to pass an IAM service role to Amazon GameLift Servers. A service role gives Amazon GameLift Servers limited ability to access other resources and services on your behalf, as is described in [Set up an IAM service role for Amazon GameLift Servers](#). For example, when responding to a CreateBuild request, Amazon GameLift Servers needs access to your build files in an Amazon S3 bucket. For more information about the PassRole action, see [IAM: Pass an IAM role to a specific AWS service](#) in the *IAM User Guide*.

JSON

```

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

```

Player user permission examples

These examples allow a backend service or other entity to make API calls to the Amazon GameLift Servers API. They cover the common scenarios for managing game sessions, player sessions, and matchmaking. For more details, see [Set up programmatic access for your game](#).

Example Syntax for game session placement permissions

The following example extends access to the Amazon GameLift Servers APIs that use game session placement queues to create game sessions and manage player sessions.

JSON

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

Example Syntax for matchmaking permissions

The following example extends access to the Amazon GameLift Servers APIs that manage FlexMatch matchmaking activities. FlexMatch matches players for new or existing game sessions and initiates game session placement for games hosted on Amazon GameLift Servers. For more information about FlexMatch, see [What is Amazon GameLift Servers FlexMatch?](#)

JSON

```
{
```

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

Example Syntax for manual game session placement permissions

The following example extends access to the Amazon GameLift Servers APIs that manually create game sessions and player sessions on specified fleets. This scenario supports games that don't use placement queues, such as games that let players join by choosing from a list of available game sessions (the "list-and-pick" method).

JSON

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

Set up an IAM service role for Amazon GameLift Servers

Some Amazon GameLift Servers features require you to extend limited access to other AWS resources that you own. You can do this by creating an AWS Identity and Access Management (IAM) role. An [IAM role](#) is an IAM identity that you can create in your account that has specific permissions. An IAM role is similar to an IAM user in that it is an AWS identity with permissions policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it. Also, a role does not have standard long-term credentials such as a password or access keys associated with it. Instead, when you assume a role, it provides you with temporary security credentials for your role session.

This topic covers how to create a role that you can use with your Amazon GameLift Servers managed fleets. If you use Amazon GameLift Servers FleetIQ to optimize game hosting on your Amazon Elastic Compute Cloud (Amazon EC2) instances, see [Set up your AWS account for Amazon GameLift Servers FleetIQ](#).

In the following procedure, create a role with a custom permissions policy and a trust policy that allows Amazon GameLift Servers to assume the role.

Create an IAM service role for an Amazon GameLift Servers managed EC2 fleet

Step 1: Create a permissions policy.

Use the instructions and examples on this page to create a custom permissions policy for the type of Amazon GameLift Servers fleet you're working with.

To use the JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane on the left, choose **Policies**.

If this is your first time choosing **Policies**, the **Welcome to Managed Policies** page appears. Choose **Get Started**.

3. At the top of the page, choose **Create policy**.
4. In the **Policy editor** section, choose the **JSON** option.
5. Enter or paste a JSON policy document. For details about the IAM policy language, see [IAM JSON policy reference](#).

6. Resolve any security warnings, errors, or general warnings generated during [policy validation](#), and then choose **Next**.

 Note

You can switch between the **Visual** and **JSON** editor options anytime. However, if you make changes or choose **Next** in the **Visual** editor, IAM might restructure your policy to optimize it for the visual editor. For more information, see [Policy restructuring](#) in the *IAM User Guide*.

7. (Optional) When you create or edit a policy in the AWS Management Console, you can generate a JSON or YAML policy template that you can use in CloudFormation templates.

To do this, in the **Policy editor** choose **Actions**, and then choose **Generate CloudFormation template**. To learn more about CloudFormation, see [AWS Identity and Access Management resource type reference](#) in the *AWS CloudFormation User Guide*.

8. When you are finished adding permissions to the policy, choose **Next**.
9. On the **Review and create** page, enter a **Policy name** and a **Description** (optional) for the policy that you are creating. Review **Permissions defined in this policy** to see the permissions that are granted by your policy.
10. (Optional) Add metadata to the policy by attaching tags as key-value pairs. For more information about using tags in IAM, see [Tags for AWS Identity and Access Management resources](#) in the *IAM User Guide*.
11. Choose **Create policy** to save your new policy.

Step 2: Create a role that Amazon GameLift Servers can assume.

To create an IAM role

1. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
2. On the **Select trusted entity** page, choose the **Custom trust policy** option. This selection opens the **Custom trust policy** editor.
3. Replace the default JSON syntax with the following, and then choose **Next** to continue.

JSON

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

4. On the **Add permissions** page, locate and select the permissions policy that you created in Step 1. Choose **Next** to continue.
5. On the **Name, review and create** page, enter a **Role name** and a **Description** (optional) for the role that you are creating. Review the **Trust entities** and **Added permissions**.
6. Choose **Create role** to save your new role.

Create an IAM role for Amazon GameLift Servers managed containers

If you're using Amazon GameLift Servers managed containers, you need to create an IAM service role for use with a container fleet. This role grants limited permissions that Amazon GameLift Servers needs to manage your container fleet resources and take actions on your behalf.

To create an IAM role for a container fleet

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
3. On **Select trusted entity** page, choose **AWS service** and select the **Use case** "GameLift". Choose **Next**.
4. On **Add permissions**, choose the managed policy `GameLiftContainerFleetPolicy`. Choose **Next**. See [AWS managed policies for Amazon GameLift Servers](#) for more information about this policy.

5. On **Name, review, and create**, enter a role name and choose **Create role** to save the new role.

Permission policy syntax

- **Permissions for Amazon GameLift Servers to assume the service role**

JSON

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

- **Permissions to access AWS Regions that aren't enabled by default**

JSON

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

```
}
```

Get Amazon GameLift Servers development tools

Amazon GameLift Servers provides a set of SDKs and other tools to help you build game hosting solutions for your games. The SDKs add functionality to game servers, game clients, and backend services that enables them to interact with the Amazon GameLift Servers service. For the latest information about Amazon GameLift Servers SDK versions and compatibility, see [Amazon GameLift Servers release notes](#).

For game servers

Integrate and build your 64-bit game servers with the server SDK for Amazon GameLift Servers. The game server uses the server SDK to communicate with the Amazon GameLift Servers service for game session management, including starting, updating, and stopping game sessions. For help with integrating the server SDK into your game projects, see [Prepare a game for hosting with Amazon GameLift Servers](#).

Development support

- **Development OS**
 - Windows
 - Linux
- **Programming languages**

[Get the Amazon GameLift Servers SDK](#). For version-specific information and install instructions, see the included readme files in each package.

- [C++ server SDK](#)
 - [Server SDK reference](#)
 - [How to integrate](#)
- [C# server SDK](#) (Support for .NET 4, .NET 6, .NET 8 varies by version, see [SDK versions](#))
 - [Server SDK reference](#)
 - [How to integrate](#)
- [Go server SDK](#)
 - [Server SDK reference](#)

- [How to integrate](#)
- **Game engine support**

The full-featured plugin for Amazon GameLift Servers includes UI workflows and sample assets, as well as built-in versions of the AWS SDK and server SDK. The workflows guide you through how to configure and deploy your game server for hosting with managed EC2 fleets, managed container fleets, or self-managed Anywhere fleets. If you don't need the guided workflows, you can get just the server SDK for your game engine from the same GitHub repositories.

If you're using another game engine or development environment that the plugin doesn't support, get the server SDK for your programming language and add it to your game project.

For version-specific information and install instructions, see the included readme files in each package.

- [Plugin for Unreal Engine](#) – Built for use with Unreal Engine versions 5.0, 5.1, 5.2, 5.3, 5.4, 5.5, and 5.6. Check version-specific readme files for Unreal support.
 - [Plugin guide for Unreal Engine](#)
 - [C++ \(Unreal\) server SDK 5.x for Amazon GameLift Servers -- Actions](#)
- [Plugin for Unity](#) – Built for use with LTS versions of Unity Editor 6.0, 2022.3, or 2021.3. It supports Unity's .NET Framework and .NET Standard profiles, with .NET Standard 2.1 and .NET 4.x. Check version-specific readmes for Unity support.
 - [Plugin guide for Unity](#)
 - [C# server SDK reference](#)
- [Server SDK for Unreal](#)
 - [Server SDK reference](#)
 - [Integrate Amazon GameLift Servers into an Unreal Engine project](#)
- [Server SDK for Unity](#)
 - [Server SDK reference](#)
 - [Integrate Amazon GameLift Servers into a Unity project](#)

Runtime support

For a managed hosting solution, build your game server to run on one of the following Amazon machine images (AMIs). See [Amazon GameLift Servers AMI versions](#) for Amazon GameLift Servers for more details.

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

Note

Amazon Linux 2 (AL2) will reach end of support on June 30, 2026. See more details in the [Amazon Linux 2 FAQs](#). For game servers that are hosted on AL2 and use Amazon GameLift Servers server SDK 4.x, first update the game server build to server SDK 5.x, and then deploy to AL2023 instances. See [Migrate to server SDK 5.x for Amazon GameLift Servers](#).

Additional tools

[Game server wrapper for Amazon GameLift Servers](#)

This tool helps you deploy a game server for hosting with a set of basic game session management functionality. With this tool, you don't need to make changes to your game code or integrate the server SDK for Amazon GameLift Servers. Use the game server wrapper to package your game server and deploy it for game hosting using any of the three Amazon GameLift Servers hosting solutions (Anywhere, managed EC2, or managed containers). This tool is best suited for early evaluation or prototyping with your own game or a sample game, as this tool doesn't support game server customization. If your game doesn't need custom features, you can deploy your game server with the game server wrapper for production hosting.

[Amazon GameLift Servers Toolkit](#)

The Amazon GameLift Servers Toolkit is a collection of scripts and other tools that we've developed to help developers with common scenarios and issues. Toolkit materials include scripts, sample code, and readmes.

- [Containers starter kit](#) – Use this tool to streamline the tasks of setting up game server builds for hosting with Amazon GameLift Servers managed containers. The kit integrates essential game session management features into a game server, and uses pre-configured templates to create a container fleet and set up an automated deployment pipeline for your game server build. After deployment, you can monitor fleet performance, manage game sessions, and analyze metrics using the Amazon GameLift Servers console and API tools. The kit integrates with

AWS CodeBuild for build automation, Amazon Simple Storage Service for storage, and AWS CloudFormation for infrastructure deployment.

- [Fast build update tool](#) – Use this tool to modify a game server build that's already deployed to a managed EC2 fleet. The tool is built to help you quickly swap out game build files without having to configure and create new EC2 fleets with every change. You can update individual instances or all instances in the fleet. Options let you replace specific build files or an entire build, and let you manage how to restart game servers after the updates.

For game clients

Integrate your game client with the client SDK for Amazon GameLift Servers. The client SDK provides utilities for game clients to use Amazon GameLift Servers features including player gateway and UDP ping beacons.

Get the client SDK

- [Client SDK for C++](#) – Includes player gateway and UDP ping beacons utilities with built-in endpoint selection algorithms, plus a sample application demonstrating integration. Requires C++17 and a threading library.
- [Client SDK for Unreal Engine](#) – Includes player gateway (with or without engine modifications) and UDP ping beacons utilities with built-in endpoint selection algorithms. Requires source-built Unreal Engine 5.1 or later. Access requires membership in the [Epic Games organization on GitHub](#).
- [Client SDK for C#](#) – Includes player gateway and UDP ping beacons utilities with built-in endpoint selection algorithms. Works in any .NET Standard 2.1-compatible project.
- [Client SDK for Unity](#) – Builds on the C# client SDK to provide player gateway and UDP ping beacons integration for Unity game clients. Requires Unity 2021.3 LTS or later.

For integration guides, see the documentation in each GitHub repository.

For game backend services

Create a 64-bit backend service for your game and integrate it with functionality from the AWS SDK, which includes the service API for Amazon GameLift Servers. Use the backend service to handle interactions with the Amazon GameLift Servers service, including starting or finding game sessions and joining players to games.

[Get the AWS SDK](#)

For more information about using the AWS SDK with Amazon GameLift Servers, see the following resources:

- [Amazon GameLift Servers API Reference](#)
- Integrate game client functionality
 - [Client-side service integration for games with custom game server builds](#)
 - [Client-side service integration for games with Realtime servers](#)
- [Design a backend service](#)

For Amazon GameLift Servers resource management

Use the following tools to create, update, and monitor your Amazon GameLift Servers managed hosting resources.

- [AWS Management Console](#) – The AWS Console is a web-based application that provides centralized access to all individual AWS service consoles, including Amazon GameLift Servers. Use the Console to create or sign into an AWS account and open the Amazon GameLift Servers console to work with your game hosting resources. You can configure and deploy hosting fleets and other resources, view usage and performance metrics, track resources in the dashboard, and many other tasks. [Go to the Amazon GameLift Servers console.](#)
- [Service API for Amazon GameLift Servers](#) – This API gives you programmatic access to all of your Amazon GameLift Servers resources. It is part of the AWS SDK, which you can download for use with most popular programming languages. [Get the AWS SDK.](#)
- [AWS command line interface \(CLI\)](#) – The AWS CLI lets you interact with AWS services using a command-line shell. The tools provides direct access to the public APIs for AWS services as well as customized commands that are available for a service. [Get the AWS CLI.](#)
- [AWS CloudFormation](#) for Amazon GameLift Servers – The CloudFormation service helps you model and set up AWS resources to streamline infrastructure deployment and management. Create an CloudFormation template to describe the Amazon GameLift Servers resources for your hosting solution, and then use the template to build additional resources or update configurations. View the [Amazon GameLift Servers resource type reference.](#)

For Amazon GameLift Servers Realtime

Configure and deploy Realtime servers to host your multiplayer games. To allow your game clients to connect to Realtime servers, use the Amazon GameLift Servers Realtime client SDK. To get started, [download the Realtime client SDK](#). For configuration information, see [Integrating a game client for Amazon GameLift Servers Realtime](#).

SDK support

The Realtime Client SDK contains source for the following languages:

- C# (.NET)

Development environments

Build the SDK from source as needed for the following supported development operating systems and game engines:

- **Operating systems** – Windows, Linux, Android, iOS
- **Game engines** – Unity, engines that support C# libraries
- **Node.js runtimes** – 10.x, 24.x

Note

Node.js 10 will reach end of support on September 30, 2026. See more details in the [Node.js 10 FAQs](#). For migration guidance, see [Migrating from Node.js 10 to 24](#).

Game server operating systems

You can deploy Realtime servers onto hosting resources that run on the following platforms:

- [Amazon Linux 2023](#)
- [Amazon Linux 2](#)

Tutorial: Quick onboarding with the Amazon GameLift Servers wrapper

Welcome to the onboarding tutorial for Amazon GameLift Servers. In this tutorial, you rapidly deploy your game server to be hosted on a fleet of cloud-based compute resources. Use this tutorial to skip the work of integrating the server SDK for Amazon GameLift Servers into your game code, and instead deploy your game with the minimal functionality to communicate with the Amazon GameLift Servers service and run game sessions. You will set up a basic hosting solution and use it to experience the full range of features such as automatic scaling and matchmaking. It's also a great way to host a prototype of your game as part of a live demo or for testing.

Key benefits of this onboarding method:

- Quickly deploy your game server for hosting fast.
- Zero game code changes and no modifications required.
- Use this method with any game executable, regardless of game engine.
- Explore all Amazon GameLift Servers management tools, including monitoring for game session activity and hosting health.

Note

The wrapper is intended for evaluation and basic production use. Advanced features such as detailed player session management require a full server SDK integration.

Prerequisites

Before you begin, ensure you have:

- An AWS account with appropriate permissions
- AWS CLI installed
- Go 1.18+
- A multiplayer game server executable
- Make (Linux/Mac)
- Git installed with an active account

Overview

In this tutorial you will:

1. Get and build the wrapper
2. Prepare the game build
3. Configure the wrapper
4. Upload the game server build
5. Create the managed EC2 fleet
6. Create and connect to a game session
7. Monitor and manage your game servers
8. Scale your game servers

Step 1: Get and build the game server wrapper

Use the following commands to get the game server wrapper source and build the wrapper. These commands use SSH, but you can also go to the GitHub repository directly.

Windows

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

Mac and Linux

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

On a successful build, an "out" directory is added to amazon-gamelift-servers-game-server-wrapper. In this directory are three folders, one for each supported hosting fleet option,

that contain a set of build artifacts. For this tutorial, you're deploying to managed EC2 fleets, so you'll use the folder `gamelift-servers-managed-ec2`.

Step 2: Prepare your game server build

In this step, you prepare your game server build files for uploading to the Amazon GameLift Servers.

Create the game directory

Now, prepare a game directory on your local machine. This directory must contain all the files you need to run your game server with Amazon GameLift Servers. This includes the game server wrapper, your game server build and the `config.yaml` that makes the wrapper work with your game server.

Use the following steps:

1. Managed EC2 fleet. In the game server wrapper output folder, find the build artifact for deploying to a Managed EC2 fleet. The build will be written to an out directory as indicated here: `out\linux\amd64\gamelift-servers-managed-ec2`.
2. Copy your game server executable and all associated files that it requires to run into the `gamelift-servers-managed-ec2` folder. You can have nested directories as needed.

An example directory structure will look like this:

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

Step 3: Configure the wrapper for your fleet

Amazon GameLift Servers manages the lifecycle of a fleet's compute instances, spinning up new instances with your server build installed and recycling instances as needed. The service manages

the game server process lifecycle that runs on each instance. A Managed EC2 fleet can have instances in multiple locations to support players wherever they are.

Edit the `config.yaml` file to configure the wrapper for logging, port setup, and server initialization.

1. **Configure the logging settings.** The game server wrapper generates log messages for each game server process. By default, the log level is set to debug for maximum verbosity. This is really useful during setup and troubleshooting and determines how detailed the log messages are - in this case, the most verbose. Options include debug, info, warn, and error (least verbose).
2. **Specify the path to the game server log directory.** The default path for game server logs is `./game-server-logs`. This directory contains all the logs generated by your game server and each instance has them. The logs are automatically uploaded to Amazon GameLift Servers where they are accessed from the **Events** tab. View the Troubleshooting section for more details.
3. **Define network port configuration.** Set the game port to whatever you prefer. For this tutorial, specify one port only, as you'll create a fleet that runs only one concurrent game server process per instance. If you decide to run multiple processes at a time, you'll need to configure enough ports for each concurrent process. The default value is 37016 as shown in the config file, but in general, for fleets using Linux builds use ports 22 and 1026-60000. For fleets using Windows builds, use ports 1026-60000.
4. **Set up the path to the game server executable.** For `./MyGame/my-server-executable.exe` customize the path to your game server executable with the actual name and location for it. This is the entry point for launching your game server.
5. **Configure the game server arguments.** At a minimum, specify a `-port` argument and use the same game port value you defined before. The "pos" value 0 indicates this is the first argument. Add other arguments as needed. These arguments are passed to your game server when it's launched, allowing you to configure its runtime behavior.
 - a. Argument: `--port`
 - b. Value: `"{{.port number here}}"`
 - c. Position: 0 (First argument in the list)

Example configuration:

```
log-config:
  wrapper-log-level: debug
game-server-logs-dir: ./game-server-logs
```

```
ports:
  gamePort: 37016
game-server-details:
  executable-file-path: ./MyGame/my-server-executable
game-server-args:
  - arg: "--port"
    val: "{{.gameport}}"
    pos: 0
```

Step 4: Upload the game server build

You've now completed all the required elements of your game server build (game server wrapper, config.yaml, and your game server files), and you're ready to upload your game build to Amazon GameLift Servers for hosting. The quickest way to upload your game build is by using the AWS CLI command `upload-build` as shown in the following example.

Uploading a game build with Windows:

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

Note

For Mac and Linux builds, use `--operating-system AMAZON_LINUX_2023`

When you create the build, record the build ID from the API response to use it for fleet creation.

Step 5: Create the managed EC2 fleet

The following steps describe a minimal fleet configuration, so you can be up and running as soon as possible.

To create your fleet:

1. Log into the AWS Management Console and navigate to Amazon GameLift Servers.
2. On the menu bar at the top of the console window, check to make sure what region your build is in. Make note of it because your fleet needs to be in the same region or you won't be able to find or choose your build.
3. In the navigation pane in the **Managed EC2** section, choose **Builds**.
4. Select the build you uploaded previously to display the **Build details** page.
5. In the Fleets section choose **Create Fleet** which displays the Define Managed EC2 fleet details page from which you are able to track status for your fleet as well as view fleet creation events on the **Events** tab.
6. Fill out the name and description and choose **Next**.
7. In the Define instance details page the build's region is shown by default. Choose any additional regions you want to add.
8. For **Fleet type** choose **On-Demand**.
9. Under **Instance types** choose **c5.large** and choose **Next**.
10. Under **Runtime configuration**, Because your uploaded game build uses the wrapper, you need to point to the wrapper executable instead. For Windows game servers this is `C:\game\amazon-gamelift-servers-game-server-wrapper.exe`. For Linux game servers, this is `/local/game/amazon-gamelift-servers-game-server-wrapper`.

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

Also, configure the game port values with a range that allows for the port that was set in the `config.yaml` and the runtime config launch parameters. The port in `config.yaml` does not need to match the one specified in the fleet's runtime configuration, but at runtime if the configuration specifies a different port, that value overrides what is in the `config.yaml`. Launch parameters entered in the runtime config also override what is in the `config.yaml`.

11. On the **Review and create** page, double-check all configurations, then choose **Submit** to create your fleet. The fleet status will change as it spins up capacity to host the game server, and very soon shows a status of **Active**. Once activation is complete and the fleet deployed, the service launches the wrapper which is ready to receive a game session request.

Step 6: Create and connect to a game session

When your fleet status shows **Active**, this means that game servers are ready and waiting to host a game session. To start a game session, make a game session request to the Amazon GameLift Servers service. Here, you will use the AWS CLI to make this request.

Note

Keep in mind that creating a game session using the AWS CLI is useful for testing and becoming familiar with the process. At some point, you'll add programmatic AWS SDK calls to your game backend service as part of your game's matchmaking or game session placement system.

Use the following to create a game session:

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

You can also pass customized game properties to your server executable. See the Game Server Arguments in the README for details. When it receives the create-game-session call, the Amazon GameLift Servers inform the wrapper to launch the game server executable and start a game session. What is in the `config.yaml` affects the game server's configuration, and the launch parameters that are set in the console determine the configuration of the game session itself.

Example format for adding game properties:

```
defaultArgs:  
  - arg: "--port"  
    val: "${.GamePort}"  
    pos: 0  
  - arg: "--ipAddress"  
    val: "${.IpAddress}"  
    pos: 1
```



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

Step 7: Manage and monitor your fleet

Now that your game server fleet is set up and a game session started, you can manage and monitor it from the Amazon GameLift Servers console. The best way to accomplish this is from the Fleet details page where you can edit fleet details or change fleet scaling and capacity on the **Scaling** tab. See the following section about scaling your game servers.

Choose the **Metrics** tab to view graphs depicting monitoring for Activity and Hardware metrics. For details about the metrics graphs, choose the **Info** link next to your fleet's ID. Additionally, while you can closely monitor your game servers from the **Metrics** tab, you can also add alarms to these metrics in the CloudWatch dashboard.

To navigate to the CloudWatch dashboard from the console:

1. Type "CloudWatch" in the search bar and choose it from the search results list to display the CloudWatch Overview.
2. Scroll down and choose **View GameLift dashboard** to see graphs based on key metrics for your fleets and game sessions.

Step 8: Scale your game servers

For the next step, you set up auto scaling. With auto scaling you dynamically scale your fleet capacity in response to game server activity. As players arrive and start game sessions, auto scaling adds more instances, and as player demand decreases, auto scaling releases unused instances. This is an effective way to minimize your hosting resources and costs while still providing a smooth, fast player experience.

As you prepare for game launch, you'll want to set up auto scaling for your fleets. Auto scaling is recommended as an effective way to minimize your hosting resources and costs while still providing a smooth, fast player experience.

To manually set up fleet capacity

To manually set up fleet capacity

1. Go to the **Scaling** tab in the details page for your fleet.
2. Select a location and choose **Edit**.
3. Change the **Desired instances** value and change the **Min** and **Max** size settings in order to scale beyond their current values, and choose **Confirm**.

 Note

Use the max instances setting as a stop gap to prevent excessive scaling and costs.

To use target-based auto scaling

To use target-based auto scaling

Target-based autoscaling (target tracking) links fleet scaling to the percentage of available game sessions. As players surge to play your game and available game sessions decrease, it responds by automatically adding more instances to the fleet.

1. Under **Target-based auto scaling policy**, choose **Add policy** and set the fleet's capacity to change automatically when it reaches the threshold of the percentage of available game sessions that you set. A larger buffer can better handle surges, getting new players get into games fast, but it may also lead to higher hosting costs.
2. Choose **Confirm** to accept the changes.

Rule-based auto scaling gives you more granular control, such as the ability to link scaling to other fleet metrics and set custom thresholds and scaling responses. It offers powerful options, but it also requires using the CLI and significant testing to understand how custom rules behave in action. This tutorial focuses on first setting up the target-based approach.

Troubleshoot common issues

The following are common issues you may encounter while getting your game servers and game sessions running. When either the server or the game session isn't running properly, the first step is to check the logs which may reveal one of the issues described below for new deployments or games in production.

The following is often revealed in the logs:

- Game server process can't launch. This might be an error in the wrapper config—verify the file has the right launch path and correct launch parameters and arguments.
- Game server build isn't runnable. This is likely an error in the game code.
- Players can't connect to game sessions. This is likely a port configuration error.
- Lag or slow connections. Review scaling policies and thresholds.
- No connection. Verify the port rules and configuration for your fleet.

To view the event logs for your Amazon GameLift Servers fleet

To view the event logs for your Amazon GameLift Servers fleet

1. Open the Amazon GameLift Servers console.
2. On the Fleet Details page choose the **Events** tab and download the log. You can also monitor activity and hardware metrics for game server health and game session activations from the **Metrics** tab.

To view game session logs

To view game session logs

1. From the console, open your fleet and open the **Game sessions** tab.
2. Choose a game session ID from the list to display the Overview page for it.
3. Choose **Download logs** to download a log file locally.

To view game session logs with the CLI use the `GetGameSessionLogURL` API. Amazon GameLift Servers automatically stores the logs for 14 days.

You can also configure Amazon CloudWatch logs for your fleet. This provides additional logging capabilities and integration with other AWS monitoring services.

For real-time log access or extended retention periods through CloudWatch:

1. At the top of your Amazon GameLift Servers console dashboard type "CloudWatch" in the search bar and choose it from the results drop-down.
2. Go to the CloudWatch Log Groups and search for specific sessions. The easiest method is to click **Search All** and filter using `gameSessionId` or `clientId`.

Next steps

- [Create a multi-location fleet to add hosting in additional locations](#)
- [Add a game session queue to make the best possible placements for game sessions across multiple locations](#)
- [Experiment with FlexMatch matchmaking by creating a matchmaker and rule set for your game](#)
- [Start working on functionality for your game client and backend service components, so players can make join requests and directly connect to game sessions](#)
- [When ready, move to a fully-integrated solution](#)

Amazon GameLift Servers examples

If you're considering using Amazon GameLift Servers to manage your custom game server, or you're interested in taking advantage of Amazon GameLift Servers Realtime, we recommend that you try the following examples before you use the service for your own game. The custom game server example gives you experience with game hosting in the Amazon GameLift Servers console. The Amazon GameLift Servers Realtime example shows you how to prepare a game for hosting using Realtime servers.

Custom game server example

This example demonstrates the process of deploying a sample game server to Amazon GameLift Servers managed EC2 fleet for hosting. Use the sample game client to connect to a live game session. You can experience how to use Amazon GameLift Servers .tools, including the console and the AWS CLI, to monitor the fleet's hosting performance and usage.

The example walks you through the following steps:

- Upload the sample game server build.
- Create a fleet to run the game server build.
- Get the sample game client and use it to connect to a game server and join a game session.
- Review fleet and game session metrics.

Start up multiple game clients and play the game to generate hosting data. Use the Amazon GameLift Servers console to view hosting resources, track metrics, and explore options for scaling the fleet's hosting capacity.

To get started, sign in to the [Amazon GameLift Servers console](#). In the left-side navigation, go to **Resources, Try a sample game**.

Building a game hosting solution with Amazon GameLift Servers

This section includes guidance on how to build all the components of an Amazon GameLift Servers game hosting solution, including design best practices. Whether you're starting from scratch or migrating an existing game, you'll find comprehensive information to help you architect, implement, and optimize your game hosting infrastructure using Amazon GameLift Servers.

Hosting solution components

Your hosting solution must have some version of the following components. For more detail on these components and how they work together, see [How hosting with Amazon GameLift Servers works](#).

- **Game client** runs on player devices.
- **Backend service** coordinates communication between game clients and Amazon GameLift Servers.
- **Game server** software manages multiplayer gameplay.
- **Placement system** matches players to game servers.
- **Game hosting fleets** provide computing resources in geographic locations.
- **Game hosting management system** monitors game hosting status and manages capacity.

Topics

- [Development roadmaps](#)
- [Prepare a game for hosting with Amazon GameLift Servers](#)
- [Build a backend service for Amazon GameLift Servers](#)
- [Deploy hosting fleets for Amazon GameLift Servers](#)
- [Configure game session placement](#)
- [Customize to your game hosting solution](#)

Development roadmaps

Development roadmaps provide structured, step-by-step guidance for building different types of game hosting solutions with Amazon GameLift Servers. Each roadmap is tailored to a specific

hosting approach and walks you through the essential tasks, from initial setup to deployment and testing.

Choose the roadmap that best matches your hosting requirements and technical constraints. You can also combine approaches by following multiple roadmaps to create a hybrid solution that meets your specific needs.

Topics

- [Development roadmap for hosting with Amazon GameLift Servers managed EC2](#)
- [Development roadmap for hosting with Amazon GameLift Servers managed containers](#)
- [Development roadmap for hosting with Amazon GameLift Servers Anywhere](#)
- [Development roadmap for hybrid hosting with Amazon GameLift Servers](#)

Development roadmap for hosting with Amazon GameLift Servers managed EC2

This roadmap guides you through how to develop an Amazon GameLift Servers managed EC2 hosting solution for your multiplayer game. Amazon GameLift Servers offers several game hosting options; for more information on these options, see [Amazon GameLift Servers game hosting options](#).

With Amazon GameLift Servers managed hosting, your game server is hosted on AWS Cloud-based virtual computing resources that Amazon GameLift Servers owns and operates based on your configuration. You get the security, reliability, and global availability of Amazon Elastic Compute Cloud (Amazon EC2) instances that are further optimized for use with multiplayer game hosting. Amazon GameLift Servers streamlines hosting management with tools such as automatic server deployments, life cycle handling, and capacity auto scaling.

An Amazon GameLift Servers managed solution is composed of the following components:

- A game server build, integrated with the server SDK for Amazon GameLift Servers, to deploy across all fleets.
- A game client and backend service, integrated with the AWS SDK, to interact with the Amazon GameLift Servers service and request game sessions.
- An Amazon GameLift Servers queue or other placement mechanism to place new game sessions with available game servers across all fleets.

- (Optional) A FlexMatch matchmaker to create multi-player matches and set up game sessions for them.
- One or more Amazon GameLift Servers managed fleets, which use Amazon Elastic Compute Cloud (Amazon EC2) instances optimized for multiplayer game hosting.

This roadmap presents a streamlined path to getting your multiplayer game up and running successfully with Amazon GameLift Servers managed EC2 hosting. After you have the necessary components in place, you can continue to iterate on game development and customize your hosting solution. As you get closer to launch, see these [Prepare for launch with Amazon GameLift Servers hosting](#) for help with preparing your hosting solution for production-level usage.

** Get a jump start with the Amazon GameLift Servers plugin for Unreal Engine and Unity**

For faster deployment, try the [Amazon GameLift Servers plugin](#) for Unreal Engine and Unity. It provides guided UI workflows to quickly deploy your game server with minimal setup, so you can try out your game components in action. Then you can build on this foundation to create a custom hosting solution for your game. For more details, see [Prepare your Unreal or Unity game with the Amazon GameLift Servers plugin](#).

Step 1: Prepare your game server to work with Amazon GameLift Servers

Add functionality to your game server so that it can communicate with the Amazon GameLift Servers service when it's deployed for hosting.

- **Get the server SDK for Amazon GameLift Servers (version 5.x) for your game project.** The server SDK is available in C++, C#, and Go. [Download an Amazon GameLift Servers server SDK](#).
- **Modify your game server code to add server SDK functionality.** For guidance, see [Prepare a game for hosting with Amazon GameLift Servers](#). At a minimum, do the following:
 - Add code to initialize the Amazon GameLift Servers SDK and establish a WebSocket connection with the Amazon GameLift Servers service. Use the server SDK action `InitSdk()`.
 - Add code to report to the Amazon GameLift Servers service when the server process is ready to host game sessions. Use the server SDK action `ProcessReady()`.
 - Implement the required callback functions `OnProcessTerminate()`, and `OnStartGameSession()`. With these functions, game server processes can maintain a

connection with the Amazon GameLift Servers service, initiate a game session when prompted by Amazon GameLift Servers, and respond to a prompt to end the game server process.

- Add code to report to the Amazon GameLift Servers service when the server process is ending a game session. Use the server SDK action `ProcessEnding()`.
- **Package your game server build.** Create an install script with your build files, dependencies and supporting software. See [Package your game build files](#). We recommend using an Amazon Simple Storage Service (Amazon S3) bucket to store versions of your game build.
- **Test your game server integration.** For this task, we recommend setting up an Amazon GameLift Servers Anywhere fleet for a local workstation, as described in [Set up local testing with Amazon GameLift Servers Anywhere](#). For this step, manually install your game server build onto the test device and start a server process. Use the AWS CLI to request a new game session, and verify that the Amazon GameLift Servers service successfully prompts your server process to start a game session.

Step 2: Prepare your game client to join hosted game sessions

Create a way for your game client to request to join a game session, get connection info, and then connect directly to a hosted game session. The most common approach is to set up backend service functionality that serves as a middleman between your game client and the Amazon GameLift Servers service. This approach protects your hosting resources and gives you greater control over how players are placed into game sessions.

- **Build backend service functionality for hosting.** The backend service communicates with the Amazon GameLift Servers service and delivers connection information to a game client. This functionality includes starting game sessions, placing players into games, and retrieving game session information. For guidance, see [Prepare a game for hosting with Amazon GameLift Servers](#). At a minimum, do the following:
 - Get the AWS SDK for Amazon GameLift Servers and add it to your backend service project. See [Amazon GameLift Servers SDK resources for client services](#).
 - Add code to initialize an Amazon GameLift Servers client and store key settings. See [Set up the Amazon GameLift Servers API](#).
 - Add functionality to call the AWS SDK action `CreateGameSession()` and provide game session connection information to a game client. See [Create a game session on a specific fleet](#).

Calling `CreateGameSession()` is a convenient starting point for requesting new game sessions. After you have a game session placement system in place (see Step 3), you'll replace

this code with a call to `StartGameSessionPlacement()` (or `StartMatchmaking()` if you're using FlexMatch).

For guidance on designing your backend service, see [Build a backend service for Amazon GameLift Servers](#).

- **Add functionality to your game client that lets players join a hosted game session.** The game client makes requests to your backend service, not directly to Amazon GameLift Servers. After the backend service provides game session connection information, the game client connects directly with the game session to play the game.
- **Test your game client integration.** You can use the same Amazon GameLift Servers Anywhere fleet with a local workstation for testing.

For fast iterative development or when working with multi-person teams, we recommend that you set up a [cloud-based test environment](#). This Amazon GameLift Servers Toolkit solution mimics the behavior of an Amazon GameLift Servers managed fleet but lets you update game server builds with minimal turnaround time.

Step 3: Set up game session placement

Customize how you want Amazon GameLift Servers to process requests for new game session and locate available game servers to host them. Amazon GameLift Servers automatically tracks the availability of all game servers on all fleets. When a game client sends a request to join a game session, Amazon GameLift Servers looks for the "best possible" placement based on a set of defined priorities such as minimum latency, cost, and availability.

- **Create a game session queue for placing new game session with available game servers.** Queues are the primary mechanism for game session placement. For guidance, see [Create a game session queue](#).
 - At minimum, add your Anywhere fleets as destinations in your queue. All other settings are optional customizations.
- **In your backend service code, convert the `CreateGameSession()` call to `StartGameSessionPlacement()`.** See [Create a game session in a multi-location queue](#).
- **Create a mechanism to notify a game client when a game session is ready to join.** While in development, you can poll for game session status using a call to `DescribeGameSessionPlacement`. Before using a queue to process high volumes, however, you'll need to enable event notifications. See [Set up event notification for game session placement](#).

- (Optional) **Add FlexMatch matchmaking components.** For guidance, see the [Amazon GameLift Servers FlexMatch developer guide](#).

Step 4: Create managed cloud-based fleets

Up to this point you've worked with self-managed Anywhere fleets to test and iterate on your game components and you've fine-tuned your game session placement. The final piece of your solution is to set up the type of hosting resources you'll need for a production system. To start planning and configuring for production, you want to transition to working with an Amazon GameLift Servers managed fleet.

- **Package your game server build and upload to Amazon GameLift Servers.** Create an install script with your build files, dependencies and supporting software. See [Create a game server build for Amazon GameLift Servers](#). You can upload your build to Amazon GameLift Servers using either the console or the AWS CLI.

Before uploading your build, decide in what AWS Region you want to create a fleet. You must upload the build to the same Region. For more on choosing a fleet location, see [Geographic locations](#).

- **Create a managed EC2 fleet.** When you create a fleet, Amazon GameLift Servers immediately begins deploying your game server build for hosting. You can configure many aspects of a managed fleet. For guidance, see [Create an Amazon GameLift Servers managed EC2 fleet](#). At minimum, do the following:
 - Give the fleet a name and specify which uploaded game build to deploy.
 - Choose On-Demand Instances for your fleet and select an instance type that's available in your fleet location. Spot fleets are a valuable option but require additional design and configuration.
 - Create a runtime configuration for the fleet. At minimum, specify the launch path for your game server executable.
 - Specify port settings to allow inbound traffic to access your game servers.
- **Add the managed fleets to your queue.** In your game session queue, replace the Anywhere fleets with your managed fleets.
- **Test game hosting with your managed fleets.** At this point you should be able to test the entire hosting cycle, with a game client requesting a game session, getting connection info, and successfully connecting to a game session.

Step 5: Customize your managed fleets

As you prepare for game launch, you'll need to fine-tune your managed hosting resources. Some of the decisions to consider include:

- Consider adding Spot fleets for cost savings. See [Reduce game hosting costs with Spot fleets](#).
- If your game server needs to communicate other AWS resources, set up IAM roles to manage access. See [Connect your Amazon GameLift Servers hosted game server to other AWS resources](#).
- Determine where geographically you want to position game servers. Add remote locations to your managed fleets. See [Hosting resource customizations](#).
- Optimize fleet performance by selecting an instance type and size and configuring the runtime to run multiple server processes. See [Optimize game server runtime configuration on managed Amazon GameLift Servers](#).
- Experiment with game session placement options for managed fleets, including customizing prioritization settings. See [Customize a game session queue](#).
- Set up automatic capacity scaling to meet expected player demand. See [Scaling game hosting capacity with Amazon GameLift Servers](#).
- Set up standby fleets in other AWS Regions and modify queues and auto scaling to handle failovers if needed.
- Set up hosting observability tools, including analytics and logging. See [Monitoring Amazon GameLift Servers](#).
- Automate your deployment using [infrastructure as code \(IaC\)](#). See [Manage Amazon GameLift Servers hosting resources using CloudFormation](#).

Amazon GameLift Servers supports the use of AWS CloudFormation templates for any deployment-specific configurations. You can also use the AWS Cloud Development Kit (AWS CDK) to define your Amazon GameLift Servers resources. For more information about the AWS CDK, see the [AWS Cloud Development Kit \(AWS CDK\) Developer Guide](#).

To manage the deployment of your CloudFormation stacks, we recommend using continuous integration and continuous delivery (CI/CD) tools and services such as AWS CodePipeline. These tools help you deploy automatically or with approval whenever you build game server binary. With a CI/CD tool or service, resources deployment for a new game server version can look like this:

- Build and test your game server binary.
- Upload the binary to Amazon GameLift Servers.

- Deploy new fleets with the new build.
- Add the new fleets to your game session queue and remove the fleets with the previous build version.
- When the fleets with the previous build are no longer hosting active game sessions, delete the CloudFormation stacks of those fleets.

Development roadmap for hosting with Amazon GameLift Servers managed containers

This roadmap guides you through how to develop an Amazon GameLift Servers managed hosting solution for your containerized game servers. Managed containers is just one hosting solution that is offered by Amazon GameLift Servers. For more information on hosting options, see [Amazon GameLift Servers game hosting options](#).

A managed container solution with Amazon GameLift Servers has the following components:

- A container image with your game server build, uploaded to Amazon Elastic Container Registry (Amazon ECR) private repository. The game server build is integrated with the server SDK for Amazon GameLift Servers and built to run on Linux.
- A backend service that interacts with the Amazon GameLift Servers service on behalf of your game clients. The backend service uses functionality in the service API for Amazon GameLift Servers, which is part of the AWS SDK.
- An Amazon GameLift Servers game session queue or other placement mechanism that processes requests for new game sessions, searches for available game servers across all fleets, and prompts a game server to start a game session.
- (Optional) A FlexMatch matchmaker to create multi-player matches and set up game sessions for them.
- One or more container fleets, which use Amazon Elastic Compute Cloud (Amazon EC2) instances and are optimized for multiplayer game hosting.

This roadmap presents a streamlined path to getting your containerized game servers up and running successfully with Amazon GameLift Servers managed containers. After you have the necessary components in place, you can continue to iterate on game development and customize your hosting solution. As you get closer to launch, see these [Prepare for launch with Amazon GameLift Servers hosting](#) for help with preparing your hosting solution for production-level usage.

Speed up onboarding with these tools for managed containers:

- The [containers starter kit](#) streamlines integration and fleet setup. It adds essential game session management features to your game server, and uses pre-configured templates to build a container fleet and an automated deployment pipeline for your game server. After deployment, use the Amazon GameLift Servers console and API tools to monitor fleet performance, manage game sessions, and analyze metrics.
- For Unreal Engine and Unity developers, use the [Amazon GameLift Servers plugins](#) to integrate your game server and build a container fleet from inside your game engine's development environment. The plugin's guided workflows help you create a fast, simple solution with cloud-based hosting using managed containers. You can build on this foundation to create a custom hosting solution for your game.

Step 1: Prepare your game server to work with Amazon GameLift Servers

Add functionality to your game server so that it can communicate with the Amazon GameLift Servers service when it's deployed for hosting.

- **Get the server SDK for Amazon GameLift Servers (version 5.2 or greater) for your game project.** The server SDK is available in C++, C#, and Go. [Download the server SDK for Amazon GameLift Servers](#). The server SDK is available in C++, C#, and Go.
- **Modify your game server code to add server SDK functionality.** For guidance, see [Prepare a game for hosting with Amazon GameLift Servers](#). At a minimum, do the following:
 - Add code to initialize the Amazon GameLift Servers SDK and establish a WebSocket connection with the Amazon GameLift Servers service. Use the server SDK action `InitSdk()`.
 - Add code to report to the Amazon GameLift Servers service when the server process is ready to host game sessions. Use the server SDK action `ProcessReady()`.
 - Implement the required callback functions `OnStartGameSession()`, and `OnProcessTerminate()`. With these functions, game server processes can maintain a connection with the Amazon GameLift Servers service, initiate a game session when prompted by Amazon GameLift Servers, and respond to a request to end the game server process.
 - Add code to report to the Amazon GameLift Servers service when the server process is ending a game session. Use the server SDK action `ProcessEnding()`.

- **Package your game server build.** Build your game server to run on Linux. Prepare the build and other files that are needed to run the game server. If you're developing on Windows, this step might involve setting up a separate Linux workspace or using a tool such as Windows subsystem for Linux (WSL). You'll need a Linux environment to test your game server build, and also to build and test your container images.
- **Test your game server integration.** Verify that your integrated game server can connect to the Amazon GameLift Servers service and respond to prompts. We recommend setting up a simple Amazon GameLift Servers Anywhere fleet with a local workstation as a test host, as described in [Set up local testing with Amazon GameLift Servers Anywhere](#). Install your game server build onto the test host and start a server process. Use the AWS CLI to request a new game session, and verify that the Amazon GameLift Servers service successfully prompts your server process to start a game session.

Step 2: Create a game server container image

After you've successfully integrated your game server, create a container image with your game server executable. Store it in an Amazon Elastic Container Registry (Amazon ECR) repository for use with Amazon GameLift Servers. For detailed instructions, see [Build a container image for Amazon GameLift Servers](#).

- **Get the Dockerfile template for a game server container (provided by Amazon GameLift Servers).** Modify the file for your game server build files.
- **Build a game server container image.** Working in a Linux environment, use the Docker tool to create your image.
- **Push your container image to Amazon ECR.** Create a public or private repository in Amazon ECR, using the same AWS account and AWS Region where you plan to deploy your container fleet. Push your container image to it.
- **Test your container images using your Anywhere fleet (optional).** You might want to test your container images locally before deploying them to a cloud-hosted container fleet. You can use your existing Amazon GameLift Servers Anywhere fleet with a local workstation for testing. Install and run the game server container and verify that: (1) the Amazon GameLift Servers service successfully prompts your server process to start a game session, and (2) a game client can connect to the game session.

Step 3: Create a cloud-based container fleet

Up to this point you've worked with a self-managed Anywhere fleet to test and iterate on your game components. When you have a working game server build that's integrated for Amazon GameLift Servers, you can start setting up the cloud-based Amazon GameLift Servers managed container fleet hosting resources that you'll need for a production environment.

- **Create container group definitions.** Container group definitions describe the container architecture for a fleet, and identify which container images to deploy. See [Create a container group definition for an Amazon GameLift Servers container fleet](#). Create your container group definition in the same AWS Region where the container images are stored. For more on choosing a fleet location, see [Geographic locations](#). At a minimum, do the following:
 - Create a game server container group definition.
 - Add a container definition with a container image with your game server build.
 - Configure a port range for the container's game server processes.
- **Create a managed container fleet.** When you create a fleet, Amazon GameLift Servers immediately begins deploying your game server build for hosting. You can configure many aspects of a managed fleet. For guidance, see [Create an Amazon GameLift Servers managed container fleet](#). At minimum, do the following:
 - Set up an AWS Identity and Access Management (IAM) service role for the container fleet. See [Set up an IAM service role for Amazon GameLift Servers](#).
 - Specify the game server container group definition to deploy to fleet instances.
 - Use default values where available for all other parameters. Amazon GameLift Servers calculates some parameters for optimal configuration.
- **Add the container fleets to your queue.** In your game session queue, replace the Anywhere test fleet with your managed container fleet.
- **Test game hosting with your container fleets.** At this point you should be able to test the entire solution. Start a game client and request a game session through the backend service. Get connection info and connect to a game session on the container fleet.
- **Iterate on your fleet deployments.** You can update container group definitions and fleet configurations, and then deploy updates to existing fleets.

Step 4: Prepare your game client to join hosted game sessions

Create a way for your game client to request to join a game session, get connection info, and then connect directly to a hosted game session. The most common approach is to set up backend service functionality that serves as a middleman between your game client and the Amazon GameLift Servers service. This approach protects your hosting resources and gives you greater control over how players are placed into game sessions.

- **Build backend service functionality for hosting.** The backend service communicates with the Amazon GameLift Servers service and delivers connection information to a game client. This functionality includes starting game sessions, placing players into games, and retrieving game session information. For guidance, see [Prepare a game for hosting with Amazon GameLift Servers](#). At a minimum, do the following:
 - Get the AWS SDK for Amazon GameLift Servers and add it to your backend service project. See [Amazon GameLift Servers SDK resources for client services](#).
 - Add code to initialize an Amazon GameLift Servers client and store key settings. See [Set up the Amazon GameLift Servers API](#).
 - Add functionality to call the AWS SDK action `CreateGameSession()` and provide game session connection information to a game client. See [Create a game session on a specific fleet](#).

Calling `CreateGameSession()` is a convenient starting point for requesting new game sessions. After you have a game session placement system in place (see Step 3), you'll replace this code with a call to `StartGameSessionPlacement()` (or `StartMatchmaking()` if you're using FlexMatch).

For guidance on designing your backend service, see [Build a backend service for Amazon GameLift Servers](#).

- **Add functionality to your game client that lets players join a hosted game session.** The game client makes requests to your backend service, not directly to Amazon GameLift Servers. After the backend service provides game session connection information, the game client connects directly with the game session to play the game.
- **Test your game client integration.** You can use your existing Amazon GameLift Servers Anywhere fleet with a local workstation for testing. Use the new backend service to request a new game session, and verify that: (1) the Amazon GameLift Servers service successfully prompts your server process to start a game session, and (2) a game client can connect to the game session.

Step 5: Set up game session placement

Customize how you want Amazon GameLift Servers to process requests for new game session and locate available game servers to host them. Amazon GameLift Servers automatically tracks the availability of all game servers on all fleets. When a game client sends a request to join a game session, Amazon GameLift Servers looks for the "best possible" placement based on a set of defined priorities such as minimum latency, cost, and availability.

- **Create a game session queue for placing new game session with available game servers.** Queues are the primary mechanism for game session placement. For guidance, see [Create a game session queue](#).
 - At minimum, add your Anywhere fleets as destinations in your queue. All other settings are optional customizations.
- **In your backend service code, convert the `CreateGameSession()` call to `StartGameSessionPlacement()`.** See [Create a game session in a multi-location queue](#).
- **Create a mechanism to notify a game client when a game session is ready to join.** While in development, you can poll for game session status using a call to `DescribeGameSessionPlacement`. Before using a queue to process high volumes, however, you'll need to enable event notifications. See [Set up event notification for game session placement](#).
- **Add FlexMatch matchmaking (optional).** Build a matchmaking rule set and create a matchmaking configuration to work with your game session queue. For guidance on setting up a matchmaking system, see the [Amazon GameLift Servers FlexMatch developer guide](#).
- **Test the placement system.** You can use your existing Amazon GameLift Servers Anywhere fleet with a local workstation for testing. Use the backend service to request a new game session, and verify that the Amazon GameLift Servers service successfully prompts your server process to start a game session.

Step 6: Customize your managed container solution

As you prepare for game launch, you'll need to fine-tune your managed hosting resources. Some of the decisions to consider include:

- Optimize your container fleet configuration. See [Customize an Amazon GameLift Servers container fleet](#).
- Consider adding Spot fleets for cost savings. See [Reduce game hosting costs with Spot fleets](#).

- If your game server needs to communicate other AWS resources, set up IAM roles to manage access. See [Connect your Amazon GameLift Servers hosted game server to other AWS resources](#).
- Determine where geographically you want to position game servers. Add remote locations to your managed fleets. See [Hosting resource customizations](#).
- Experiment with game session placement options for managed fleets, including customizing prioritization settings. See [Customize a game session queue](#).
- Set up automatic capacity scaling to meet expected player demand. See [Scaling game hosting capacity with Amazon GameLift Servers](#).
- Create fleets in other AWS Regions and modify queues and auto scaling to handle failovers as needed.
- Set up hosting observability tools, including analytics and logging. See [Monitoring Amazon GameLift Servers](#).
- Automate your fleet deployments using [infrastructure as code \(IaC\)](#). See [Manage Amazon GameLift Servers hosting resources using CloudFormation](#).

Amazon GameLift Servers supports the use of AWS CloudFormation templates for any deployment-specific configurations. You can also use the AWS Cloud Development Kit (AWS CDK) to define your Amazon GameLift Servers resources. For more information about the AWS CDK, see the [AWS Cloud Development Kit \(AWS CDK\) Developer Guide](#).

To manage the deployment of your CloudFormation stacks, we recommend using continuous integration and continuous delivery (CI/CD) tools and services such as AWS CodePipeline. These tools help you deploy automatically or with approval whenever you build game server binary. With a CI/CD tool or service, resources deployment for a new game server version can look like this:

- Build and test your game server binary.
- Upload the binary to Amazon GameLift Servers.
- Deploy new fleets with the new build.
- Add the new fleets to your game session queue and remove the fleets with the previous build version.
- When the fleets with the previous build are no longer hosting active game sessions, delete the CloudFormation stacks of those fleets.

Development roadmap for hosting with Amazon GameLift Servers Anywhere

This roadmap guides you through how to develop a hosting solution for your multiplayer game to use with your own resources (on-premises hardware or virtual machines). Amazon GameLift Servers offers several game hosting options; for more information on these options, see [Amazon GameLift Servers game hosting options](#).

With Amazon GameLift Servers Anywhere hosting, your game server is hosted on computing resources that you supply and manage. You can create an Anywhere fleet with the configurations that you need and geographically located wherever your players are. Amazon GameLift Servers delivers the following features for an Anywhere fleet:

- Handles the game session placement process for you based on your configuration, including:
 - Tracking game server availability across your Anywhere fleets.
 - Processing game requests from your game client service and matching game requests with available servers.
 - Prompting game servers on Anywhere fleets to start game sessions.
 - Communicating connection details back to game clients.
- Collects performance metrics for the session placement process and usage metrics for game sessions and players.
- Supports the full FlexMatch matchmaking feature set, so that you can build a matchmaker and integrate it with the game session placement system.
- Offers the Amazon GameLift Servers Agent to handle key host management tasks on an Anywhere fleet.
- Supports combining with Amazon GameLift Servers managed fleets for a flexible hybrid solution.

An Amazon GameLift Servers Anywhere solution is composed of the following components:

- A game server build, integrated with the server SDK for Amazon GameLift Servers, to deploy across all fleets.
- A game client and backend service, integrated with the AWS SDK, to interact with the Amazon GameLift Servers service and request game sessions.

- An Amazon GameLift Servers queue or other placement mechanism to place new game sessions with available game servers across all fleets.
- (Optional) A FlexMatch matchmaker to create multi-player matches and set up game sessions for them.
- One or more Amazon GameLift Servers Anywhere fleets with your on-premises or other hosting resources, managed with your existing configuration management and deployment tooling. (You can optionally use the AWS Systems Manager.)

This roadmap presents a streamlined path to getting your multiplayer game up and running successfully with Amazon GameLift Servers Anywhere hosting. After you have the necessary components in place, you can continue to iterate on game development and customize your hosting solution. As you get closer to launch, see these [Prepare for launch with Amazon GameLift Servers hosting](#) for help with preparing your hosting solution for production-level usage.

Get a jump start with the Amazon GameLift Servers plugin for Unreal Engine and Unity

For faster deployment, try the [Amazon GameLift Servers plugin](#) for Unreal Engine and Unity. It provides guided UI workflows to quickly deploy your game server with minimal setup, so you can try out your game components in action. Then you can build on this foundation to create a custom hosting solution for your game. For more details, see [Prepare your Unreal or Unity game with the Amazon GameLift Servers plugin](#).

Step 1: Prepare your game server to work with Amazon GameLift Servers

Add functionality to your game server so that it can communicate with the Amazon GameLift Servers service when it's deployed for hosting.

- **Get the server SDK for Amazon GameLift Servers (version 5.x) for your game project.** The server SDK is available in C++, C#, and Go. [Download an Amazon GameLift Servers server SDK](#).
- **Modify your game server code to add server SDK functionality.** For guidance, see [Prepare a game for hosting with Amazon GameLift Servers](#). At a minimum, do the following:
 - Add code to initialize the Amazon GameLift Servers SDK and establish a WebSocket connection with the Amazon GameLift Servers service. Use the server SDK action `InitSdk()` and include server parameters, which are required for an Anywhere fleet.

- Add code to report to the Amazon GameLift Servers service when the server process is ready to host game sessions. Use the server SDK action `ProcessReady()`.
- Implement the required callback functions `OnProcessTerminate()`, and `OnStartGameSession()`. With these functions, game server processes can maintain a connection with the Amazon GameLift Servers service, initiate a game session when prompted by Amazon GameLift Servers, and respond to a prompt to end the game server process.
- Add code to report to the Amazon GameLift Servers service when the server process is ending a game session. Use the server SDK action `ProcessEnding()`.
- **Package your game server build.** Create an install script with your build files, dependencies and supporting software. See [Package your game build files](#). We recommend using an Amazon Simple Storage Service (Amazon S3) bucket to store versions of your game build.
- **Test your game server integration.** For this task, we recommend setting up an Amazon GameLift Servers Anywhere fleet for a local workstation, as described in [Set up local testing with Amazon GameLift Servers Anywhere](#). For this step, manually install your game server build onto the test device and start a server process. Use the AWS CLI to request a new game session, and verify that the Amazon GameLift Servers service successfully prompts your server process to start a game session.

Step 2: Prepare your game client to join hosted game sessions

Create a way for your game client to request to join a game session, get connection info, and then connect directly to a hosted game session. The most common approach is to set up backend service functionality that serves as a middleman between your game client and the Amazon GameLift Servers service. This approach protects your hosting resources and gives you greater control over how players are placed into game sessions.

- **Build backend service functionality for hosting.** The backend service communicates with the Amazon GameLift Servers service and delivers connection information to a game client. This functionality includes starting game sessions, placing players into games, and retrieving game session information. For guidance, see [Prepare a game for hosting with Amazon GameLift Servers](#). At a minimum, do the following:
 - Get the AWS SDK for Amazon GameLift Servers and add it to your backend service project. See [Amazon GameLift Servers SDK resources for client services](#).
 - Add code to initialize an Amazon GameLift Servers client and store key settings. See [Set up the Amazon GameLift Servers API](#).

- Add functionality to call the AWS SDK action `CreateGameSession()` and provide game session connection information to a game client. See [Create a game session on a specific fleet](#).

Calling `CreateGameSession()` is a convenient starting point for requesting new game sessions. After you have a game session placement system in place (see Step 3), you'll replace this code with a call to `StartGameSessionPlacement()` (or `StartMatchmaking()` if you're using FlexMatch).

For guidance on designing your backend service, see [Build a backend service for Amazon GameLift Servers](#).

- **Add functionality to your game client that lets players join a hosted game session.** The game client makes requests to your backend service, not directly to Amazon GameLift Servers. After the backend service provides game session connection information, the game client connects directly with the game session to play the game.
- **Test your game client integration.** You can use the same Amazon GameLift Servers Anywhere fleet with local workstations for testing.

Step 3: Set up game session placement

Customize how you want Amazon GameLift Servers to process requests for new game session and locate available game servers to host them. Amazon GameLift Servers automatically tracks the availability of all game servers on all fleets. When a game client sends a request to join a game session, Amazon GameLift Servers looks for the "best possible" placement based on a set of defined priorities such as minimum latency, cost, and availability.

- **Create a game session queue for placing new game session with available game servers.** Queues are the primary mechanism for game session placement. For guidance, see [Create a game session queue](#).
 - At minimum, add your Anywhere fleets as destinations in your queue. All other settings are optional.
- **In your backend service code, convert the `CreateGameSession()` call to `StartGameSessionPlacement()`.** See [Create a game session in a multi-location queue](#).
- **Create a mechanism to notify a game client when a game session is ready to join.** While in development, you can poll for game session status using a call to `DescribeGameSessionPlacement`. Before using a queue to process high volumes, however, you'll need to enable event notifications. See [Set up event notification for game session placement](#).

- (Optional) **Add FlexMatch matchmaking components.** For guidance, see the [Amazon GameLift Servers FlexMatch developer guide](#).

Step 4: Set up an Anywhere fleet with the Amazon GameLift Servers Agent

Up to this point you've been working with local devices (registered as Anywhere fleet computes) to test and iterate on your game components. The next step is to set up the type of fleet you'll need for a production system. For these resources, use the Amazon GameLift Servers Agent to manage some key on-compute host management tasks. For more details, see [Work with the Amazon GameLift Servers Agent](#).

- **Get the Amazon GameLift Servers Agent and add it to your game server install package.** Get and build the Agent source code, available in the [Amazon GameLift Servers Agent GitHub repository](#). Place the resulting JAR file executable into the same directory as your game build executable.
- **Modify your startup script for the Agent as needed.** Ensure that Agent executable launches as soon as a compute starts running. See the readme file in the Agent repo for help with installing and running the Agent on your hosting computes. Your launch command should include options to specify, at minimum, the Anywhere fleet ID and AWS Region, a custom location, and a compute name.

The Agent automatically handles the following tasks for you, so if you've been handling these tasks with scripts, you can remove them:

- Calls `RegisterCompute()` to add the compute to an Anywhere fleet.
- Calls `GetComputeAuthToken()` to authenticate game servers when they connect to the Amazon GameLift Servers service. The Agent manages getting and refreshing the auth token, which can be used by all game server processes that are running on the compute.
- Starts new server processes on the compute based on a set of runtime instructions.
- **Create a runtime configuration for computes in your Anywhere fleet.** You can use the Amazon GameLift Servers console or the AWS CLI to create or modify runtime instructions for the fleet. The Agent carries out these instructions and periodically requests updates from the Amazon GameLift Servers service.
- **Set up or modify your game session queue as needed.** Create a new queue (or update an existing one) to use the Anywhere fleets that are deployed with the Amazon GameLift Servers Agent.

- **Test the Agent integration with your Anywhere fleets.** Check that the Agent is properly starting server processes based on the runtime configuration.

Step 5: Customize your Anywhere fleets

As you prepare for game launch, you'll need to fine-tune your managed hosting resources. Some of the decisions to consider include:

- Automate the process of starting and shutting down computes as needed, including installing and running game server software. Recycling computes is useful to ensure that they are updated regularly, and shutting down computes can save costs when they're not needed.
- If your game server needs to communicate other AWS resources, set up IAM roles to manage access. See [Connect your Amazon GameLift Servers hosted game server to other AWS resources](#).
- Determine where geographically you want to position game servers. Add remote locations to your managed fleets. See [Hosting resource customizations](#).
- Optimize fleet performance by selecting compute resource configurations, then configure the runtime instructions to run an optimal number of server processes per compute.
- Experiment with game session placement options for managed fleets, including customizing prioritization settings. See [Customize a game session queue](#).
- Create mechanisms to handle manual or automated capacity scaling to meet expected player demand. Consider what factors should prompt the system to increase or decrease the number of computes that are available to host game sessions.
- Design and implement failover to other resources if needed.
- Set up hosting observability tools, including analytics and logging. See [Monitoring Amazon GameLift Servers](#).

Development roadmap for hybrid hosting with Amazon GameLift Servers

This roadmap guides you through how to develop a hosting solution for your multiplayer game. Amazon GameLift Servers offers several game hosting options; for more information on these options, see [Amazon GameLift Servers game hosting options](#).

A hybrid solution uses a combination of hosting resources, including cloud-based resources managed by Amazon GameLift Servers and your own self-managed hosting resources. For a more

detailed discussion of hybrid hosting, see this article: [Hybrid game server hosting with Amazon GameLift Servers Anywhere](#). With Amazon GameLift Servers, you can set up a hybrid solution that uses common components and processes, so you can centrally manage a global fleet and easily move loads between all types of resources.

A hybrid architecture consists of the following components:

- A single game server build, integrated with the server SDK for Amazon GameLift Servers, to deploy across all fleets.
- A single game client and backend service, integrated with the AWS SDK, to interact with the Amazon GameLift Servers service and request game sessions.
- A shared Amazon GameLift Servers queue to place new game sessions with available game servers and balance load across all fleets.
- (Optional) A process manager agent, which is deployed with an Anywhere fleet, to simplify server process management tasks across computes in all fleets.
- (Optional) A FlexMatch matchmaker to create multi-player matches and set up game sessions for them.
- One or more Amazon GameLift Servers managed fleets, which use Amazon Elastic Compute Cloud (Amazon EC2) instances optimized for multiplayer game hosting.
- One or more Amazon GameLift Servers Anywhere fleets, which use your existing on-premises or other hosting resources, including your configuration management and deployment tooling. (You can optionally use the AWS Systems Manager.)

This roadmap presents a streamlined path to getting your multiplayer game up and running successfully in a hybrid hosting solution with Amazon GameLift Servers. After you have the necessary components in place, you can continue to iterate on game development and customize your hosting solution. As you get closer to launch, see these [Prepare for launch with Amazon GameLift Servers hosting](#) for help with preparing your hosting solution for production-level usage.

Get a jump start with the Amazon GameLift Servers plugin

If you're developing projects with Unreal Engine or Unity, start setting up your game for hosting with the Amazon GameLift Servers plugin. With the plugin, you can add the Amazon GameLift Servers SDKs to your game project and use the guided workflows to build a simple working version of a hybrid hosting solution with both an Anywhere fleet

and an Amazon GameLift Servers managed fleet. You can then use these fundamentals to build on and customize as needed.

Step 1: Prepare your game server to work with Amazon GameLift Servers

Add functionality to your game server so that it can communicate with the Amazon GameLift Servers service when it's deployed for hosting. The same functionality is required if the game server is running on an Amazon GameLift Servers managed fleet or an Anywhere fleet.

- **Get the server SDK for Amazon GameLift Servers (version 5.x) for your game project.** The server SDK is available in C++, C#, and Go. [Download the server SDK for Amazon GameLift Servers](#).
- **Modify your game server code to add server SDK functionality.** For guidance, see [Prepare a game for hosting with Amazon GameLift Servers](#). At a minimum, do the following:
 - Add code to initialize the Amazon GameLift Servers SDK and establish a WebSocket connection with the Amazon GameLift Servers service. Use the server SDK action `InitSdk()`. Include code to specify server parameters when running on an Anywhere fleet compute.
 - Add code to report to the Amazon GameLift Servers service when the server process is ready to host game sessions. Use the server SDK action `ProcessReady()`.
 - Implement the required callback functions `OnProcessTerminate()`, and `OnStartGameSession()`. With these functions, game server processes can maintain a connection with the Amazon GameLift Servers service, initiate a game session when prompted by Amazon GameLift Servers, and respond to a prompt to end the game server process.
 - Add code to report to the Amazon GameLift Servers service when the server process is ending a game session. Use the server SDK action `ProcessEnding()`.
- **Package your game server build.** Create an install script with your build files, dependencies and supporting software. See [Package your game build files](#). We recommend using an Amazon Simple Storage Service (Amazon S3) bucket to store versions of your game build.
- **Test your game server integration.** For this task, we recommend setting up an Amazon GameLift Servers Anywhere fleet with a local workstation, as described in [Set up local testing with Amazon GameLift Servers Anywhere](#). For this step, manually install your game server build onto the test device and start a server process. Use the AWS CLI to request a new game session, and verify that the Amazon GameLift Servers service successfully prompts your server process to start a game session.

Step 2: Prepare your game client to join hosted game sessions

Create a way for your game client to request to join a game session, get connection info, and then connect directly to a hosted game session. The most common approach is to set up backend service functionality that serves as a middleman between your game client and the Amazon GameLift Servers service. This approach protects your hosting resources and gives you greater control over how players are placed into game sessions.

- **Build backend service functionality for hosting.** The backend service communicates with the Amazon GameLift Servers service and delivers connection information to a game client. This functionality includes starting game sessions, placing players into games, and retrieving game session information. For guidance, see [Prepare a game for hosting with Amazon GameLift Servers](#). At a minimum, do the following:
 - Get the AWS SDK for Amazon GameLift Servers and add it to your backend service project. See [Amazon GameLift Servers SDK resources for client services](#).
 - Add code to initialize an Amazon GameLift Servers client and store key settings. See [Set up the Amazon GameLift Servers API](#).
 - Add functionality to call the AWS SDK action `CreateGameSession()` and provide game session connection information to a game client. See [Create a game session on a specific fleet](#).

Calling `CreateGameSession()` is a convenient starting point for requesting new game sessions. After you have a game session placement system in place (see Step 3), you'll replace this code with a call to `StartGameSessionPlacement()` (or `StartMatchmaking()` if you're using FlexMatch).

For guidance on designing your backend service, see [Build a backend service for Amazon GameLift Servers](#).

- **Add functionality to your game client that lets players join a hosted game session.** The game client makes requests to your backend service, not directly to Amazon GameLift Servers. After the backend service provides game session connection information, the game client connects directly with the game session to play the game.
- **Test your game client integration.** You can use the same Amazon GameLift Servers Anywhere fleet with a local workstation for testing.

During the development phase, if you want to test how your game build behaves in an Amazon GameLift Servers managed fleet, we recommend that you also set up a [cloud-based test](#)

[environment](#). This Amazon GameLift Servers Toolkit solution mimics the behavior of a managed fleet but lets you update game server builds with minimal turnaround time.

Step 3: Set up game session placement

Customize how you want Amazon GameLift Servers to process requests for new game session and locate available game servers to host them. Amazon GameLift Servers automatically tracks the availability of all game servers on all fleets. When a game client sends a request to join a game session, Amazon GameLift Servers looks for the "best possible" placement based on a set of defined priorities such as minimum latency, cost, and availability.

- **Create a game session queue for placing new game session with available game servers.** Queues are the primary mechanism for game session placement. For guidance, see [Create a game session queue](#).
 - At minimum, add your Anywhere fleets as destinations in your queue. All other settings are optional customizations.
- **In your backend service code, convert the `CreateGameSession()` call to `StartGameSessionPlacement()`.** See [Create a game session in a multi-location queue](#).
- **Create a mechanism to notify a game client when a game session is ready to join.** While in development, you can poll for game session status using a call to `DescribeGameSessionPlacement`. Before using a queue to process high volumes, however, you'll need to enable event notifications. See [Set up event notification for game session placement](#).
- (Optional) **Add FlexMatch matchmaking components.** For guidance, see the [Amazon GameLift Servers FlexMatch developer guide](#).

Step 4: Set up an Anywhere fleet with the Amazon GameLift Servers Agent

Up to this point you've been working with local devices (registered as Anywhere fleet computes) to test and iterate on your game components. The next step is to set up the type of fleets you'll need for a production system. Start with an Anywhere fleet, and add the Amazon GameLift Servers Agent to manage some key on-compute host management tasks. For more details, see [Work with the Amazon GameLift Servers Agent](#).

- **Get the Amazon GameLift Servers Agent and add it to your game server install package.** Get and build the Agent source code, available in the [Amazon GameLift Servers Agent GitHub repository](#). Place the resulting JAR file executable into the same directory as your game build executable.

- **Modify your startup script for the Agent as needed.** Ensure that Agent executable launches as soon as a compute starts running. See the readme file in the Agent repo for help with installing and running the Agent on your hosting computes. Your launch command should include options to specify, at minimum, the Anywhere fleet ID and AWS Region, a custom location, and a compute name.

The Agent automatically handles the following tasks for you, so if you've been handling these tasks with scripts, you can remove them:

- Calls `RegisterCompute()` to add the compute to an Anywhere fleet.
- Calls `GetComputeAuthToken()` to authenticate game servers when they connect to the Amazon GameLift Servers service. The Agent manages getting and refreshing the auth token, which can be used by all game server processes that are running on the compute.
- Starts new server processes on the compute based on a set of runtime instructions.
- **Create a runtime configuration for computes in your Anywhere fleet.** At minimum, specify the launch path for your game server executable. You can use the Amazon GameLift Servers console or the AWS CLI to create or modify runtime instructions for the fleet. The Agent carries out these instructions and periodically requests updates from the Amazon GameLift Servers service.
- **Set up or modify your game session queue as needed.** Create a new queue (or update an existing one) and designate a destination for the Anywhere fleet deployed with the Amazon GameLift Servers Agent.
- **Test the Agent integration with your Anywhere fleets.** Check that the Agent is properly starting server processes based on the runtime configuration.

Step 5: Create a managed cloud-based fleet

Create an Amazon GameLift Servers managed EC2 fleet to supplement your Anywhere fleet. If you set up a cloud-based test environment in Step 2 to speed up development, plan to create a managed fleet after you've completed most of your game development and testing. You need a fully managed fleet to configure and test additional settings such as automatic capacity scaling.

- **Package your game server build and upload to Amazon GameLift Servers.** Create an install script with your build files, dependencies and supporting software. You can use the same build software with both your Anywhere and managed fleets. See [Create a game server build for Amazon GameLift Servers](#). You can upload your build to Amazon GameLift Servers using either the console or the AWS CLI.

Before uploading your build, decide in what AWS Region you want to create the managed fleet. You must upload the build to the same Region. For more on choosing a fleet location, see [Geographic locations](#).

- **Create a managed EC2 fleet.** You can use the Amazon GameLift Servers console or the AWS CLI to create a managed fleet. When you create a fleet, Amazon GameLift Servers immediately begins deploying your game server build for hosting. You can configure many aspects of a managed fleet. For guidance, see [Create an Amazon GameLift Servers managed EC2 fleet](#). At minimum, do the following:
 - Give the fleet a name and specify which uploaded game build to deploy.
 - Choose On-Demand Instances for your fleet and select an instance type that's available in your fleet location. Spot fleets are a valuable option but require additional design and configuration.
 - Create a runtime configuration with similar settings as you used with the Anywhere fleet. At minimum, specify the launch path for your game server executable.
 - Specify port settings to allow inbound traffic to access your game servers.
- **Add the managed fleet to your shared game session queue.** Update the queue from Step 4 so that it includes destinations for both the managed fleet and the Anywhere fleet deployed with the Amazon GameLift Servers Agent.
- **Test game hosting with your managed fleets.** At this point you should be able to test the entire hosting cycle, with a game client requesting a game session, getting connection info, and successfully connecting to a game session.

Step 6: Customize your fleets

As you prepare for game launch, you'll need to fine-tune your hosting solutions. Some of the decisions to consider include:

- For Anywhere fleets, automate the process of starting and shutting down computes as needed, including installing and running game server software. Recycling computes is useful to ensure that they are updated regularly, and shutting down computes can save costs when they're not needed.
- If your game server needs to communicate other AWS resources, set up IAM roles to manage access. See [Connect your Amazon GameLift Servers hosted game server to other AWS resources](#).

- Determine where geographically you want to position game servers. Add remote locations to your managed fleets. See [Hosting resource customizations](#).
- For managed fleets, consider using Spot fleets for cost savings. See [Reduce game hosting costs with Spot fleets](#).
- Optimize fleet performance by selecting compute resource configurations, then configure your the runtime instructions to run the optimal number of server processes per compute. Do this for both Anywhere fleets and managed fleets. See [Optimize game server runtime configuration on managed Amazon GameLift Servers](#).
- Experiment with game session placement options for managed fleets, including customizing prioritization settings. See [Customize a game session queue](#).
- For managed fleets, set up automatic capacity scaling to meet expected player demand. See [Scaling game hosting capacity with Amazon GameLift Servers](#).
- For Anywhere fleets, create mechanisms to handle manual or automated capacity scaling to meet expected player demand.
- Design and implement failover to other resources if needed. Set up standby fleets in other AWS Regions and modify queues and auto scaling to handle failovers if needed.
- Set up hosting observability tools, including analytics and logging. See [Monitoring Amazon GameLift Servers](#). Create metric groups to aggregate analytics for all your hosting resources.
- Automate your deployment using [infrastructure as code \(IaC\)](#). See [Manage Amazon GameLift Servers hosting resources using CloudFormation](#).

Amazon GameLift Servers supports the use of AWS CloudFormation templates for any deployment-specific configurations. You can also use the AWS Cloud Development Kit (AWS CDK) to define your Amazon GameLift Servers resources. For more information about the AWS CDK, see the [AWS Cloud Development Kit \(AWS CDK\) Developer Guide](#).

To manage the deployment of your CloudFormation stacks, we recommend using continuous integration and continuous delivery (CI/CD) tools and services such as AWS CodePipeline. These tools help you deploy automatically or with approval whenever you build game server binary. With a CI/CD tool or service, resources deployment for a new game server version can look like this:

- Build and test your game server binary.
- Upload the binary to Amazon GameLift Servers.
- Deploy new fleets with the new build.

- Add the new fleets to your game session queue and remove the fleets with the previous build version.
- When the fleets with the previous build are no longer hosting active game sessions, delete the CloudFormation stacks of those fleets.

Prepare a game for hosting with Amazon GameLift Servers

Before you can deploy your game for hosting on Amazon GameLift Servers, you need to prepare both your game server and game client components. This involves integrating your game server with the Amazon GameLift Servers Server SDK and building a backend service that handles player authentication and game session management.

The Server SDK integration enables your game server to communicate with the Amazon GameLift Servers service, report its health status, and manage game sessions. The backend service acts as an intermediary between your game clients and the Amazon GameLift Servers service, handling player requests and coordinating game session placement.

Topics

- [Prepare your Unreal or Unity game with the Amazon GameLift Servers plugin](#)
- [Integrate a game server with Amazon GameLift Servers](#)
- [Package a game server build for deployment](#)
- [Game client/server interactions with Amazon GameLift Servers](#)

Prepare your Unreal or Unity game with the Amazon GameLift Servers plugin

The Amazon GameLift Servers plugin is a full-featured add-on to your Unreal or Unity game engine. It guides you through the basic steps to deploy your game for hosting with Amazon GameLift Servers. With the plugin's tool set and workflows, you can work in your game engine development environment to prepare your game server for hosting, set up hosting on a local machine for testing, create a simple backend service, and deploy your game server to managed cloud-based hosting.

Use the plugin to experience working with Amazon GameLift Servers and get a game hosting solution up and running fast. You can work with sample game assets or your own game project. The plugin automates a number of steps so that you can quickly build a simple working solution.

When you complete the plugin's guided workflows, you'll be able to connect a game client to live hosted game sessions through Amazon GameLift Servers. After using the plugin to create a simple hosting solution, you can customize your solution to meet the needs of your game.

The plugin is available for the following game engines:

- Unreal Engine
- Unity

The plugin includes these components for each game engine:

- Plugin modules for the game engine editor. When the plugin is installed, a new main menu button gives you access to Amazon GameLift Servers functionality.
- Libraries for the Amazon GameLift Servers service API with client-side functionality.
- Libraries for the Amazon GameLift Servers server SDK (version 5).
- Sample assets for use with testing a server integration.
- Editable configurations, in the form of CloudFormation templates, that define your game server solution.

Topics

- [Plugin workflow](#)
- [Amazon GameLift Servers plugin for Unreal Engine](#)
- [Amazon GameLift Servers plugin for Unity \(server SDK 5.x\)](#)
- [Amazon GameLift Servers plugin for Unity for server SDK 4](#)

Plugin workflow

The following steps describe a typical path to preparing and deploying your game project on Amazon GameLift Servers. You complete these steps by working in the game engine editor and your game code.

1. Create a user profile that links to your AWS account user and provides access credentials with permissions to use Amazon GameLift Servers.
2. Set up related AWS resources that the plugin uses in the hosting solution (referred to as "bootstrapping").

3. Add server code to your project to establish communication between a running game server and the Amazon GameLift Servers service.
4. Add client code to your project that lets game clients send requests to Amazon GameLift Servers to start new game sessions and then connect to them.
5. Use the Anywhere workflow to set up your local workstation as an Anywhere compute and host your game server. Launch your game server and client locally through the plugin, connect to a game session, and test the integration.
6. Use the Managed EC2 workflow to upload your game server to Amazon GameLift Servers and deploy a simple but complete cloud hosting solution. Launch your game client locally through the plugin, request a game session and connect to it, and play your game.

When working in the plugin, you'll create and use AWS resources. These actions might incur charges to the AWS account in use. If you're new to AWS, these actions might be covered under the [AWS Free Tier](#).

Amazon GameLift Servers plugin for Unreal Engine

This plugin adds the Amazon GameLift Servers C++ server SDK and tools to the UE editor. Use the guided UI workflows to integrate server SDK functionality into your game project and deploy an Amazon GameLift Servers hosting solution for your game server.

With the plugin, you can build a basic working hosting solution and then optimize and customize as needed. Set up an Amazon GameLift Servers Anywhere fleet with your local workstation as a host. For cloud hosting with managed EC2 or managed container fleets, deploy your game server with a complete solution to manage game session requests and client connections.

Topics

- [Install the plugin for your Unreal game project](#)
- [Next steps: Customize your game hosting solution](#)
- [Plugin for Unreal: Set up an AWS user profile](#)
- [Plugin for Unreal: Integrate your game code](#)
- [Plugin for Unreal: Host your game locally with Amazon GameLift Servers Anywhere](#)
- [Plugin for Unreal: Deploy your game to a managed EC2 fleet](#)
- [Plugin for Unreal: Deploy your game to a managed container fleet](#)

Install the plugin for your Unreal game project

[Get the Amazon GameLift Servers plugin for Unreal Engine from GitHub](#)

See the GitHub repository readme for information about how to install the plugin in your Unreal Editor for a game project.

The plugin includes these components:

- Plugin modules for the UE editor. When the plugin is installed, a new main menu button gives you access to Amazon GameLift Servers functionality.
- C++ libraries for the Amazon GameLift Servers service API. Use API functionality in a client-side backend service to help game clients request game sessions and send/retrieve game session information.
- Unreal libraries for the Amazon GameLift Servers server SDK (version 5). Use the server SDK in your game server code to manage communication between hosted game server processes and the Amazon GameLift Servers service.
- Content for testing, including a startup game map and two testing maps with basic blueprints and UI elements for use with testing a server integration.
- Editable configurations, in the form of CloudFormation templates, that the plugin uses when deploying your game server for hosting.

This plugin uses AWS CloudFormation templates to deploy hosting solutions for common gaming scenarios. Use these solutions as provided or customize them as needed for your games.

Next steps: Customize your game hosting solution

Using the plugin's guided workflows is a good way to get up and running fast with an Amazon GameLift Servers hosting solution. With the plugin, you can set up basic versions of each of your solution's components.

When you're ready, you can build on this basic solution by customizing each component, and fine-tuning your solution as you prepare for game launch. Consider these options:

- Modify your fleets and fleet configurations. See [Hosting resource customizations](#).
- Customize your game session queue configuration. See [Customize a game session queue](#):
- Add functionality to your game server and game client. See [Integrate a game server with Amazon GameLift Servers](#) and [Integrate Amazon GameLift Servers game client functionality](#).

- Customize your backend service. See [Build a backend service for Amazon GameLift Servers](#).
- Set up automatic capacity scaling to meet expected player demand. See [Scaling game hosting capacity with Amazon GameLift Servers](#).
- Set up hosting observability tools, including analytics and logging. See [Monitoring Amazon GameLift Servers](#).
- Automate your deployment using [infrastructure as code \(IaC\)](#). The plugin's guided workflows for managed solutions use AWS CloudFormation templates. You can customize these as needed. See [Manage Amazon GameLift Servers hosting resources using CloudFormation](#).

Topics

- [Plugin for Unreal: Set up an AWS user profile](#)
- [Plugin for Unreal: Integrate your game code](#)
- [Plugin for Unreal: Host your game locally with Amazon GameLift Servers Anywhere](#)
- [Plugin for Unreal: Deploy your game to a managed EC2 fleet](#)
- [Plugin for Unreal: Deploy your game to a managed container fleet](#)

Plugin for Unreal: Set up an AWS user profile

After installing the plugin, set up a user profile with a valid AWS account. You can maintain multiple profiles in the plugin, but you can have only one profile selected at a time. Whenever you work in the plugin, select a profile to use. Each workflow page displays the currently selected profile.

Maintaining multiple profiles gives you the ability to switch between different hosting deployments. For example, you might set up profiles that use the same AWS account but deploy to different AWS Regions. Or you might set up profiles with different AWS accounts or users and permission sets.

Note

If you've installed the AWS CLI on your workstation and have a profile already configured, the Amazon GameLift Servers plugin will detect it and list it as an existing profile. The plugin automatically selects any profile named `[default]`. You can use an existing profile or create a new one.

All profiles must be bootstrapped to set up some required AWS resources under your account user.

To manage your AWS profiles

1. In the Unreal editor main toolbar, choose the Amazon GameLift Servers menu, and select **AWS Access Credentials**. This action opens the Amazon GameLift Servers plugin to the page **Set up your user profile**.
2. Use the buttons to create a new AWS account or set up a user profile for an AWS account that you already have.
3. If you don't already have a user profile, you're prompted to enter profile details and create a new profile. Provide the following information:
 - An AWS account. If you create a new AWS account, use the link to the AWS Management Console and follow the prompts. See [Create an AWS account](#) for more details.
 - An AWS user with permissions to use Amazon GameLift Servers and other required AWS services. See [Set up an AWS user account](#) for instructions on setting up an AWS Identity and Access Management (IAM) user with Amazon GameLift Servers permissions and programmatic access with long-term credentials.

Note

The "Set up an IAM service role" step is optional for plugin users. You only need an IAM service role if your game server accesses other AWS resources, such as Amazon S3 or DynamoDB.

- Credentials for your AWS user. These credentials consist of an AWS access key ID and AWS secret key. See [Get your access keys](#) for more details.
 - AWS region. This is a geographic location where you want to create your AWS resources for hosting. During development, we recommend using a region close to your physical location. Choose a region from the list of [supported AWS regions](#).
4. If the plugin detects existing profiles, it displays a list of available profiles. Select an existing profile from the list, or choose **Add another profile** to create a new one.

Bootstrap a user profile

All profiles must be bootstrapped to use with the Amazon GameLift Servers plugin. Bootstrapping creates an Amazon S3 bucket specific to the profile. It's used to store project configurations, build artifacts, and other dependencies. Buckets are not shared between other profiles.

Bootstrapping involves creating new AWS resources and might incur costs.

To bootstrap your profile:

1. On the **AWS Access Credentials** page, check the bootstrap status of the user profile that you want to use. If the profile's bootstrap status is "Inactive" and there's no S3 bucket listed, you need to bootstrap the profile.
2. Select the profile you want to use and choose **Bootstrap profile**.
3. Wait for bootstrap status to change to "Active". This can take a few minutes.

Plugin for Unreal: Integrate your game code

Before you can deploy your game server to a fleet, you need to make a series of updates to game code and package game components for use with the Amazon GameLift Servers service.

This topic walks through the steps for doing a minimal integration. For server integration, use the provided code sample to update your project's game mode.

- [Set up build targets and module rules](#)
- [Update your game server code](#)
- [Integrate your client game map](#)
- [Package your game components](#)

Set up build targets and module rules

Modify your game project files to properly generate build components for use with Amazon GameLift Servers.

To add client and server build targets:

1. Open your game project's code files and locate the file `.../Games/[your application name]Source/[your application name]Target.cs` file. Example: `.../Source/GameLiftUnrealAppTarget.cs`. (If you use Visual Studio, open the project's `.sln` file.)

2. Copy this file to create two new target files in the Source/ directory.

- Client target – Rename the new file to *[your application name]*Client.Target.cs. Edit the contents to update the class name and target type values, as illustrated in the following sample code:

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

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

- Server target – Rename the new file to *[your application name]*Server.Target.cs. Edit the contents to update the class name and target type values, as illustrated in the following sample code:

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

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

3. Regenerate your project files. If you're using Visual Studio, you can right-click your game project's .uproject file and select **Generate Visual Studio Project Files**.

To update the game project module rules:

Update the game project's module rules to take a dependency on the plugin.

1. Open your game project's code files and locate the file `.../Games/[your application name]Source/[your application name].Build.cs` file. Example: `.../Source/GameLiftUnrealApp.Build.cs`. (If you use Visual Studio, open the project's `.sln` file.)
2. Locate the `ModuleRules` class and update as illustrated in the following sample code:

```
using UnrealBuildTool;

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

3. After creating the new target files and modifying the module rules, rebuild your game project.

Update your game server code

Update your game server code to enable communication between a game server process and the Amazon GameLift Servers service. Your game server must be able to respond to requests from Amazon GameLift Servers, such as to start and stop new game sessions.

To add server code for Amazon GameLift Servers

1. In your code editor, open the solution (.sln) file for your game project, usually found in the project root folder. For example: GameLiftUnrealApp.sln.
2. With the solution open, locate the project game mode header file: [project-name]GameMode.h file. For example: GameLiftUnrealAppGameMode.h.
3. Change the header file to align with the following code. Be sure to replace "GameLiftServer" with your own project name. These updates are specific to the game server; we recommend that you make a backup copy of the original game mode files for use with your client.

Example gameMode.h code

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

#pragma once

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

struct FProcessParameters;

DECLARE_LOG_CATEGORY_EXTERN(GameServerLog, Log, All);

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

public:
    AGameLiftUnrealAppGameMode();

protected:
    virtual void BeginPlay() override;

private:
    void InitGameLift();

private:
    TSharedPtr<FProcessParameters> ProcessParameters;
```

```
};
```

- Open the related source file `[project-name]GameMode.cpp` file (for example `GameLiftUnrealAppGameMode.cpp`). Change the code to align with the following example code. Be sure to replace "GameLiftUnrealApp" with your own project name. These updates are specific to the game server; we recommend that you make a backup copy of the original file for use with your client.

The following example code shows how to add the minimum required elements for server integration with Amazon GameLift Servers:

- Initialize an Amazon GameLift Servers API client. The `InitSDK()` call with server parameters is required for an Amazon GameLift Servers Anywhere fleet. When you connect to an Anywhere fleet, the plugin stores the server parameters as console arguments. The sample code can access the values at runtime.
- Implement required callback functions to respond to requests from the Amazon GameLift Servers service, including `OnStartGameSession`, `OnProcessTerminate`, and `onHealthCheck`.
- Call `ProcessReady()` with a designated port to notify the Amazon GameLift Servers service when ready to host game sessions.

Example game server code

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

#include "GameLiftUnrealAppGameMode.h"

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

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

#include "GenericPlatform/GenericPlatformOutputDevices.h"

DEFINE_LOG_CATEGORY(GameServerLog);
```

```

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

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

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

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

#ifdef WITH_GAMELIFT
    InitGameLift();
#endif
}

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

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

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

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

```

```

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

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

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

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

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

```

```

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

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

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

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

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

        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_YELLOW);
        UE_LOG(GameServerLog, Log, TEXT(">>>> WebSocket URL: %s"),
*ServerParametersForAnywhere.m_webSocketUrl);

```

```

        UE_LOG(GameServerLog, Log, TEXT(">>>> Fleet ID: %s"),
*ServerParametersForAnywhere.m_fleetId);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Process ID: %s"),
*ServerParametersForAnywhere.m_processId);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Host ID (Compute Name): %s"),
*ServerParametersForAnywhere.m_hostId);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Auth Token: %s"),
*ServerParametersForAnywhere.m_authToken);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Aws Region: %s"),
*ServerParametersForAnywhere.m_awsRegion);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Access Key: %s"),
*ServerParametersForAnywhere.m_accessKey);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Secret Key: %s"),
*ServerParametersForAnywhere.m_secretKey);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Session Token: %s"),
*ServerParametersForAnywhere.m_sessionToken);
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
    }

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

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

    ProcessParameters = MakeShared<FProcessParameters>();

```

```

    //When a game session is created, Amazon GameLift Servers sends an activation
    request to the game server and passes along the game session object containing game
    properties and other settings.
    //Here is where a game server should take action based on the game session object.
    //Once the game server is ready to receive incoming player connections, it should
    invoke GameLiftServerAPI.ActivateGameSession()
    ProcessParameters->OnStartGameSession.BindLambda([=]
(Aws::GameLift::Server::Model::GameSession InGameSession)
    {
        FString GameSessionId = FString(InGameSession.GetGameSessionId());
        UE_LOG(GameServerLog, Log, TEXT("GameSession Initializing: %s"),
*GameSessionId);
        GameLiftSdkModule->ActivateGameSession();
    });

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

```



```

        error.m_errorMessage.IsEmpty() ? TEXT("Unknown error") :
*error.m_errorMessage);
    }
    });

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

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

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

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

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

//Here, the game server tells Amazon GameLift Servers where to find game session
log files.
//At the end of a game session, Amazon GameLift Servers uploads everything in the
specified

```

```

//location and stores it in the cloud for access later.
TArray<FString> Logfiles;
Logfiles.Add(TEXT("GameLiftUnrealApp/Saved/Logs/server.log"));
ProcessParameters->logParameters = Logfiles;

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

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

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

```

Integrate your client game map

The startup game map contains blueprint logic and UI elements that already include basic code to request game sessions and use connection information to connect to a game session. You can use the map as is or modify these as needed. Use the startup game map with other game assets, such as the Third Person template project provided by Unreal Engine. These assets are available in Content Browser. You can use them to test the plugin's deployment workflows, or as a guide to create a custom backend service for your game.

The startup map has the following characteristics:

- It includes logic for both an Anywhere fleet and a managed EC2 fleet. When you run your client, you can choose to connect to either fleet.
- Client functionality includes find a game session (`SearchGameSessions()`), create a new game session (`CreateGameSession()`), and join a game session directly.
- It gets a unique player ID from your project's Amazon Cognito user pool (this is part of a deployed Anywhere solution).

To use the startup game map

1. In the UE editor, open the **Project Settings, Maps & Modes** page, and expand the **Default Maps** section.
2. For **Editor Startup Map**, select "StartupMap" from the dropdown list. You might need to search for the file, which is located in `... > Unreal Projects/[project-name]/Plugins/Amazon GameLift Servers Plugin Content/Maps`.

Note

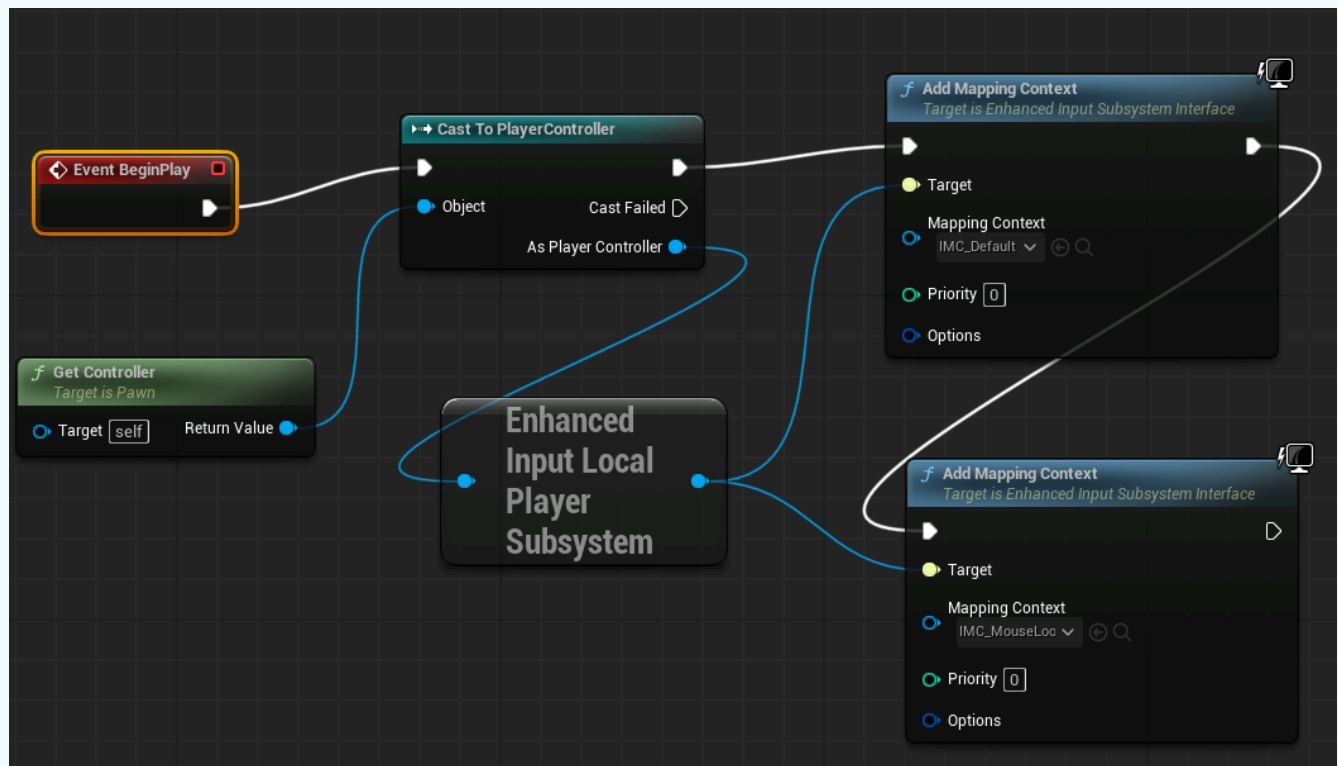
To find the sample startup map, select the settings icon and choose **Show Plugin Content**.

3. For **Game Default Map**, select the same "StartupMap" from the dropdown list.
4. For **Server Default Map**, select "Lv1_ThirdPerson" for Unreal Engine 5.6 or later, or "ThirdPersonMap" for earlier versions. This is a default map included in your game project. This map is designed for two players in the game.
5. Open the details panel for the server default map. Set **GameMode Override** to "None".
6. Expand the **Default Modes** section, and set **Global Default Server Game Mode** to the game mode you updated for your server integration.

After you've made these changes to your project, you're ready to build your game components.

Note

For Unreal Engine 5.6 or later, if you cannot move the character after connecting to the game server, update the BP_ThirdPersonCharacter blueprint to add input mapping context for IMC_Default and IMC_MouseLook as shown below:



Package your game components

To package your game server and game client builds

1. Open your game project in a source-built version of the Unreal Engine editor.
2. If using Unreal Engine 5.6 or later, go to **Edit, Project Settings, Packaging**. Find **Cook everything in the project content directory** and enable it.
3. Use the editor to package your game client and server builds.
 - a. Choose a target. Go to **Platforms, Windows** and select one of the following:
 - Server: [your-application-name]Server
 - Client: [your-application-name]Client
 - b. Start the build. Go to **Platform, Windows, Package Project**.

Each packaging process generates an executable: [your-application-name]Client.exe or [your-application-name]Server.exe.

In the plugin, set the paths to the client and server build executables on your local workstation.

Plugin for Unreal: Host your game locally with Amazon GameLift Servers Anywhere

Use this workflow to set up your local workstation as a game server host using an Anywhere fleet. You can use it to test your game server integration before deploying to a cloud-based managed fleet. It can also be useful for local testing during iterative game development.

To start the Amazon GameLift Servers Anywhere workflow:

- In the Unreal editor main toolbar, choose the Amazon GameLift Servers menu, and select **Host with Anywhere**. This action opens the plugin page **Deploy Anywhere**, which presents a six-step process to integrate, build, and launch your game components.

Step 1: Set your profile

Choose the profile you want to use when following this workflow. The profile you select impacts all steps in the workflow. All resources you create are associated with the profile's AWS account and are placed in the profile's default AWS Region. The profile user's permissions determine your access to AWS resources and actions.

To set a user profile

1. Select a profile from the dropdown list of available profiles. If you don't have a profile yet or want to create a new one, go to the **Amazon GameLift** menu and choose **Set AWS User Profiles**.
2. If bootstrap status is not "Active", choose **Bootstrap profile** and wait for the status to change to "Active".

Step 2: Set up your game code

In this step, prepare your game server and game client builds to work with Amazon GameLift Servers. If you haven't yet integrated your game code, see [Plugin for Unreal: Integrate your game code](#). Enter the paths to your game executables on your local workstation.

- Game server: Integrate your game server with the server SDK for Amazon GameLift Servers and package your game server build. For instructions, see [Plugin for Unreal: Integrate your game code](#). The game server must be integrated with the server SDK in order to establish communication with the Amazon GameLift Servers service and respond to prompts to start new game sessions and accept game client connections.

- **Game client:** At minimum, you need a game client that can connect to your game server using IP address and port information. If you don't yet have your game client components set up for Amazon GameLift Servers, you can use the AWS CLI tool to manually request new game sessions, get connection information, and use that information to connect the game client.

At some point, you'll need to have a backend service to send new game sessions requests to the Amazon GameLift Servers service and relay connection information back to a game client. You can use the test maps included with the plugin to add client Amazon GameLift Servers functionality to your game project. For help with building a custom solution, see [Integrate Amazon GameLift Servers game client functionality](#).

Step 3: Connect to an Anywhere fleet

In this step, you designate an Anywhere fleet to use. An Anywhere fleet defines a collection of compute resources, which can be located anywhere, for game server hosting.

- If the AWS account you're currently using has existing Anywhere fleets, open the Fleet name dropdown field and choose a fleet. This dropdown only shows the Anywhere fleets in the AWS Region for the currently active user profile.
- If there are no existing fleets—or you want to create a new one, choose Create new Anywhere fleet and provide a fleet name.

After you've chosen an Anywhere fleet for your project, Amazon GameLift Servers verifies that fleet status is active and displays the fleet ID. You can track progress of this request in the Unreal editor's output log.

Step 4: Register your workstation

In this step, you register your local workstation as a compute resource in the new Anywhere fleet.

To register your workstation as an Anywhere compute

1. Enter a compute name for your local machine. If you add more than one compute in the fleet, the names must be unique.
2. Provide an IP address for your local machine. This field defaults to your machine's public IP address. You can also use localhost (127.0.0.1) as long as you're running your game client and server on the same machine.

3. Choose Register compute. You can track progress of this request in the Unreal editor's output log.

In response to this action, Amazon GameLift Servers verifies that it can connect to the compute and returns information about the newly registered compute. It also creates the console arguments that your game executables need when initializing communication with the Amazon GameLift Servers service.

Step 5: Generate auth token

Game server processes that are running on your Anywhere compute need an authentication token to make calls to the Amazon GameLift Servers service. The plugin automatically generates and stores an auth token for the Anywhere fleet whenever you launch the game server from the plugin. The auth token value is stored as a command line argument, which your server code can retrieve at runtime.

The code examples given above also allow you to use [AWS Signature Version 4 \(SigV4\) for API requests](#). SigV4 is the AWS signing protocol for adding authentication information to API requests.

You do not have to take any action in this step.

Step 6: Launch game

At this point, you've completed all of the tasks needed to launch and play your multiplayer game on a local workstation using Amazon GameLift Servers.

To play your hosted game

1. Launch your game server. The game server will notify Amazon GameLift Servers when it is ready to host game sessions.
2. Launch your game client and use the new functionality to start a new game session. This request is sent to Amazon GameLift Servers via the new backend service. In response, Amazon GameLift Servers, calls the game server, running on your local machine, to start a new game session. When the game session is ready to accept players, Amazon GameLift Servers provides connection information for the game client to join the game session.

Plugin for Unreal: Deploy your game to a managed EC2 fleet

In this workflow, deploy your game for hosting on cloud-based compute resources managed by Amazon GameLift Servers. Upload your integrated game server build to the Amazon GameLift

Servers service for deployment. If you haven't yet integrated your game code, see [Plugin for Unreal: Integrate your game code](#). When this workflow is complete, you'll have a working game client that can connect to your game servers in the cloud.

To start the Amazon GameLift Servers managed Amazon EC2 workflow:

- In the Unreal editor main toolbar, choose the Amazon GameLift Servers menu, and select **Host with Managed EC2**. This action opens the plugin page **Deploy Amazon EC2 Fleet**, which presents a six-step process to integrate, build, deploy, and launch your game components.

Step 1: Set your profile

Choose the profile you want to use when following this workflow. The profile you select impacts all steps in the workflow. All resources you create are associated with the profile's AWS account and are placed in the profile's default AWS Region. The profile user's permissions determine your access to AWS resources and actions.

To set a user profile

1. Select a profile from the dropdown list of available profiles. If you don't have a profile yet or want to create a new one, go to the **Amazon GameLift** menu and choose **Set AWS User Profiles**.
2. If bootstrap status is not "Active", choose **Bootstrap profile** and wait for the status to change to "Active".

Step 2: Set up your game code

In this step, prepare your game server and game client builds to work with Amazon GameLift Servers C++ server SDK for Unreal. If you haven't yet integrated your game code and built game client and server executables, see [Plugin for Unreal: Integrate your game code](#). Enter the paths to your game executables on your local workstation.

This step in the workflow the plugin provides links to instructions and source code for setting up a source-built version of the Unreal Editor. You need to use the source-built version when building your client and server components.

After building game server that's integrated with the server SDK, complete the following tasks to prepare it for uploading to Amazon GameLift Servers for hosting.

To prepare your server build for cloud deployment (Windows)

In the `WindowsServer` folder, where the Unreal editor stores your server build files by default, make the following additions:

1. **Copy the server build install script into the root of the `WindowsServer` folder.** The install script is included in the plugin download. Look for the file `[project-name]/Plugins/Resources/CloudFormation/extra_server_resources/install.bat`. Amazon GameLift Servers uses this file to install the server build onto your hosting computes.
2. **Copy the `VC_redist.x64.exe` file into the root of the `WindowsServer` folder.** You can skip this step if you're using Unreal Engine version 5.6 or later. This file is included in your Visual Studio installation. It is commonly located at `C:/Program Files (x86)/Microsoft Visual Studio/2019/Professional/VC/Redist/MSVC/v142`.
3. **Add the OpenSSL library files to your game server build..** You can skip this step if your game server is integrated with server SDK 5.3 or later. This version is included in the Amazon GameLift Servers plugin for Unreal, version 3.0 or later.

Manually locate and copy the OpenSSL libraries to your game build package directory at `<YourGame>/Binaries/Win64`. You **must** use the same OpenSSL version that your Unreal Engine 5 version uses. Game builds that are deployed with the wrong OpenSSL libraries won't be able to communicate with the Amazon GameLift Servers service.

Look for the OpenSSL libraries in your game engine source. The location varies depending on your development environment:

On Windows:

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

On Linux:

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

To prepare your server build for cloud deployment (Linux)

For more detailed instructions on preparing a game server built for Linux, see [Building the server SDK for Amazon GameLift Servers for Unreal Engine 5 on Amazon Linux](#).

1. **Designate a working directory to organize your build files.** The working directory's structure is deployed as is onto each hosting compute. Add your Linux-built game server and all dependent files.
2. **Create a server build install script in the root of your working directory.** If needed, create an `install.sh` file and add any commands needed to properly install your game server build. Amazon GameLift Servers uses this file to install the server build onto each EC2 hosting resource.
3. **Add the OpenSSL library files to your game server build.** You can skip this step if your game server is integrated with server SDK 5.3 or later.

Manually locate and copy the libraries. You **must** use the same OpenSSL version that your Unreal Engine 5 version uses. Game builds that are deployed with the wrong OpenSSL libraries won't be able to communicate with the Amazon GameLift Servers service.

- a. Look for the OpenSSL libraries in your game engine source. The location varies depending on your development environment:

On Windows:

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

On Linux:

- `Engine/Source/Thirdparty/OpenSSL/1.1.1n/include/libssl.so.1.1`
 - `Engine/Source/Thirdparty/OpenSSL/1.1.1n/include/libcrypto.so.1.1`
- b. When you locate the OpenSSL libraries, copy them to your game build package directory at `<YourGame>/Binaries/Linux`.

Step 3: Select deployment scenario

In this step, you choose the game hosting solution that you want to deploy at this time. You can have multiple deployments of your game, using any of the scenarios.

- **Single-region fleet:** Deploys your game server to a single fleet of hosting resources in the active profile's default AWS region. This scenario is a good starting point for testing your server integration with AWS and server build configuration. It deploys the following resources:
 - AWS fleet (On-Demand) with your game server build installed and running.
 - Amazon Cognito user pool and client to enable players to authenticate and start a game.
 - API gateway authorizer that links user pool with APIs.
 - WebACL for throttling excessive player calls to API gateway.
 - API gateway + Lambda function for players to request a game slot. This function calls `CreateGameSession()` if none are available.
 - API gateway + Lambda function for players to get connection info for their game request.
- **FlexMatch fleet:** Deploys your game server to a set of fleets and sets up a FlexMatch matchmaker with rules to create player matches. This scenario uses low-cost Spot hosting with a multi-fleet, multi-location structure for durable availability. This approach is useful when you're ready to start designing a matchmaker component for your hosting solution. In this scenario, you'll create the basic resources for this solution, which you can customize later as needed. It deploys the following resources:
 - FlexMatch matchmaking configuration and matchmaking rule set to accept player requests and form matches.
 - Three AWS fleets with your game server build installed and running in multiple locations. Includes two Spot fleets and one On-Demand fleet as a backup.
 - AWS game session placement queue that fulfills requests for proposed matches by finding the best possible hosting resource (based on viability, cost, player latency, etc.) and starting a game session.
 - Amazon Cognito user pool and client to enable players to authenticate and start a game.
 - API gateway authorizer that links user pool with APIs.
 - WebACL for throttling excessive player calls to API gateway.
 - API gateway + Lambda function for players to request a game slot. This function calls `StartMatchmaking()`.
 - ~~API gateway + Lambda function for players to get connection info for their game request.~~

- Amazon DynamoDB tables to store matchmaking tickets for players and game session information.
- SNS topic + Lambda function to handle GameSessionQueue events.

Step 4: Set game parameters

In this step, you describe your game for uploading to AWS;

- Server build name: Provide a meaningful name for your game server build. AWS uses this name to refer to the copy of your server build that's uploaded and used for deployments.
- Server build OS: Enter the operating system that your server is built to run on. This tells AWS what type of compute resources to use to host your game.
- Game server folder: Identify the path to your local server build folder.
- Game server build: Identify the path to the game server executable.
- Game client path: Identify the path to the game client executable.
- Client configuration output: This field needs to point to a folder in your client build that contains your AWS configuration. Look for it in the following location: `[client-build]/[project-name]/Content/CloudFormation`.

Step 5: Deploy scenario

In this step, you deploy your game to a cloud hosting solution based on the deployment scenario you chose. This process can take several minutes while AWS validates your server build, provisions hosting resources, installs your game server, launches server processes, and gets them ready to host game sessions.

To start deployment, choose **Deploy CloudFormation**. You can track the status of your game hosting here. For more detailed information, you can sign in to the AWS Management console for AWS and view event notifications. Be sure to sign in using the same account, user, and AWS Region as the active user profile in the plugin.

When deployment is complete, you have your game server installed on an AWS EC2 instance. At least one server process is running and ready to start a game session.

Step 6: Launch client

At this point, you've completed all of the tasks needed to launch and play your multiplayer game hosted with Amazon GameLift Servers. To play the game, launch an instance of your game client.

If you deployed the single fleet scenario, you can open a single client instance with one player, enter the server map and move around. Open additional instances of the game client to add a second player to the same server game map.

If you deployed the FlexMatch scenario, the solution waits for at least two clients to be queued for game session placement before the players can enter the server map.

Plugin for Unreal: Deploy your game to a managed container fleet

Use this guided plugin workflow to create a container image for your game server and deploy it to a container-based hosting solution. If you haven't yet integrated your game code, see [Plugin for Unreal: Integrate your game code](#). When you've successfully completed this workflow, your containerized game server is running in the cloud, and you can use the plugin to start a game client, connect to a game session, and play the game.

Before you start

This workflow assumes that you've completed the following tasks.

- **Integrate your game server code with Amazon GameLift Servers server SDK.** Your hosted game server must be able to communicate with the Amazon GameLift Servers service so that it can respond to requests to start new game sessions and report game session status. If you haven't completed this task, we recommend that you follow the plugin workflow [Host with Anywhere](#) first. For guidance on preparing your game server code, see [Update your game server code](#). For a managed container fleet, you must integrate your game with server SDK version 5.2 or higher.

Note

If you're using the startup game map, this task is already done for you.

- **Package your game server executable to run on Linux.** If you're developing on Windows and integrating C++ server SDK version 5.2.x or earlier, you'll need to work with the [Unreal cross compile toolkit](#). Alternatively, you might set up a separate Linux workspace or use a tool such as Windows subsystem for Linux (WSL).
- **Gather files to deploy with your game server build.** On your local machine, create a working directory to organize the files, which will be built into your game server container image. These might include game dependencies, a script to launch game servers and other processes when starting a container, etc.

- **Add the OpenSSL library files for your game server build.** You can skip this step if your game server is integrated with server SDK 5.3 or later.

Manually locate and copy the OpenSSL libraries to your game build package directory at <YourGame>/Binaries/Win64. You **must** use the same OpenSSL version that your Unreal Engine 5 version uses. Game builds that are deployed with the wrong OpenSSL libraries won't be able to communicate with the Amazon GameLift Servers service.

To find the OpenSSL libraries, look in your game engine source. The location varies depending on your development environment:

On Windows:

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

On Linux:

- Engine/Source/Thirdparty/OpenSSL/1.1.1n/include/libssl.so.1.1
- Engine/Source/Thirdparty/OpenSSL/1.1.1n/include/libcrypto.so.1.1
- **Integrate your game client code with Amazon GameLift Servers.** One way to complete this task is to add a sample asset (included with the plugin) that's already integrated. For guidance on preparing your game client code, see [Integrate your client game map](#).
- **Install Docker on your local machine.** You need this tool installed if you want the plugin to create container images for you and push them to an ECR repository. Alternatively you can do these tasks manually and instruct the plugin to use an existing container image. For more information about building your image manually, see [Build a container image for Amazon GameLift Servers](#).

To start the Amazon GameLift Servers managed containers workflow:

- In the Unreal editor main toolbar, choose the Amazon GameLift Servers menu, and select **Managed Containers**. This action opens the plugin page **Host with Managed Containers**, which presents a step-by-step process to create a container image with your game server build, deploy it to a container fleet, and launch your game.

Step 0: Set your profile

This section displays your currently selected user profile. Verify that the current user profile is the one you want to use for this workflow. All the resources that you create in this workflow are associated with the profile's AWS account and are placed in the profile's default AWS Region. The profile user's permissions determine your access to AWS resources and actions.

You might need to modify the selected user profile if:

- No profile is currently selected.
- You want to select a different profile or create a new profile.
- You need to bootstrap the selected profile (if bootstrap status is inactive).

To set or change the selected user profile

- In the Amazon GameLift Servers menu, choose **Open AWS Access Credentials**.

Step 1: Assess container readiness

Before deploying your game server to a container fleet, you must package it into a container image and store in an Amazon ECR repository. The plugin can complete these tasks for you or you can do these tasks manually. In this step, provide information about the status of your container image and the ECR repository.

Use the assessment questions to tell the plugin what steps it needs to take:

- **Create a new container image.** If you choose this option, the next step will prompt you for the location of your game server build directory and the build executable. The plugin uses a Dockerfile template (supplied by Amazon GameLift Servers) and automatically configures it for your game. You can view the template at [Build a container image for Amazon GameLift Servers](#). After choosing this option, indicate where you want the plugin to store the new image:
 - Create a new Amazon ECR repository and push the container image to it. The plugin creates a private ECR repo using the AWS account and default AWS Region in your selected user profile.
 - Push the container image to a previously created Amazon ECR repository. If you choose this option, the next step will prompt you to select an existing Amazon ECR repository from a list. The list includes all Amazon ECR repositories for the AWS account and default AWS Region in your selected user profile. You can select a public or private repository.

- **Use an existing container image.** If you've manually built an image, we recommend that you use the Dockerfile template supplied by Amazon GameLift Servers, which is available at [Build a container image for Amazon GameLift Servers](#). After choosing this option, indicate where the image is located.
 - A locally stored Docker-generated image. If you choose this option, the plugin creates a new Amazon ECR private repository and pushes the local image file to it. The next step will prompt you for an image ID, which the plugin uses to locate the image file.
 - A container image that's already stored in an Amazon ECR repository. If you choose this option, the next step will prompt you to select an existing Amazon ECR repository and image from a list. The list includes all Amazon ECR repositories for the AWS account and default AWS Region in your selected user profile. You can select a public or private repository.

Step 2: Configure image deployment

In this step, provide information that the plugin needs to deploy your container image to a container fleet. This step requests the following information:

- The location of your game server build, container image, or Amazon ECR repository, based on your selections in Step 1.
- The scenario to use for your managed containers deployment.
- The client configuration output path. Select the folder in your client build that contains your AWS configuration. Look for it in the following location: [client-build]/[project-name]/Content/CloudFormation.
- Optional deployment settings. This section has configuration settings that the plugin uses by default. You can modify these or keep the default values
 - Game name is set to the name of your game project by default. All AWS resources that the plugin creates references the game name value.
 - Port range, memory limit, and vCPU limit are configuration settings for the container fleet. For more information about customizing these values, see [Configure network connections](#) for connection port range, and [Set resource limits](#) for resource limits.
 - Container image tag is used to categorize your container images in Amazon ECR. The default value is unreal-gamelift-plugin.
 - Name of the Amazon ECR repository. You can edit this field to suggest a custom name only when the plugin is creating an ECR repository for you. The default value is unreal-game-lift-plugin-ecr-repository.

Deployment scenario options

Single-region container fleet

This scenario deploys your game server to a single container fleet. It's a good starting point for testing your server integration with AWS and your container configuration. It deploys the following resources.

- Amazon GameLift Servers container group definition describes how to deploy and run your container images on a container fleet.
- Amazon GameLift Servers container fleet (On-Demand) with your game server container installed and running, with alias.
- Amazon Cognito user pool and client to enable players to authenticate and start a game.
- API Gateway authorizer that links the user pool with APIs.
- Web access control list (ACL) for throttling excessive player calls to API Gateway.
- Backend service to make requests to the Amazon GameLift Servers service on behalf of game clients, such as to request game sessions and join games:
 - API Gateway + Lambda function for players to request a game session slot. This function calls `CreateGameSession()` if no open slots are available.
 - API Gateway + Lambda function for players to get connection info for their game request.

Single-region container fleet with FlexMatch

This scenario deploys your game server to a container fleet, configures game session placement, and sets up FlexMatch matchmaking. This scenario is useful when you're ready to start designing a custom matchmaker for your hosting solution. Use this scenario to create the basic resources for this solution, which you can customize later as needed. It deploys the following resources:

- Amazon GameLift Servers container group definition that describes how to deploy and run your container images on a container fleet.
- Amazon GameLift Servers container fleet (On-Demand) with your game server container installed and running, with alias.
- FlexMatch matchmaking configuration and matchmaking rule set to accept player requests and form matches.

- Amazon GameLift Servers game session queue that fulfills requests for proposed matches by finding the best possible hosting resource (based on viability, cost, player latency, etc.) and starting a game session.
- Amazon Cognito user pool and client to enable players to authenticate and start a game.
- API Gateway authorizer that links the user pool with APIs.
- Web access control list (ACL) for throttling excessive player calls to API Gateway.
- Backend service to make requests to the Amazon GameLift Servers service on behalf of game clients, such as to request game sessions and join games:
 - API Gateway + Lambda function for players to request a game session slot. This function calls `StartMatchmaking()` if no open slots are available.
 - API Gateway + Lambda function for players to get connection info for their game request.
- DynamoDB tables to store matchmaking tickets for players and game session information.
- Amazon SNS topic + Lambda function to handle `GameSessionQueue` events.

Deploy container fleet

When your fleet configuration is complete, choose the **Deploy container fleet** button to start deployment. This process can take several minutes while the plugin creates a container image and pushes it to ECR, provisions hosting resources for the container fleet, deploys the fleet and other AWS resources for the selected hosting solution scenario.

When you start a deployment, you can track the progress of each step. Depending on your configuration, the steps might include the following:

- Configuring container image
- Creating an Amazon ECR repository
- Building an image and pushing to Amazon ECR
- Creating container group definition
- Creating container fleet

For more detailed deployment information, choose **View in AWS Management Console**. When the container fleet reaches active status, the fleet is actively running containers with server processes that are ready to host game sessions.

When deployment is complete, you have a working container fleet that's ready to host game sessions and accept player connections.

You can't stop a deployment in progress. If the deployment enters a bad state or fails, you can start over by using the **Reset deployment** option.

Launch client

At this point, you've completed all the tasks to launch and play your multiplayer game hosted with Amazon GameLift Servers. To play your game, choose **Start Client** to launch a local instance of your game client.

- If you deployed the single fleet scenario, open one instance of your game client with one player and enter the server map to move around. You can open a second instance of the game client to add a second player to the same server game map.
- If you deployed the FlexMatch scenario, the hosting solution waits for at least two game clients to make matchmaking requests. Open at least two instances of your game client with one player. The two players will be matched and prompted to join a game session for the match.

Update a container fleet

If you've successfully deployed a managed containers hosting solution, you can use the **Update deployment** feature. This option lets you update configuration settings for a deployed container fleet, without having to create a new fleet.

When updating a deployment, you can deploy a container image with a different game server build, change the Amazon ECR repository, choose a different deployment scenario, and customize optional configuration settings.

When you're ready to deploy your changes, choose Update. The time required for a deployment update is similar to a full deployment. For detailed deployment information, choose **View in AWS Management Console**.

Clean up deployed resources

As a best practice, clean up the AWS resources for your managed containers solution as soon as you no longer need them. You might continue to incur costs for these resources if you don't remove them.

Delete the following resources:

- **Managed container resource stack.** The resources in this stack depends on the deployment scenario you selected. To delete the entire stack, use the CloudFormation console. Stacks that are generated from the Amazon GameLift Servers plugin use the following naming convention: `GameLiftPluginForUnreal-{GameName}-Containers`. Wait for the stack deletion process to complete before you initiate a new managed containers deployment in the plugin. For more information, see [Delete a stack from the CloudFormation console](#).
- **Amazon ECR repository.** If you used the plugin to create a repository for your container image, you might want to delete any repositories that are no longer needed. You don't need to delete a repository before resetting a managed containers deployment. If you update or reset a deployment, the plugin will automatically use the same repository unless directed to use another one. For more information, see [Deleting a private repository in Amazon ECR](#).

Amazon GameLift Servers plugin for Unity (server SDK 5.x)

This plugin adds the Amazon GameLift Servers C# server SDK and tools to the Unity editor. Use the guided UI workflows to integrate server SDK functionality into your game project and deploy an Amazon GameLift Servers hosting solution for your game server.

With the plugin, you can build a basic working hosting solution and then optimize and customize as needed. Set up an Amazon GameLift Servers Anywhere fleet with your local workstation as a host. For cloud hosting with managed EC2 or managed container fleets, deploy your game server with a complete solution to manage game session requests and client connections.

Topics

- [Install the plugin for your Unity game project](#)
- [Plugin for Unity: Set up an AWS user profile](#)
- [Plugin for Unity: Set up local testing with Amazon GameLift Servers Anywhere](#)
- [Plugin for Unity: Deploy your game to a managed EC2 fleet](#)
- [Plugin for Unity: Deploy your game to a managed container fleet](#)

Install the plugin for your Unity game project

[Get the Amazon GameLift Servers plugin for Unity from GitHub](#)

See the GitHub repository readme for information about how to install the plugin for a game project.

The plugin includes these components:

- Plugin modules for the Unity editor. When the plugin is installed, a new main menu item gives you access to Amazon GameLift Servers functionality.
- C# libraries for the Amazon GameLift Servers service API with client-side functionality.
- C# libraries for the Amazon GameLift Servers server SDK (version 5.x).
- Sample game content, including assets and scenes, so you can try out Amazon GameLift Servers even if you don't have a build-ready multiplayer game.
- Solution configurations, provided as CloudFormation templates, that the plugin uses when deploying your game server to the cloud for hosting.

This plugin uses AWS CloudFormation templates to deploy hosting solutions for common gaming scenarios. Use these solutions as provided or customize them as needed for your games.

Plugin for Unity: Set up an AWS user profile

After installing the plugin, set up a user profile with a valid AWS account. You can maintain multiple profiles in the plugin, but you can have only one profile selected at a time. Whenever you work in the plugin, select a profile to use. Each workflow page displays the currently selected profile.

Maintaining multiple profiles gives you the ability to switch between different hosting deployments. For example, you might set up profiles that use the same AWS account but deploy to different AWS Regions. Or you might set up profiles with different AWS accounts or users and permission sets.

Note

If you've installed the AWS CLI on your workstation and have a profile already configured, the Amazon GameLift Servers plugin will detect it and list it as an existing profile. The plugin automatically selects any profile named `[default]`. You can use an existing profile or create a new one.

All profiles must be bootstrapped to set up some required AWS resources under your account user.

To manage your AWS profiles

1. In the Unity main toolbar, choose the Amazon GameLift Servers menu, and select **AWS Access Credentials**. This action opens the Amazon GameLift Servers plugin to the page **Set up your user profile**.
2. Use the buttons to create a new AWS account or set up a user profile for an AWS account that you already have.
3. If you don't already have a user profile, you're prompted to enter profile details and create a new profile. Provide the following information:
 - An AWS account. If you create a new AWS account, use the link to the AWS Management Console and follow the prompts. See [Create an AWS account](#) for more details.
 - An AWS user with permissions to use Amazon GameLift Servers and other required AWS services. See [Set up an AWS user account](#) for instructions on setting up an AWS Identity and Access Management (IAM) user with Amazon GameLift Servers permissions and programmatic access with long-term credentials.

Note

The "Set up an IAM service role" step is optional for plugin users. You only need an IAM service role if your game server accesses other AWS resources, such as Amazon S3 or DynamoDB.

- Credentials for your AWS user. These credentials consist of an AWS access key ID and AWS secret key. See [Get your access keys](#) for more details.
 - AWS region. This is a geographic location where you want to create your AWS resources for hosting. During development, we recommend using a region close to your physical location. Choose a region from the list of [supported AWS regions](#).
4. If the plugin detects existing profiles, it displays a list of available profiles. Select an existing profile from the list, or choose **Add another profile** to create a new one.

Bootstrap a user profile

All profiles must be bootstrapped to use with the Amazon GameLift Servers plugin. Bootstrapping creates an Amazon S3 bucket specific to the profile. It's used to store project configurations, build artifacts, and other dependencies. Buckets are not shared between other profiles.

Bootstrapping involves creating new AWS resources and might incur costs.

To bootstrap your profile:

1. On the **AWS Access Credentials** page, check the bootstrap status of the user profile that you want to use. If the profile's bootstrap status is "Inactive" and there's no S3 bucket listed, you need to bootstrap the profile.
2. Select the profile you want to use and choose **Bootstrap profile**.
3. Wait for bootstrap status to change to "Active". This can take a few minutes.

Plugin for Unity: Set up local testing with Amazon GameLift Servers Anywhere

In this workflow, you add client and server game code for Amazon GameLift Servers functionality and use the plugin to designate your local workstation as a test game server host. When you've completed integration tasks, use the plugin to build your game client and server components.

To start the Amazon GameLift Servers Anywhere workflow:

- In the Unity editor main menu, choose **Amazon GameLift Servers** and select **Host with Anywhere**. This action opens the plugin page for setting up your game with an @Anywhere fleet. The page presents a five-step process to integrate, build, and launch your game components.

Set your profile

Choose the profile you want to use when following this workflow. The profile you select impacts all steps in the workflow. All resources you create are associated with the profile's AWS account and are placed in the profile's default AWS Region. The profile user's permissions determine your access to AWS resources and actions.

1. Select a profile from the dropdown list of available profiles. If you don't have a profile yet or want to create a new one, go to the **Amazon GameLift Servers** menu and choose **Set AWS Account Profiles**.
2. If bootstrap status is not "Active", choose **Bootstrap profile** and wait for the status to change to "Active".

Integrate your game code with the C# server SDK

Note

If you imported the sample game, you can skip this step. The sample game assets already have the necessary server and client code in place.

For this step in the workflow, you make updates to the client and server code in your game project.

- Game servers must be able to communicate with the Amazon GameLift Servers service to receive prompts to start a game session, provide game session connection information, and report status.
- Game clients must be able to get information about game sessions, join or start game sessions, and get connection information to join a game.

Integrate your server code

If you're using your own game project with custom scenes, use provided sample code to add required server code to your game project:

1. In your game project files, open the `Assets/Scripts/Server` folder. If it doesn't exist, create it.
2. Go to the GitHub repo [aws/amazon-gamelift-plugin-unity](https://github.com/aws/amazon-gamelift-plugin-unity) and open the path `Samples~/SampleGame/Assets/Scripts/Server`.
3. Locate the file `GameLiftServer.cs` and copy it into your game project's `Server` folder. When you build a server executable, use this file as the build target.

The sample code includes these minimum required elements, which use Amazon GameLift Servers C# server SDK (version 5):

- Initializes an Amazon GameLift Servers API client. The `InitSDK()` call with server parameters is required for an Amazon GameLift Servers Anywhere fleet. These settings are automatically set for use in the plugin.
- Implements required callback functions to respond to requests from the Amazon GameLift Servers service, including `OnStartGameSession`, `OnProcessTerminate`, and `onHealthCheck`.

- Calls `ProcessReady()` with a designated port to notify the Amazon GameLift Servers service when the server process is ready to host game sessions.

If you want to customize the sample server code, see these resources:

- [Add Amazon GameLift Servers to your game server with the server SDK](#)
- [C# server SDK 5.x for Amazon GameLift Servers -- Actions](#)

Integrate your client code

If you're using your own game project with custom scenes, then you need to integrate basic functionality into your game client. You also need to add UI elements so that players can sign in and join a game session. Use the service API for Amazon GameLift Servers (in the AWS SDK) to get game session information, create new game sessions, or join existing game sessions,

When building a client for local testing with an Anywhere fleet, you can add direct calls to the Amazon GameLift Servers service. When you develop your game for cloud hosting—or if you plan to use Anywhere fleets for production hosting—you'll need to create a client-side backend service to handle all communication between game clients and the Amazon GameLift Servers service.

To integrate Amazon GameLift Servers into your client code, use the following resources as a guide.

- Integrate the client with the `GameLiftCoreApi` class in the GitHub repo `aws/amazon-gamelift-plugin-unity`. This class provides controls for player authentication and for retrieving game session information.
- View sample game integrations, available in the GitHub repo `aws/amazon-gamelift-plugin-unity`, `Samples~/SampleGame/Assets/Scripts/Client/GameLiftClient.cs`.
- Follow instructions in `Add Amazon GameLift Servers to your Unity game client`.

For game clients connecting to an Anywhere fleet, your game client needs the following information. The plugin automatically updates your game project to use the resources that you create in the plugin.

- `FleetId` - The unique identifier for your Anywhere fleet.
- `FleetLocation` - The custom location of your Anywhere fleet.
- `AwsRegion` - The AWS region where your Anywhere fleet is hosted. This is the region you set in your user profile.

- **ProfileName** - An AWS credentials profile on your local machine that allows access to the AWS SDK for Amazon GameLift Servers . The game client uses these credentials to authenticate requests to the Amazon GameLift Servers service.

Note

The credentials profile is generated by the plugin and stored on the local machine. As a result, you must run the client on the local machine (or on a machine with the same profile).

Connect to an Anywhere fleet

In this step, you designate an Anywhere fleet to use. An Anywhere fleet defines a collection of compute resources, which can be located anywhere, for game server hosting.

- If the AWS account you're currently using has existing Anywhere fleets, open the **Fleet name** dropdown field and choose a fleet. This dropdown only shows the Anywhere fleets in the AWS Region for the currently active user profile.
- If there are no existing fleets—or you want to create a new one, choose **Create new Anywhere fleet** and provide a fleet name.

After you've chosen an Anywhere fleet for your project, Amazon GameLift Servers verifies that fleet status is active and displays the fleet ID. You can track progress of this request in the Unity editor's output log.

Register a compute

In this step, you register your local workstation as a compute resource in the new Anywhere fleet.

1. Enter a compute name for your local machine. If you add more than one compute in the fleet, the names must be unique.
2. Choose **Register compute**. You can track progress of this request in the Unity editor's output log.

The plugin registers your local workstation with the IP address set to localhost (127.0.0.1). This setting assumes that you'll run your game client and server on the same machine.

In response to this action, Amazon GameLift Servers verifies that it can connect to the compute and returns information about the newly registered compute.

Launch game

In this step you build your game components and launch them to play the game. Complete the following tasks:

1. Configure your game client. In this step, you prompt the plugin to update a `GameLiftClientSettings` asset for your game project. The plugin uses this asset to store certain information that your game client needs to connect to the Amazon GameLift Servers service.
 - a. If you didn't import and initialize the sample game, create a new `GameLiftClientSettings` asset. In the Unity editor main menu, choose **Assets, Create, Amazon GameLift, Client Settings**. If you create multiple copies of `GameLiftClientSettings` in your project, the plugin automatically detects this and notifies you which asset the plugin will update.
 - b. In **Launch Game**, choose **Configure Client: Apply Anywhere Settings**. This action updates your game client settings to use the Anywhere fleet that you just set up.
2. Build and run your game client.
 - a. Build a client executable using the standard Unity build process. In **File, Build Settings**, switch the platform to **Windows, Mac, Linux**. If you imported the sample game and initialized the settings, the build list and build target are automatically updated.
 - b. Launch one or more instances of the newly built game client executable.
3. Launch a game server in your Anywhere fleet. Choose **Server: Launch Server** in Editor. This task starts a live server that your client can connect to as long as the Unity editor remains open.
4. Start or join a game session. In your game client instances, use the UI to join each client to a game session. How you do this depends on how you added functionality to the client.

If you're using the sample game client, it has the following characteristics:

- A player login component. When connecting to a game server on an Anywhere fleet, there is no player validation. You can enter any values to join the game session.

- A simple join game UI. When a client attempts to join a game, the client automatically looks for an active game session with an available player slot. If no game session is available, the client requests a new game session. If a game session is available, the client requests to join the available game session. When testing your game with multiple concurrent clients, the first client starts the game session, and the remaining clients automatically join the existing game session.
- Game sessions with four player slots. You can launch up to four game client instances concurrently and they will join the same game session.

Launch from a server executable (optional)

You can build and launch your game server executable for testing on an Anywhere fleet.

1. Build a server executable using the standard Unity build process. In **File, Build Settings**, switch the platform to **Dedicated Server** and build.
2. Get a short-term authentication token by calling the AWS CLI command [get-compute-auth-token](#) with your Anywhere fleet ID and AWS Region. The fleet ID is displayed in **Connect to an Anywhere Fleet** when you create the fleet. The AWS Region is displayed in **Set Your Profile** when you select your active profile.

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

3. Launch the newly built game server executable from a command line and pass in a valid auth token.

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

Plugin for Unity: Deploy your game to a managed EC2 fleet

In this workflow, you use the plugin to prepare your game for hosting on cloud-based compute resources that are managed by Amazon GameLift Servers. You add client and server game code for Amazon GameLift Servers functionality, then upload your server build to the Amazon GameLift Servers service for hosting. When this workflow is complete, you'll have game servers running in the cloud and a working game client that can connect to them.

To start the Amazon GameLift Servers managed Amazon EC2 workflow:

- In the Unity editor main menu, choose **Amazon GameLift Servers** and select **Host with Managed EC2**. This workflow presents a six-step process to integrate, build, deploy, and launch your game components.

Set your profile

Choose the profile you want to use when following this workflow. The profile you select impacts all steps in the workflow. All resources you create are associated with the profile's AWS account and are placed in the profile's default AWS Region. The profile user's permissions determine your access to AWS resources and actions.

1. Select a profile from the dropdown list of available profiles. If you don't have a profile yet or want to create a new one, go to the Amazon GameLift Servers menu and choose **Set AWS Account Profiles**.
2. If bootstrap status is not "Active", choose **Bootstrap profile** and wait for the status to change to "Active".

Integrate your game with Amazon GameLift Servers

For this task, you make updates to the client and server code in your game project.

- Game servers must be able to communicate with the Amazon GameLift Servers service to receive prompts to start a game session, provide game session connection information, and report status.
- Game clients must be able to get information about game sessions, join or start game sessions, and get connection information to join a game.

Note

If you imported the sample game, you can skip this step. The sample game assets already have the necessary server and client code in place.

Integrate your server code

When using your own game project with custom scenes, use the provided sample code to add required server code to your game project. If you integrated your game project for testing with an Anywhere fleet, you've already completed the instructions in this step.

1. In your game project files, open the `Assets/Scripts/Server` folder. If it doesn't exist, create it.
2. Go to the GitHub repo [aws/amazon-gamelift-plugin-unity](https://github.com/aws/amazon-gamelift-plugin-unity) and open the path `Samples~/SampleGame/Assets/Scripts/Server`.
3. Locate the file `GameLiftServer.cs` and copy it into your game project's `Server` folder. When you build a server executable, use this file as the build target.

The sample code includes these minimum required elements, which use Amazon GameLift Servers C# server SDK (version 5):

- Initializes an Amazon GameLift Servers API client. The `InitSDK()` call with server parameters is required for an Amazon GameLift Servers Anywhere fleet. These settings are automatically set for use in the plugin.
- Implements required callback functions to respond to requests from the Amazon GameLift Servers service, including `OnStartGameSession`, `OnProcessTerminate`, and `onHealthCheck`.
- Calls `ProcessReady()` with a designated port to notify the Amazon GameLift Servers service when the server process is ready to host game sessions.

If you want to customize the sample server code, see these resources:

- [Add Amazon GameLift Servers to your game server with the server SDK](#)
- [C# server SDK 5.x for Amazon GameLift Servers -- Actions](#)

Integrate your client code

For game clients that connect to cloud-based game servers, it's a best practice to use a client-side backend service to make calls to the Amazon GameLift Servers service, instead of making the calls directly from the game client.

In the plugin workflow for hosting on a managed EC2 fleet, each deployment scenario includes a pre-built backend service that includes the following components:

- A set of Lambda functions and DynamoDB tables that are used to request game sessions and retrieve game session information. These components use an API gateway as the proxy.
- An Amazon Cognito user pool that generates unique player IDs and authenticates player connections.

To use these components, your game client needs functionality to send requests to the backend service to do the following:

- Create a player user in the AWS Cognito user pool and authenticate.
- Join a game session and receive connection information.
- Join a game using matchmaking.

Use the following resources as a guide.

- Integrate the client with the [GameLiftCoreApi](#) class in the GitHub repo [aws/amazon-gamelift-plugin-unity](#). This class provides controls for player authentication and for retrieving game session information.
- To view the sample game integrations go to the GitHub repo [aws/amazon-gamelift-plugin-unity](#), `Samples~/SampleGame/Assets/Scripts/Client/GameLiftClient.cs`.
- [Integrate Amazon GameLift Servers game client functionality](#).

(Unity 6.3+ Only) Integrate install.sh script

Unity 6.3 on Linux requires glibc 2.35. Amazon Linux 2023 includes glibc 2.34. An `install.sh` script is provided which handles building glibc 2.35 from source and configuring the Unity binaries to use glibc 2.35.

1. Go to the GitHub repo [aws/amazon-gamelift-plugin-unity](#) and open the path `ExtraServerResources`.
2. Locate the file `install.sh` and copy it into the folder containing your server build.
3. If you have an existing `install.sh`, merge the contents of both scripts into a single `install.sh`.

Select deployment scenario

In this step, you choose the game hosting solution that you want to deploy at this time. You can have multiple deployments of your game, using any of the scenarios.

- **Single-region fleet:** Deploys your game server to a single fleet of hosting resources in the active profile's default AWS region. This scenario is a good starting point for testing your server integration with AWS and server build configuration. It deploys the following resources:
 - AWS fleet (On-Demand) with your game server build installed and running.
 - Amazon Cognito user pool and client to enable players to authenticate and start a game.
 - API gateway authorizer that links user pool with APIs.
 - WebACL for throttling excessive player calls to API gateway.
 - API gateway + Lambda function for players to request a game slot. This function calls `CreateGameSession()` if none are available.
 - API gateway + Lambda function for players to get connection info for their game request.
- **FlexMatch fleet:** Deploys your game server to a set of fleets and sets up a FlexMatch matchmaker with rules to create player matches. This scenario uses low-cost Spot hosting with a multi-fleet, multi-location structure for durable availability. This approach is useful when you're ready to start designing a matchmaker component for your hosting solution. In this scenario, you'll create the basic resources for this solution, which you can customize later as needed. It deploys the following resources:
 - FlexMatch matchmaking configuration and matchmaking rule set to accept player requests and form matches.
 - Three AWS fleets with your game server build installed and running in multiple locations. Includes two Spot fleets and one On-Demand fleet as a backup.
 - AWS game session placement queue that fulfills requests for proposed matches by finding the best possible hosting resource (based on viability, cost, player latency, etc.) and starting a game session.
 - Amazon Cognito user pool and client to enable players to authenticate and start a game.
 - API gateway authorizer that links user pool with APIs.
 - WebACL for throttling excessive player calls to API gateway.
 - API gateway + Lambda function for players to request a game slot. This function calls `StartMatchmaking()`.
 - ~~API gateway + Lambda function for players to get connection info for their game request.~~

- Amazon DynamoDB tables to store matchmaking tickets for players and game session information.
- SNS topic + Lambda function to handle GameSessionQueue events.

Set game parameters

In this step, you describe your game for uploading to AWS.

- **Game name:** Provide a meaningful name for your game project. This name is used within the plugin.
- **Fleet name:** Provide a meaningful name for your managed EC2 fleet. Amazon GameLift Servers uses this name (along with the fleet ID) when listing resources in the AWS console.
- **Build name:** Provide a meaningful name for your server build. AWS uses this name to refer to the copy of your server build that's uploaded to Amazon GameLift Servers and used for deployments.
- **Launch parameters:** Enter optional instructions to run when launching the server executable on a managed EC2 fleet instance. Maximum length is 1024 characters.
- **Game server folder:** Provide the path to a local folder containing your server build.
- **Game server file:** Specify the server executable file name.

Deploy scenario

In this step, you deploy your game to a cloud hosting solution based on the deployment scenario you chose. This process can take several minutes while AWS validates your server build, provisions hosting resources, installs your game server, launches server processes, and gets them ready to host game sessions.

To start deployment, choose **Deploy CloudFormation**. You can track the status of your game hosting here. For more detailed information, you can sign in to the AWS Management console for AWS and view event notifications. Be sure to sign in using the same account, user, and AWS Region as the active user profile in the plugin.

When deployment is complete, you have your game server installed on an AWS EC2 instance. At least one server process is running and ready to start a game session.

Launch game client

When your fleet is successfully deployed, you now have game servers running and available to host game sessions. You can now build your client, launch it, connect to join the game session.

1. Configure your game client. In this step, you prompt the plugin to update a `GameLiftClientSettings` asset for your game project. The plugin uses this asset to store certain information that your game client needs to connect to the Amazon GameLift Servers service.
 - a. If you didn't import and initialize the sample game, create a new `GameLiftClientSettings` asset. In the Unity editor main menu, choose **Assets, Create, Amazon GameLift, Client Settings**. If you create multiple copies of `GameLiftClientSettings` in your project, the plugin automatically detects this and notifies you which asset the plugin will update.
 - b. In **Launch Game**, choose **Configure Client: Apply Managed EC2 Settings**. This action updates your game client settings to use the managed EC2 fleet that you just deployed.
2. Build your game client. Build a client executable using the standard Unity build process. In File, Build Settings, switch the platform to Windows, Mac, Linux. If you imported the sample game and initialized the settings, the build list and build target are automatically updated.
3. Launch the newly build game client executable. To start playing the game, start two to four client instances and use the UI in each to join a game session.

If you're using the sample game client, it has the following characteristics:

- A player login component. When connecting to a game server on an Anywhere fleet, there is no player validation. You can enter any values to join the game session.
- A simple join game UI. When a client attempts to join a game, the client automatically looks for an active game session with an available player slot. If no game session is available, the client requests a new game session. If a game session is available, the client requests to join the available game session. When testing your game with multiple concurrent clients, the first client starts the game session, and the remaining clients automatically join the existing game session.
- Game sessions with four player slots. You can launch up to four game client instances concurrently and they will join the same game session.

Plugin for Unity: Deploy your game to a managed container fleet

Use this guided plugin workflow to create a container image for your game server and deploy it to a container-based hosting solution. When you've successfully completed this workflow, your containerized game server is running in the cloud, and you can use the plugin to start a game client, connect to a game session, and play the game.

Before you start

This workflow assumes that you've completed the following tasks.

- **Integrate your game server code with Amazon GameLift Servers server SDK.** our hosted game server must be able to communicate with the Amazon GameLift Servers service so that it can respond to requests to start new game sessions and report game session status. If you haven't completed this task, we recommend that you follow the plugin workflow [Host with Anywhere](#) first. For guidance on preparing your game server code, see [Integrate your server code](#). For a managed container fleet, you must integrate your game with server SDK version 5.2 or higher.

Note

If you imported the sample game, this task is already done for you.

- **Package your game server executable to run on Linux.**
- **Gather files to deploy with your game server build.** On your local machine, create a working directory to organize the files, which will be built into your game server container image. These might include game dependencies, a script to launch game servers and other processes when starting a container, etc.
- **Integrate your game client code with Amazon GameLift Servers.** One way to complete this task is to add a sample asset (included with the plugin) that's already integrated. For guidance on preparing your game client code, see [Integrate your client code](#).
- **Install Docker on your local machine.** You need this tool installed if you want the plugin to create container images for you and push them to an ECR repository. Alternatively you can do these tasks manually and instruct the plugin to use an existing container image. For more information about building your image manually, see [Build a container image for Amazon GameLift Servers](#).

To start the Amazon GameLift Servers managed containers workflow:

- In the Unity editor main toolbar, choose the Amazon GameLift Servers menu, and select **Managed Containers**. This action opens the plugin page **Host with Managed Containers**, which presents a step-by-step process to create a container image with your game server build, deploy it to a container fleet, and launch your game.

Step 0: Set your profile

This section displays your currently selected user profile. Verify that the current user profile is the one you want to use for this workflow. All the resources that you create in this workflow are associated with the profile's AWS account and are placed in the profile's default AWS Region. The profile user's permissions determine your access to AWS resources and actions.

You might need to modify the selected user profile if:

- No profile is currently selected.
- You want to select a different profile or create a new profile.
- You need to bootstrap the selected profile (if bootstrap status is inactive).

To set or change the selected user profile

- In the Amazon GameLift Servers menu, choose **Open AWS Access Credentials**.

Step 1: Assess container readiness

Before deploying your game server to a container fleet, you must package it into a container image and store in an Amazon ECR repository. The plugin can complete these tasks for you or you can do these tasks manually. In this step, provide information about the status of your container image and the ECR repository.

Use the assessment questions to tell the plugin what steps it needs to take:

- **Create a new container image.** If you choose this option, the next step will prompt you for the location of your game server build directory and the build executable. The plugin uses a Dockerfile template (supplied by Amazon GameLift Servers) and automatically configures it for your game. You can view the template at [Build a container image for Amazon GameLift Servers](#). After choosing this option, indicate where you want the plugin to store the new image:

- Create a new Amazon ECR repository and push the container image to it. The plugin creates a private ECR repo using the AWS account and default AWS Region in your selected user profile.
- Push the container image to a previously created Amazon ECR repository. If you choose this option, the next step will prompt you to select an existing Amazon ECR repository from a list. The list includes all Amazon ECR repositories for the AWS account and default AWS Region in your selected user profile. You can select a public or private repository.
- **Use an existing container image.** If you've manually built an image, we recommend that you use the Dockerfile template supplied by Amazon GameLift Servers, which is available at [Build a container image for Amazon GameLift Servers](#). After choosing this option, indicate where the image is located.
 - A locally stored Docker-generated image. If you choose this option, the plugin creates a new Amazon ECR private repository and pushes the local image file to it. The next step will prompt you for an image ID, which the plugin uses to locate the image file.
 - A container image that's already stored in an Amazon ECR repository. If you choose this option, the next step will prompt you to select an existing Amazon ECR repository and image from a list. The list includes all Amazon ECR repositories for the AWS account and default AWS Region in your selected user profile. You can select a public or private repository.
- Unity 6.3 on Linux requires glibc 2.35. Amazon Linux 2023 includes glibc 2.34. A Dockerfile template is provided below which handles building glibc 2.35 from source and configuring the Unity binaries to use glibc 2.35.

Dockerfile template for a Unity 6.3 game server container image

This template contains the minimum instructions that a game server container needs to usable in an Amazon GameLift Servers fleet. Modify the content as needed for your game server.

```
# Base image
# -----
# Add the base image that you want to use over here,
# Make sure to use an image with the same architecture as the
# Instance type you are planning to use on your fleets.
FROM public.ecr.aws/amazonlinux/amazonlinux

# Game build directory
# -----
# Add your game build directory in the 'GAME_BUILD_DIRECTORY' env variable below.
#
```

```

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

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

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

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

# Change directory to home
WORKDIR $HOME_DIR

# Build glibc 2.35 and patch Unity binaries
RUN GLIBC_VERSION="2.35" && \
    INSTALL_PREFIX="/opt/glibc-`${GLIBC_VERSION}`" && \
    ARCH=$(uname -m) && \
    if [ "$ARCH" = "x86_64" ]; then \
        INTERPRETER="`${INSTALL_PREFIX}`/lib/ld-linux-x86-64.so.2"; \
        MONO_ARCH="x86_64"; \

```

```

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

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

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

USER gamescale

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

# Check path before starting the container
RUN echo $PATH

# Start the game build

```

```
ENTRYPOINT ["/bin/sh", "-c", "./$GAME_EXECUTABLE $LAUNCH_PARAMS"]
```

Step 2: Configure image deployment

In this step, provide information that the plugin needs to deploy your container image to a container fleet. This step requests the following information:

- The location of your game server build, container image, or Amazon ECR repository, based on your selections in Step 1.
- The scenario to use for your managed containers deployment.
- Optional deployment settings. This section has configuration settings that the plugin uses by default. You can modify these or keep the default values
 - Game name is set to the name of your game project by default. All AWS resources that the plugin creates references the game name value.
 - Port range, memory limit, and vCPU limit are configuration settings for the container fleet. For more information about customizing these values, see [Configure network connections](#) for connection port range, and [Set resource limits](#) for resource limits.
 - Container image tag is used to categorize your container images in Amazon ECR. The default value is `unity-gamelift-plugin`.

Deployment scenario options

Single-region container fleet

This scenario deploys your game server to a single container fleet. It's a good starting point for testing your server integration with AWS and your container configuration. It deploys the following resources.

- Amazon GameLift Servers container group definition describes how to deploy and run your container images on a container fleet.
- Amazon GameLift Servers container fleet (On-Demand) with your game server container installed and running, with alias.
- Amazon Cognito user pool and client to enable players to authenticate and start a game.
- API Gateway authorizer that links the user pool with APIs.
- Web access control list (ACL) for throttling excessive player calls to API Gateway.

- Backend service to make requests to the Amazon GameLift Servers service on behalf of game clients, such as to request game sessions and join games:
 - API Gateway + Lambda function for players to request a game session slot. This function calls `CreateGameSession()` if no open slots are available.
 - API Gateway + Lambda function for players to get connection info for their game request.

Single-region container fleet with FlexMatch

This scenario deploys your game server to a container fleet, configures game session placement, and sets up FlexMatch matchmaking. This scenario is useful when you're ready to start designing a custom matchmaker for your hosting solution. Use this scenario to create the basic resources for this solution, which you can customize later as needed. It deploys the following resources:

- Amazon GameLift Servers container group definition that describes how to deploy and run your container images on a container fleet.
- Amazon GameLift Servers container fleet (On-Demand) with your game server container installed and running, with alias.
- FlexMatch matchmaking configuration and matchmaking rule set to accept player requests and form matches.
- Amazon GameLift Servers game session queue that fulfills requests for proposed matches by finding the best possible hosting resource (based on viability, cost, player latency, etc.) and starting a game session.
- Amazon Cognito user pool and client to enable players to authenticate and start a game.
- API Gateway authorizer that links the user pool with APIs.
- Web access control list (ACL) for throttling excessive player calls to API Gateway.
- Backend service to make requests to the Amazon GameLift Servers service on behalf of game clients, such as to request game sessions and join games:
 - API Gateway + Lambda function for players to request a game session slot. This function calls `StartMatchmaking()` if no open slots are available.
 - API Gateway + Lambda function for players to get connection info for their game request.
- DynamoDB tables to store matchmaking tickets for players and game session information.
- Amazon SNS topic + Lambda function to handle `GameSessionQueue` events.

Deploy container fleet

When your fleet configuration is complete, choose the **Deploy container fleet** button to start deployment. This process can take several minutes while the plugin creates a container image and pushes it to ECR, provisions hosting resources for the container fleet, deploys the fleet and other AWS resources for the selected hosting solution scenario.

When you start a deployment, you can track the progress of each step. Depending on your configuration, the steps might include the following:

- Configuring container image
- Creating an Amazon ECR repository
- Building an image and pushing to Amazon ECR
- Creating container group definition
- Creating container fleet

For more detailed deployment information, choose **View in AWS Management Console**. When the container fleet reaches active status, the fleet is actively running containers with server processes that are ready to host game sessions.

When deployment is complete, you have a working container fleet that's ready to host game sessions and accept player connections.

You can't stop a deployment in progress. If the deployment enters a bad state or fails, you can start over by using the **Reset deployment** option.

Launch client

At this point, you've completed all the tasks to launch and play your multiplayer game hosted with Amazon GameLift Servers. To play your game, choose **Start Client** to launch a local instance of your game client.

- If you deployed the single fleet scenario, open one instance of your game client with one player and enter the server map to move around. You can open a second instance of the game client to add a second player to the same server game map.
- If you deployed the FlexMatch scenario, the hosting solution waits for at least two game clients to make matchmaking requests. Open at least two instances of your game client with one player. The two players will be matched and prompted to join a game session for the match.

Update a container fleet

If you've successfully deployed a managed containers hosting solution, you can use the **Update deployment** feature. This option lets you update configuration settings for a deployed container fleet, without having to create a new fleet.

When updating a deployment, you can deploy a container image with a different game server build, change the Amazon ECR repository, choose a different deployment scenario, and customize optional configuration settings.

When you're ready to deploy your changes, choose Update. The time required for a deployment update is similar to a full deployment. For detailed deployment information, choose **View in AWS Management Console**.

Clean up deployed resources

As a best practice, clean up the AWS resources for your managed containers solution as soon as you no longer need them. You might continue to incur costs for these resources if you don't remove them.

Delete the following resources:

- Managed container resource stack. The resources in this stack depends on the deployment scenario you selected. To delete the entire stack, use the CloudFormation console. Stacks that are generated from the Amazon GameLift Servers plugin use the following naming convention: `GameLiftPluginForUnity-{GameName}-Containers`. Wait for the stack deletion process to complete before you initiate a new managed containers deployment in the plugin. For more information, see [Delete a stack from the CloudFormation console](#).
- Amazon ECR repository. If you used the plugin to create a repository for your container image, you might want to delete any repositories that are no longer needed. You don't need to delete a repository before resetting a managed containers deployment. If you update or reset a deployment, the plugin will automatically use the same repository unless directed to use another one. For more information, see [Deleting a private repository in Amazon ECR](#).

Amazon GameLift Servers plugin for Unity for server SDK 4

Note

This topic provides information for an earlier version of the Amazon GameLift Servers plugin for Unity. Version 1.x uses the server SDK for Amazon GameLift Servers 4.x or earlier. For documentation on the latest plugin version, which uses server SDK 5.x and supports newer features such as Amazon GameLift Servers Anywhere and managed container hosting, see [Amazon GameLift Servers plugin for Unity \(server SDK 5.x\)](#).

Amazon GameLift Servers provides tools for preparing your multiplayer game servers to run on Amazon GameLift Servers. The Amazon GameLift Servers plugin for Unity makes it easier to integrate Amazon GameLift Servers into your Unity game projects and deploy Amazon GameLift Servers resources for cloud hosting. Use the plugin for Unity to access Amazon GameLift Servers APIs and deploy AWS CloudFormation templates for common gaming scenarios.

After you've set up the plugin, you can try out the [Amazon GameLift Servers Unity sample](#) on GitHub.

Topics

- [Integrate Amazon GameLift Servers with a Unity game server project](#)
- [Integrate Amazon GameLift Servers with a Unity game client project](#)
- [Install and set up the plugin](#)
- [Test your game locally](#)
- [Deploy a scenario](#)
- [Integrate games with Amazon GameLift Servers in Unity](#)
- [Import and run a sample game](#)

Integrate Amazon GameLift Servers with a Unity game server project

Note

This topic provides information for an earlier version of the Amazon GameLift Servers plugin for Unity. Version 1.x uses the server SDK for Amazon GameLift Servers 4.x or earlier. For documentation on the latest plugin version, which uses server SDK 5.x

and supports newer features such as Amazon GameLift Servers Anywhere and managed container hosting, see [Amazon GameLift Servers plugin for Unity \(server SDK 5.x\)](#).

This topic helps you prepare your custom game server for hosting on Amazon GameLift Servers. The game server must be able to notify Amazon GameLift Servers about its status, to start and stop game sessions when prompted, and to perform other tasks. For more information, see [Add Amazon GameLift Servers to your game server with the server SDK](#).

Prerequisites

Before integrating your game server, complete the following tasks:

- [Set up an IAM service role for Amazon GameLift Servers](#)
- [Install and set up the plugin](#)

Set up a new server process

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

Set up communication with Amazon GameLift Servers and report that the server process is ready to host a game session.

1. Initialize the server SDK by calling `InitSDK()`.
2. To prepare the server to accept a game session, call `ProcessReady()` with the connection port and game session location details. Include the names of callback functions that Amazon GameLift Servers service invokes, such as `OnGameSession()`, `OnGameSessionUpdate()`, `OnProcessTerminate()`, `OnHealthCheck()`. Amazon GameLift Servers might take a few minutes to provide a callback.
3. Amazon GameLift Servers updates the status of the server process to `ACTIVE`.
4. Amazon GameLift Servers calls `onHealthCheck` periodically.

The following code example shows how to set up a simple server process with Amazon GameLift Servers.

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

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

var processReadyOutcome = GameLiftServerAPI.ProcessReady(processParams);

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

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

bool OnHealthCheck()
{
    bool isHealthy;
    // complete health evaluation within 60 seconds and set health
```

```
    return isHealthy;
}
```

Start a game session

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

After game initialization is complete, you can start a game session.

1. Implement the callback function `onStartGameSession`. Amazon GameLift Servers invokes this method to start a new game session on the server process and receive player connections.
2. To activate a game session, call `ActivateGameSession()`. For more information about the SDK, see [C# server SDK for Amazon GameLift Servers 4.x -- Actions](#).

The following code example illustrates how to start a game session with Amazon GameLift Servers.

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

End a game session

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

Notify Amazon GameLift Servers when a game session is ending. As a best practice, shut down server processes after game sessions complete to recycle and refresh hosting resources.

1. Set up a function named `onProcessTerminate` to receive requests from Amazon GameLift Servers and call `ProcessEnding()`.
2. The process status changes to `TERMINATED`.

The following example describes how to end a process for a game session.

```
var processEndingOutcome = GameLiftServerAPI.ProcessEnding();

if (processReadyOutcome.Success)
    Environment.Exit(0);

// otherwise, exit with error code
Environment.Exit(errorCode);
```

Create server build and upload to Amazon GameLift Servers

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

After you integrate your game server with Amazon GameLift Servers, upload the build files to a fleet so that Amazon GameLift Servers can deploy it for game hosting. For more information on how to upload your server to Amazon GameLift Servers, see [Create a game server build for Amazon GameLift Servers](#).

Integrate Amazon GameLift Servers with a Unity game client project

Note

This topic provides information for an earlier version of the Amazon GameLift Servers plugin for Unity. Version 1.x uses the server SDK for Amazon GameLift Servers 4.x or earlier. For documentation on the latest plugin version, which uses server SDK 5.x and supports newer features such as Amazon GameLift Servers Anywhere and managed container hosting, see [Amazon GameLift Servers plugin for Unity \(server SDK 5.x\)](#).

This topic helps you set up a game client to connect to Amazon GameLift Servers hosted game sessions through a backend service. Use Amazon GameLift Servers APIs to initiate matchmaking, request game session placement, and more.

Add code to the backend service project to allow communication with the Amazon GameLift Servers service. A backend service handles all game client communication with the GameLift service. For more information about backend services, see .

A backend server handles the following game client tasks:

- Customize authentication for your players.
- Request information about active game sessions from the Amazon GameLift Servers service.
- Create a new game session.
- Add a player to an existing game session.
- Remove a player from an existing game session.

Topics

- [Prerequisites](#)
- [Initialize a game client](#)
- [Create game session on a specific fleet](#)
- [Add players to game sessions](#)
- [Remove a player from a game session](#)

Prerequisites

Before setting up game server communication with the Amazon GameLift Servers client, complete the following tasks:

- [Set up an AWS user account](#)
- [Install and set up the plugin](#)
- [Integrate Amazon GameLift Servers with a Unity game server project](#)
- [Deploy hosting fleets for Amazon GameLift Servers](#)

Initialize a game client

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

Add code to initialize a game client. Run this code on launch, it's necessary for other Amazon GameLift Servers functions.

1. Initialize `AmazonGameLiftClient`. Call `AmazonGameLiftClient` with either a default client configuration or a custom configuration. For more information on how to configure a client, see [Set up the Amazon GameLift Servers API](#).
2. Generate a unique player id for each player to connect to a game session. For more information see [Generate player IDs](#).

The following examples shows how to set up an Amazon GameLift Servers client.

```
public class GameLiftClient
{
    private GameLift gl;
    //A sample way to generate random player IDs.
    bool includeBrackets = false;
    bool includeDashes = true;
    string playerId = AZ::Uuid::CreateRandom().ToString<string>(includeBrackets,
includeDashes);

    private Amazon.GameLift.Model.PlayerSession psession = null;
    public AmazonGameLiftClient aglc = null;

    public void CreateGameLiftClient()
    {
        //Access Amazon GameLift Servers service by setting up a configuration.
        //The default configuration specifies a location.
        var config = new AmazonGameLiftConfig();
        config.RegionEndpoint = Amazon.RegionEndpoint.USEast1;

        CredentialProfile profile = null;
        var nscf = new SharedCredentialsFile();
```

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

    }
}
```

Create game session on a specific fleet

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

Add code to start new game sessions on your deployed fleets and make them available to players. After Amazon GameLift Servers has created the new game session and returned a `GameSession`, you can add players to it.

- Place a request for a new game session.
 - If your game uses fleets, call `CreateGameSession()` with a fleet or alias ID, a session name, and maximum number of concurrent players for the game.
 - If your game uses queues, call `StartGameSessionPlacement()`.

The following example shows how to create a game session.

```
public Amazon.GameLift.Model.GameSession()
{
    var cgsreq = new Amazon.GameLift.Model.CreateGameSessionRequest();
    //A unique identifier for the alias with the fleet to create a game session in.
    cgsreq.AliasId = aliasId;
    //A unique identifier for a player or entity creating the game session
    cgsreq.CreatorId = playerId;
    //The maximum number of players that can be connected simultaneously to the game
    session.
    cgsreq.MaximumPlayerSessionCount = 4;
```

```
//Prompt an available server process to start a game session and retrieves
connection information for the new game session
Amazon.GameLift.Model.CreateGameSessionResponse cgsres =
aglc.CreateGameSession(cgsreq);
string gsid = cgsres.GameSession != null ? cgsres.GameSession.GameSessionId : "N/A";
Debug.Log((int)cgsres.HttpStatusCode + " GAME SESSION CREATED: " + gsid);
return cgsres.GameSession;
}
```

Add players to game sessions

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

After Amazon GameLift Servers has created the new game session and returned a `GameSession` object, you can add players to it.

1. Reserve a player slot in a game session by creating a new player session. Use `CreatePlayerSession` or `CreatePlayerSessions` with the game session ID and a unique ID for each player.
2. Connect to the game session. Retrieve the `PlayerSession` object to get the game session's connection information. You can use this information to establish a direct connection to the server process:
 - a. Use the specified port and either the DNS name or IP address of the server process.
 - b. Use the DNS name and port of your fleets. The DNS name and port are required if your fleets have TLS certificate generation enabled.
 - c. Reference the player session ID. The player session ID is required if your game server validates incoming player connections.

The following examples demonstrates how to reserve a player spot in a game session.

```
public Amazon.GameLift.Model.PlayerSession
CreatePlayerSession(Amazon.GameLift.Model.GameSession gsession)
{
```

```
var cpsreq = new Amazon.GameLift.Model.CreatePlayerSessionRequest();
cpsreq.GameSessionId = gsession.GameSessionId;
//Specify game session ID.
cpsreq.PlayerId = playerId;
//Specify player ID.
Amazon.GameLift.Model.CreatePlayerSessionResponse cpsres =
aglc.CreatePlayerSession(cpsreq);
string psid = cpsres.PlayerSession != null ? cpsres.PlayerSession.PlayerSessionId :
"N/A";
return cpsres.PlayerSession;
}
```

The following code illustrates how to connect a player with the game session.

```
public bool ConnectPlayer(int playerId, string playerSessionId)
{
    //Call ConnectPlayer with player ID and player session ID.
    return server.ConnectPlayer(playerId, playerSessionId);
}
```

Remove a player from a game session

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

You can remove the players from the game session when they leave the game.

1. Notify the Amazon GameLift Servers service that a player has disconnected from the server process. Call `RemovePlayerSession` with the player's session ID.
2. Verify that `RemovePlayerSession` returns `Success`. Then, Amazon GameLift Servers changes the player slot to be available, which Amazon GameLift Servers can assign to a new player.

The following example illustrates how to remove a player session.

```
public void DisconnectPlayer(int playerId)
```

```
{
    //Receive the player session ID.
    string playerIdSessionId = playerSessions[playerIdx];
    var outcome = GameLiftServerAPI.RemovePlayerSession(playerSessionId);
    if (outcome.Success)
    {
        Debug.Log (":) PLAYER SESSION REMOVED");
    }
    else
    {
        Debug.Log(":(PLAYER SESSION REMOVE FAILED. RemovePlayerSession()
        returned " + outcome.Error.ToString());
    }
}
```

Install and set up the plugin

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

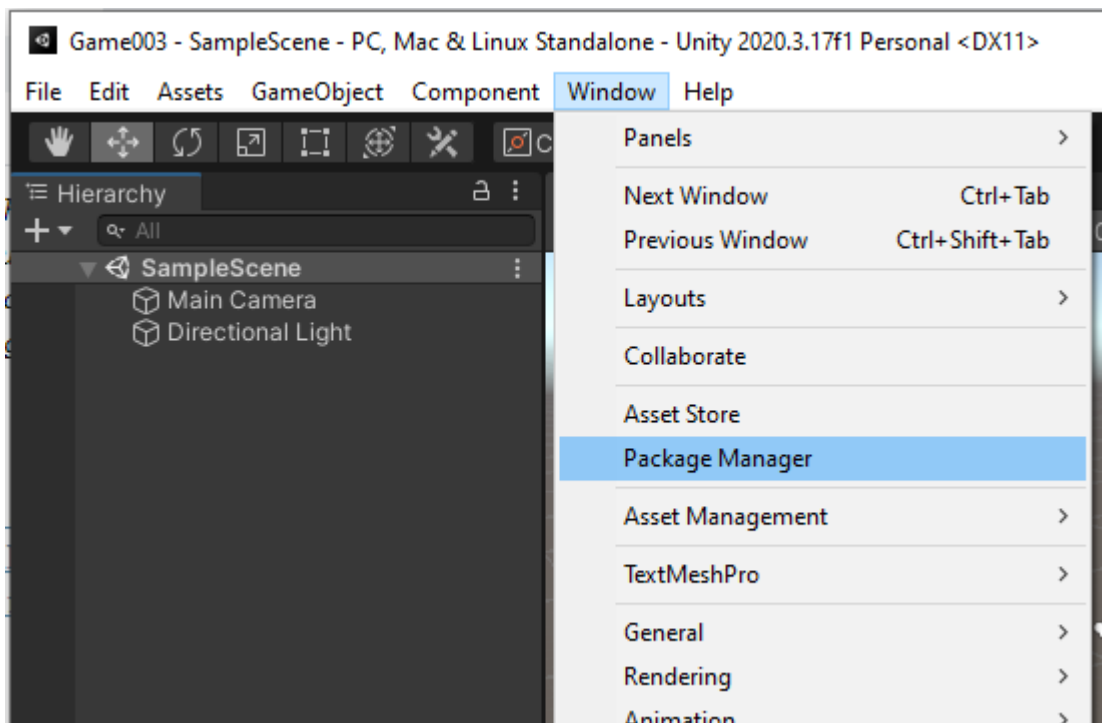
This section describes how to download, install, and set up the Amazon GameLift Servers plugin for Unity, version 1.0.0.

Prerequisites

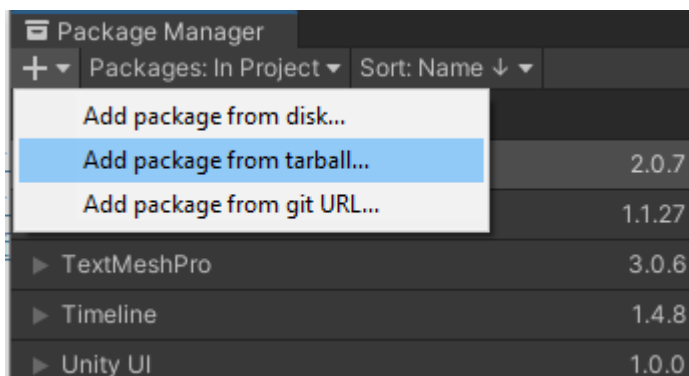
- Unity for Windows 2019.4 LTS, Windows 2020.3 LTS, or Unity for MacOS
- Current version of Java
- Current version of .NET 4.x

To download and install the plugin for Unity

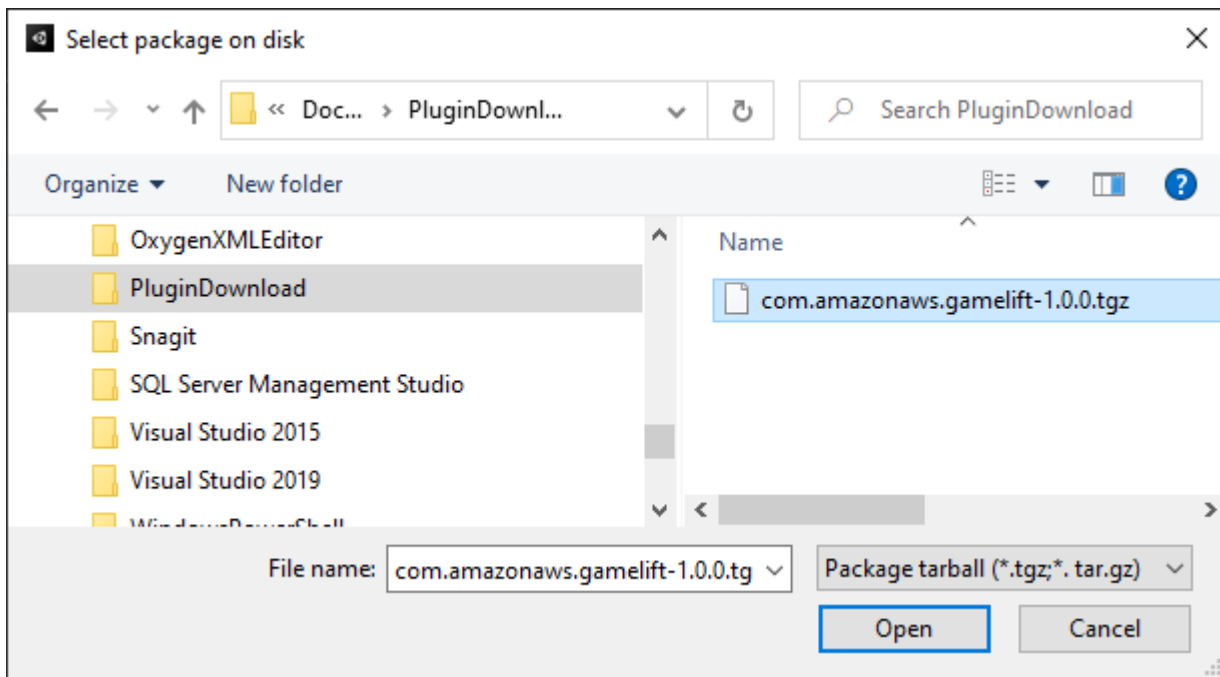
1. Download the Amazon GameLift Servers plugin for Unity. You can find the latest version on the [Amazon GameLift Servers plugin for Unity repository](#) page. Under the [latest release](#), choose **Assets**, and then download the `com.amazonaws.gamelift-version.tgz` file.
2. Launch Unity and choose a project.
3. In the top navigation bar, under **Window** choose **Package Manager**:



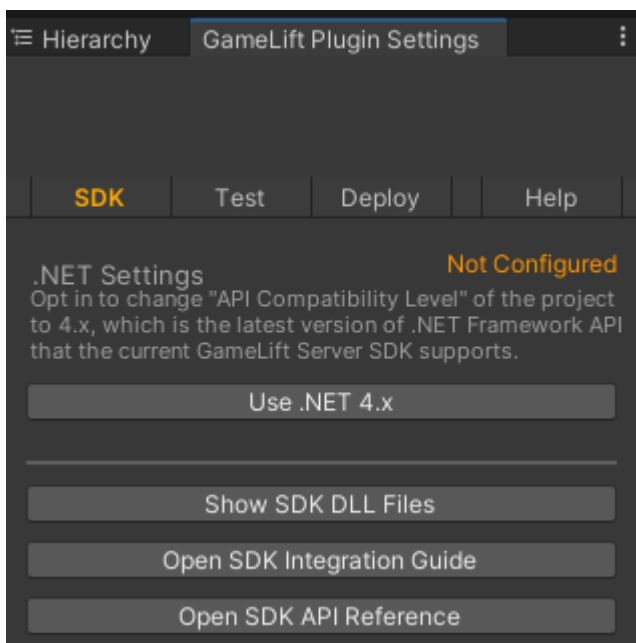
4. Under the **Package Manager** tab choose **+**, and then choose **Add package from tarball...**:



5. In the **Select packages on disk** window, navigate to the `com.amazonaws.gamelift` folder, choose the file `com.amazonaws.gamelift-version.tgz`, and then choose **Open**:



- After Unity has loaded the plug-in, **Amazon GameLift Servers** appears as a new item in the Unity menu. It may take a few minutes to install and recompile scripts. The **Amazon GameLift Servers Plugin Settings** tab automatically opens.



- In the **SDK** pane, choose **Use .NET 4.x**.

When configured, the status changes from **Not Configured** to **Configured**.

Test your game locally

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

Use Amazon GameLift Servers Local to run Amazon GameLift Servers on your local device. You can use Amazon GameLift Servers Local to verify code changes in seconds, without a network connection.

Configure local testing

1. In the plugin for Unity window, choose the **Test** tab.
2. In the **Test** pane, choose **Download Amazon GameLift Servers Local**. The plugin for Unity opens a browser window and downloads the `GameLift_06_03_2021.zip` file to your downloads folder.

The download includes the C# Server SDK, .NET source files, and a .NET component compatible with Unity.

3. Unzip the downloaded file `GameLift_06_03_2021.zip`.
4. In the **Amazon GameLift Servers Plugin Settings** window, choose **Amazon GameLift Servers Local Path**, navigate to the unzipped folder, choose the file `GameLiftLocal.jar`, and then choose **Open**.

When configured, local testing status changes from **Not Configured** to **Configured**.

5. Verify the status of the JRE. If the status is **Not Configured**, choose **Download JRE** and install the recommended Java version.

After you install and configure the Java environment, the status changes to **Configured**.

Run your local game

1. In the plugin for Unity tab, choose the **Test** tab.
2. In the **Test** pane, choose **Open Local Test UI**.
3. In the **Local Testing** window, specify a **Server executable path**. Select ... to select the path and executable name of your server application.

4. In the **Local Testing** window, specify a **GL Local port**.
5. Choose **Deploy & Run** to deploy and run the server.
6. To stop your game server, choose **Stop** or close the game server windows.

Deploy a scenario

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

A scenario uses an CloudFormation template to create the resources you need to deploy a cloud hosting solution for your game. This section describes the scenarios Amazon GameLift Servers provides and how to use them.

Prerequisites

To deploy the scenario, you need an IAM role for the Amazon GameLift Servers service. For information on how to create a role for Amazon GameLift Servers, see [Set up an AWS user account](#).

Each scenario requires permissions to the following resources:

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

Scenarios

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

The Amazon GameLift Servers Plug-in for Unity includes the following scenarios:

Auth only

This scenario creates a game backend service that performs player authentication without game server capability. The template creates the following resources in your account:

- An Amazon Cognito user pool to store player authentication information.
- An Amazon API Gateway REST endpoint-backed AWS Lambda handler that starts games and views game connection information.

Single-Region fleet

This scenario creates a game backend service with a single Amazon GameLift Servers fleet. It creates the following resources:

- An Amazon Cognito user pool for a player to authenticate and start a game.
- An AWS Lambda handler to search for an existing game session with an open player slot on the fleet. If it can't find a open slot, it creates a new game session.

Multi-Region fleet with a queue and custom matchmaker

This scenario forms matches by using Amazon GameLift Servers queues and a custom matchmaker to group together the oldest players in the waiting pool. It creates the following resources:

- An Amazon Simple Notification Service topic that Amazon GameLift Servers publishes messages to. For more information on SNS topics and notifications, see [Set up event notification for game session placement](#).
- A Lambda function that's invoked by the message that communicates placement and game connection details.

- An Amazon DynamoDB table to store placement and game connection details. `GetGameConnection` calls read from this table and return the connection information to the game client.

Spot fleets with a queue and custom matchmaker

This scenario forms matches by using Amazon GameLift Servers queues and a custom matchmaker and configures three fleets. It creates the following resources:

- Two Spot fleets that contain different instance types to provide durability for Spot unavailability.
- An On-Demand fleet that acts as a backup for the other Spot fleets. For more information on designing your fleets, see [Hosting resource customizations](#).
- An Amazon GameLift Servers queue to keep server availability high and cost low. For more information and best practices about queues, see [Customize a game session queue](#).

FlexMatch

This scenario uses FlexMatch, a managed matchmaking service, to match game players together. For more information about FlexMatch, see [What is Amazon GameLift Servers FlexMatch](#). This scenario creates the following resources:

- A Lambda function to create a matchmaking ticket after it receives `StartGame` requests.
- A separate Lambda function to listen to FlexMatch match events.

To avoid unnecessary charges on your AWS account, remove the resources created by each scenario after you are done using them. Delete the corresponding CloudFormation stack.

Update AWS credentials

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

The Amazon GameLift Servers plugin for Unity requires security credentials to deploy a scenario. You can either create new credentials or use existing credentials.

For more information about configuring credentials, see [Understanding and getting your AWS credentials](#).

To update AWS credentials

1. In Unity, in the plugin for Unity tab, choose the **Deploy** tab.
2. In the **Deploy** pane, choose **AWS Credentials**.
3. You can create new AWS credentials or choose existing credentials.
 - To create credentials, choose **Create new credentials profile**, and then specify the **New Profile Name**, **AWS Access Key ID**, **AWS Secret Key**, and **AWS Region**.
 - To choose existing credentials, choose **Choose existing credentials profile** and then select a profile name and **AWS Region**.
4. In the **Update AWS Credentials** window, choose **Update Credentials Profile**.

Update account bootstrap

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

The bootstrap location is an Amazon S3 bucket used during deployment. It's used to store game server assets and other dependencies. The AWS Region you choose for the bucket must be the same Region you will use for the scenario deployment.

For more information about Amazon S3 buckets, see [Creating, configuring, and working with Amazon Simple Storage Service buckets](#).

To update the account bootstrap location

1. In Unity, in the plugin for Unity tab, choose the **Deploy** tab.
2. In the **Deploy** pane, choose **Update Account Bootstrap**.
3. In the **Account Bootstrapping** window, you choose an existing Amazon S3 bucket or create a new Amazon S3 bucket:
 - To choose an existing bucket, choose **Choose existing Amazon S3 bucket** and **Update** to save your selection.

- Choose **Create new Amazon S3 bucket** to create a new Amazon Simple Storage Service bucket, then choose a **Policy**. The policy specifies when the Amazon S3 bucket will be expire. Choose **Create** to create the bucket.

Deploy a game scenario

Note

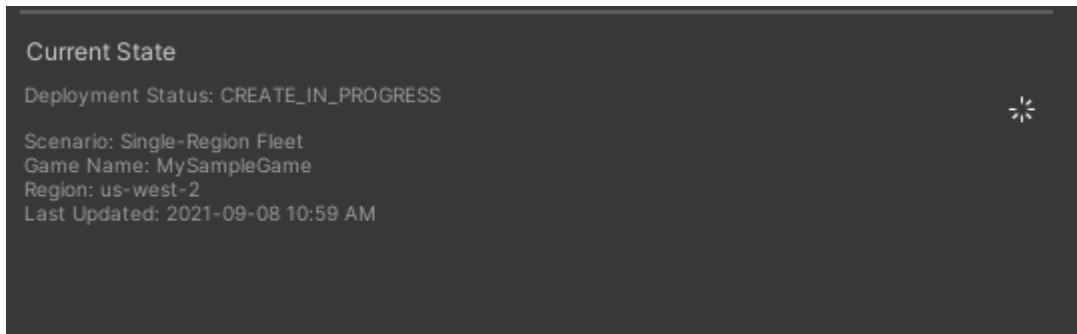
This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

You can use a scenario to test your game with Amazon GameLift Servers. Each scenario uses a CloudFormation template to create a stack with the required resources. Most of the scenarios require a game server executable and build path. When you deploy the scenario, Amazon GameLift Servers copies game assets to the bootstrap location as part of deployment.

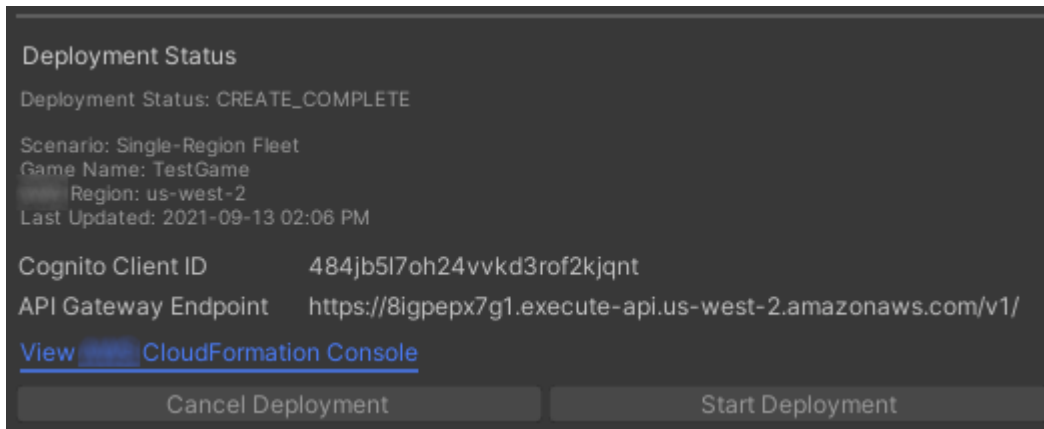
You must configure AWS credentials and an AWS account bootstrap to deploy a scenario.

To deploy a scenario

1. In Unity, in the plugin for Unity tab, choose the **Deploy** tab.
2. In the **Deploy** pane, choose **Open Deployment UI**.
3. In the **Deployment** window, choose a scenario.
4. Enter a **Game Name**. It must be unique. The game name is part of the CloudFormation stack name when you deploy the scenario.
5. Choose the **Game Server Build Folder Path**. The build folder path points to the folder containing the server executable and dependencies.
6. Choose the **Game Server Build .exe File Path**. The build executable file path points to the game server executable.
7. Choose **Start Deployment** to begin deploying a scenario. You can follow the status of the update in the **Deployment** window under **Current State**. Scenarios can take several minutes to deploy.



- When the scenario completes deployment, the **Current State** updates to include the **Cognito Client ID** and **API Gateway Endpoint** that you can copy and paste into the game.



- To update game settings, on the Unity menu, choose **Go To Client Connection Settings**. This displays an **Inspector** tab on the right side of the Unity screen.
- Deselect **Local Testing Mode**.
- Enter the **API Gateway Endpoint** and the **Cognito Client ID**. Choose the same AWS Region you used for the scenario deployment. You can then rebuild and run the game client using the deployed scenario resources.

Deleting resources created by the scenario

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

To delete the resources created for the scenario, delete the corresponding CloudFormation stack.

To delete resources created by the scenario

1. In the Amazon GameLift Servers plugin for Unity **Deployment** window, select **View AWS CloudFormation Console** to open the CloudFormation console.
2. In the CloudFormation console, choose **Stacks**, and then choose the stack that includes the game name specified during deployment.
3. Choose **Delete** to delete the stack. It may take a few minutes to delete a stack. After CloudFormation deletes the stack used by the scenario, its status changes to `ROLLBACK_COMPLETE`.

Integrate games with Amazon GameLift Servers in Unity

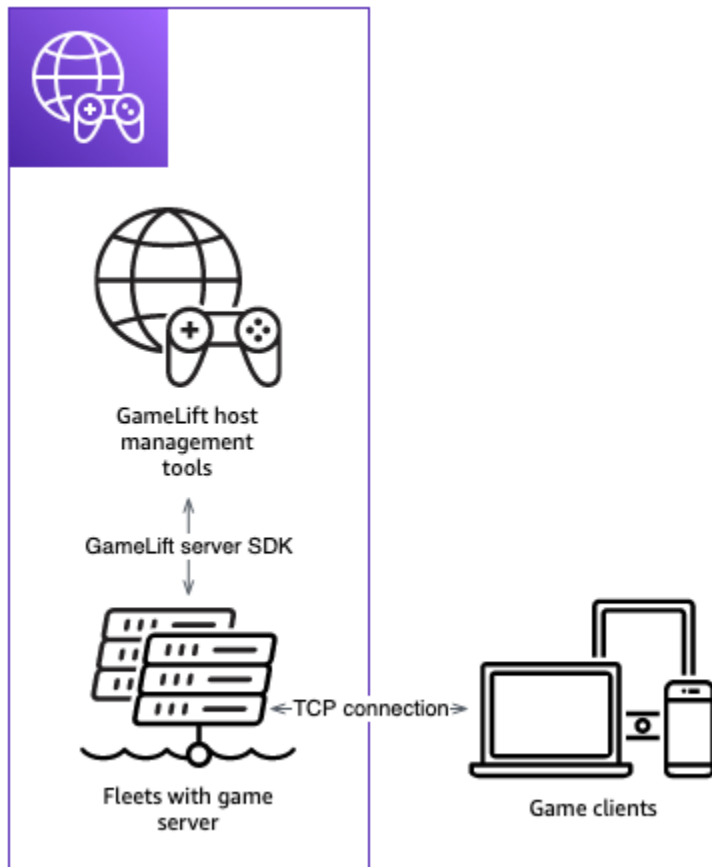
Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

Integrate your Unity game with Amazon GameLift Servers by completing the following tasks:

- [Integrate Amazon GameLift Servers with a Unity game server project](#)
- [Integrate Amazon GameLift Servers with a Unity game client project](#)

The following diagram shows an example flow of integrating a game. In the diagram, a fleet with the game server is deployed to Amazon GameLift Servers. The game client communicates with the game server, which communicates with Amazon GameLift Servers.



Import and run a sample game

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

The Amazon GameLift Servers plugin for Unity includes a sample game you can use to explore the basics of integrating your game with Amazon GameLift Servers. In this section, you build the game client and game server and then test locally using Amazon GameLift Servers Local.

Prerequisites

- [Set up an AWS user account](#)
- [Install and set up the plugin](#)

Build and run the sample game server

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

Set up the game server files of the sample game.

1. In Unity, on the menu, choose **Amazon GameLift Servers**, and then choose **Import Sample Game**.
2. In the **Import Sample Game** window, choose **Import** to import the game, its assets and dependencies.
3. Build the game server. In Unity, on the menu, choose **Amazon GameLift Servers**, and then choose **Apply Windows Sample Server Build Settings** or **Apply MacOS Sample Server Build Settings**. After you configure the game server settings, Unity recompiles the assets.
4. In Unity, on the menu, choose **File**, and then choose **Build**. Choose **Server Build**, choose **Build**, and then choose a build folder specifically for server files.

Unity builds the sample game server, placing the executable and required assets in the specified build folder.

Build and run the sample game client

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

Set up the game client files of the sample game.

1. In Unity, on the menu, choose **Amazon GameLift Servers**, and then choose **Apply Windows Sample Client Build Settings** or **Apply MacOS Sample Client Build Settings**. After the game client settings are configured, Unity will recompile assets.

2. In Unity, on the menu, select **Go To Client Settings**. This will display an **Inspector** tab on the right side of the Unity screen. In the **Amazon GameLift Servers Client Settings** tab, select **Local Testing Mode**.
3. Build the game client. In Unity, on the menu, choose **File**. Confirm **Server Build** is not checked, choose **Build**, and then select a build folder specifically for client files.

Unity builds the sample game client, placing the executable and required assets in the specified client build folder.

4. You've now built the game server and client. In the next steps, you run the game and see how it interacts with Amazon GameLift Servers.

Test the sample game locally

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

Run the sample game you imported using Amazon GameLift Servers Local.

1. Launch the game server. In Unity, in the plugin for Unity tab, choose the **Deploy** tab.
2. In the **Test** pane, choose **Open Local Test UI**.
3. In the **Local Testing** window, specify a **Game Server .exe File Path**. The path must include the executable name. For example, `C:/MyGame/GameServer/MyGameServer.exe`.
4. Choose **Deploy and Run**. The plugin for Unity launches the game server and opens an Amazon GameLift Servers Local log window. The window contains log messages including messages sent between the game server and Amazon GameLift Servers Local.
5. Launch the game client. Find the build location with the sample game client and choose the executable file.
6. In the **Amazon GameLift Servers Sample Game**, provide an email and password and then choose **Log In**. The email and password aren't validated or used.
7. In the **Amazon GameLift Servers Sample Game**, choose **Start**. The game client looks for a game session. If it can't find a session, it creates one. The game client then starts the game session. You can see game activity in the logs.

Sample game server logs

```
...
2021-09-15T19:55:3495 PID:20728 Log :) GAMELIFT AWAKE
2021-09-15T19:55:3512 PID:20728 Log :) I AM SERVER
2021-09-15T19:55:3514 PID:20728 Log :) GAMELIFT StartServer at port 33430.
2021-09-15T19:55:3514 PID:20728 Log :) SDK VERSION: 4.0.2
2021-09-15T19:55:3556 PID:20728 Log :) SERVER IS IN A GAMELIFT FLEET
2021-09-15T19:55:3577 PID:20728 Log :) PROCESSREADY SUCCESS.
2021-09-15T19:55:3577 PID:20728 Log :) GAMELIFT HEALTH CHECK REQUESTED (HEALTHY)
...
2021-09-15T19:55:3634 PID:20728 Log :) GAMELOGIC AWAKE
2021-09-15T19:55:3635 PID:20728 Log :) GAMELOGIC START
2021-09-15T19:55:3636 PID:20728 Log :) LISTENING ON PORT 33430
2021-09-15T19:55:3636 PID:20728 Log SERVER: Frame: 0 HELLO WORLD!
...
2021-09-15T19:56:2464 PID:20728 Log :) GAMELIFT SESSION REQUESTED
2021-09-15T19:56:2468 PID:20728 Log :) GAME SESSION ACTIVATED
2021-09-15T19:56:3578 PID:20728 Log :) GAMELIFT HEALTH CHECK REQUESTED (HEALTHY)
2021-09-15T19:57:3584 PID:20728 Log :) GAMELIFT HEALTH CHECK REQUESTED (HEALTHY)
2021-09-15T19:58:0334 PID:20728 Log SERVER: Frame: 8695 Connection accepted: playerId
  0 joined
2021-09-15T19:58:0335 PID:20728 Log SERVER: Frame: 8696 Connection accepted: playerId
  1 joined
2021-09-15T19:58:0338 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from playerId 0 Msg:
  CONNECT: server IP localhost
2021-09-15T19:58:0338 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from player 0:CONNECT:
  server IP localhost
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 CONNECT: player index 0
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from playerId 1 Msg:
  CONNECT: server IP localhost
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from player 1:CONNECT:
  server IP localhost
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 CONNECT: player index 1
```

Sample Amazon GameLift Servers Local logs

```
12:55:26,000 INFO || - [SocketIOServer] main - Session store / pubsub factory used:
  MemoryStoreFactory (local session store only)
12:55:28,092 WARN || - [ServerBootstrap] main - Unknown channel option 'SO_LINGER' for
  channel '[id: 0xe23d0a14]'
12:55:28,101 INFO || - [SocketIOServer] nioEventLoopGroup-2-1 - SocketIO server
  started at port: 5757
```

```

12:55:28,101 INFO || - [SDKConnection] main - GameLift SDK server (communicates with
your game server) has started on http://localhost:5757
12:55:28,120 INFO || - [SdkWebSocketServer] WebSocketSelector-20 - WebSocket Server
started on address localhost/127.0.0.1:5759
12:55:28,166 INFO || - [StandAloneServer] main - GameLift Client server (listens for
GameLift client APIs) has started on http://localhost:8080
12:55:28,179 INFO || - [StandAloneServer] main - GameLift server sdk http listener has
started on http://localhost:5758
12:55:35,453 INFO || - [SdkWebSocketServer] WebSocketWorker-12 - onOpen
socket: /?pID=20728&sdkVersion=4.0.2&sdkLanguage=CSharp and handshake /?
pID=20728&sdkVersion=4.0.2&sdkLanguage=CSharp
12:55:35,551 INFO || - [HostProcessManager] WebSocketWorker-12 - client connected with
pID 20728
12:55:35,718 INFO || - [GameLiftSdkHttpHandler] GameLiftSdkHttpHandler-thread-0 -
GameLift API to use: ProcessReady for pId 20728
12:55:35,718 INFO || - [ProcessReadyHandler] GameLiftSdkHttpHandler-thread-0 -
Received API call for processReady from 20728
12:55:35,738 INFO || - [ProcessReadyHandler] GameLiftSdkHttpHandler-thread-0 -
onProcessReady data: port: 33430
12:55:35,739 INFO || - [HostProcessManager] GameLiftSdkHttpHandler-thread-0 -
Registered new process with pId 20728
12:55:35,789 INFO || - [GameLiftSdkHttpHandler] GameLiftSdkHttpHandler-thread-0 -
GameLift API to use: ReportHealth for pId 20728
12:55:35,790 INFO || - [ReportHealthHandler] GameLiftSdkHttpHandler-thread-0 -
Received API call for ReportHealth from 20728
12:55:35,794 INFO || - [ReportHealthHandler] GameLiftSdkHttpHandler-thread-0 -
ReportHealth data: healthStatus: true
12:56:24,098 INFO || - [GameLiftHttpHandler] Thread-12 - API to use:
GameLift.DescribeGameSessions
12:56:24,119 INFO || - [DescribeGameSessionsDispatcher] Thread-12 - Received API call
to describe game sessions with input: {"FleetId":"fleet-123"}
12:56:24,241 INFO || - [GameLiftHttpHandler] Thread-12 - API to use:
GameLift.CreateGameSession
12:56:24,242 INFO || - [CreateGameSessionDispatcher] Thread-12 - Received API call to
create game session with input: {"FleetId":"fleet-123","MaximumPlayerSessionCount":4}
12:56:24,265 INFO || - [HostProcessManager] Thread-12 - Reserved process:
20728 for gameSession: arn:aws:gamelift:local::gamesession/fleet-123/
gsess-59f6cc44-4361-42f5-95b5-fdb5825c0f3d
12:56:24,266 INFO || - [WebSocketInvoker] Thread-12 - StartGameSessionRequest:
gameSessionId=arn:aws:gamelift:local::gamesession/fleet-123/
gsess-59f6cc44-4361-42f5-95b5-fdb5825c0f3d, fleetId=fleet-123, gameSessionName=null,
maxPlayers=4, properties=[], ipAddress=127.0.0.1, port=33430, gameSessionData?=false,
matchmakerData?=false, dnsName=localhost

```

```
12:56:24,564 INFO || - [CreateGameSessionDispatcher] Thread-12 - GameSession with  
id: arn:aws:gamelift:local::gamesession/fleet-123/gsess-59f6cc44-4361-42f5-95b5-  
fdb5825c0f3d created  
12:56:24,585 INFO || - [GameLiftHttpHandler] Thread-12 - API to use:  
GameLift.DescribeGameSessions  
12:56:24,585 INFO || - [DescribeGameSessionsDispatcher] Thread-12 - Received API call  
to describe game sessions with input: {"FleetId":"fleet-123"}  
12:56:24,660 INFO || - [GameLiftSdkHttpHandler] GameLiftSdkHttpHandler-thread-0 -  
GameLift API to use: GameSessionActivate for pId 20728  
12:56:24,661 INFO || - [GameSessionActivateHandler] GameLiftSdkHttpHandler-thread-0 -  
Received API call for GameSessionActivate from 20728  
12:56:24,678 INFO || - [GameSessionActivateHandler] GameLiftSdkHttpHandler-thread-0  
- GameSessionActivate data: gameId: "arn:aws:gamelift:local::gamesession/  
fleet-123/gsess-59f6cc44-4361-42f5-95b5-fdb5825c0f3d"
```

Shut down server process

Note

This topic refers to Amazon GameLift Servers plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

After you're done with your sample game, shut down the server in Unity.

1. In the game client, choose **Quit** or close the window to stop the game client.
2. In Unity, in the **Local Testing** window, choose **Stop** or close the game server windows to stop the server.

Integrate a game server with Amazon GameLift Servers

Your custom game server must be able to interact with Amazon GameLift Servers when deployed and running on hosting resources. The topics in this section describes how to integrate your game server project with the server SDK for Amazon GameLift Servers to add required and optional functionality. This section includes general instructions that you can use with any of the language-specific server SDKs. It also includes detailed guidance on integrating your Unreal Engine and Unity game projects.

Topics

- [Add Amazon GameLift Servers to your game server with the server SDK](#)
- [Integrate the server SDK for game engines](#)

Add Amazon GameLift Servers to your game server with the server SDK

This topic provides general guidance on the server SDK functionality that you need to add to your game server code/ This functionality enables game server processes to communicate with the Amazon GameLift Servers service. Use these instructions for game servers that you plan to deploy onto Amazon GameLift Servers managed EC2 fleets, managed container fleets, or Anywhere fleets.

Game server processes communicate with the Amazon GameLift Servers service to receive instructions from the service and to report server process health and game session status. For detailed information on interactions between your game hosting solution components (game server, backend service, game client, and Amazon GameLift Servers), see [Game client/server interactions with Amazon GameLift Servers](#).

Get the server SDK

To prepare your game for hosting, add the server SDK for Amazon GameLift Servers to your game server project. The Server SDK is available in several languages. For more information about tool support for game servers, including the server SDK, see [Get Amazon GameLift Servers development tools](#).

Download the server SDK for your development language from github.com/amazon-gamelift. You can find repos for the server SDK versions, plugins, and other development tools.

Note

If you're developing using Unreal Engine or Unity, download the Amazon GameLift Servers plugin for these game engines. The server SDK is built in and ready to use. See these links for additional information:

- Unreal Engine ([Download plugin](#)) ([Integration guide](#))
- Unity ([Download plugin](#)) ([Integration guide](#))

Server SDK API references:

- [C++ server SDK 5.x for Amazon GameLift Servers -- Actions](#)

- [C# server SDK 5.x for Amazon GameLift Servers -- Actions](#)
- [C++ \(Unreal\) server SDK 5.x for Amazon GameLift Servers -- Actions](#)
- [Go server SDK for Amazon GameLift Servers -- Actions](#)

Initialize the server process

Add code to establish communication with Amazon GameLift Servers service and report when the game server process is ready to host a game session. This code must run before any Amazon GameLift Servers code.

1. Initialize an Amazon GameLift Servers API client by calling `InitSdk()`. If you're preparing your game server to run on Amazon GameLift Servers managed EC2 fleets, use the default `InitSDK()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) with no parameters. The API client handles connection to the Amazon GameLift Servers service for you.

If you're preparing your game server for use on an Amazon GameLift Servers Anywhere fleet:

Initialize the Amazon GameLift Servers API client by calling `InitSdk()` with the following `ServerParameters`:

- The URL of the websocket used to connect to your game server.
- The ID of the process used to host your game server.
- The ID of the compute hosting your game server processes.
- The ID of the fleet containing your Amazon GameLift Servers Anywhere compute.
- The authorization token generated by the Amazon GameLift Servers operation [GetComputeAuthToken](#).

2. Notify the service that the game server process is ready to host a game session. Call `ProcessReady()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) with the following `ProcessParameters`. Each game server process must call `ProcessReady()` only once.
 - The port number for the server process. When the server process starts a game session, it provides the port to the Amazon GameLift Servers service, which updates the game session information. Your game can retrieve this information and provide it to game clients, which use it to connect to the server process and join the game session.

- The locations of files that you want Amazon GameLift Servers to store for you. These might include game session logs and other files that the server process generates during a game session. Although Amazon GameLift Servers saves these files temporarily on the compute where the server process is running, these are only available until the instance shuts down. You can access stored files through the [Amazon GameLift Servers console](#) or by calling the Amazon GameLift Servers API operation [GetGameSessionLogUrl\(\)](#).

** If you're preparing your game server for use on an Amazon GameLift Servers managed containers fleet:**

You don't need to specify log parameters for a container fleet. Instead, send game session and other log data to standard output. Container fleets automatically capture all container standard output as a log stream.

- The following callback functions that allow Amazon GameLift Servers to send messages or prompts to a game server process. You must implement each of these functions in your game server code. For more information, see `ProcessParameters`([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) .
 - (Optional) `onHealthCheck` – Amazon GameLift Servers calls this function regularly to request a health status report from the server.
 - `onStartGameSession` – Amazon GameLift Servers calls this function in response to the client request [CreateGameSession\(\)](#).
 - `onProcessTerminate` – Amazon GameLift Servers forces the server process to stop, letting it shut down gracefully.
 - (Optional) `onUpdateGameSession` – Amazon GameLift Servers delivers an updated game session object to the game server or provides a status update on a match backfill request. The [FlexMatch backfill](#) feature requires this callback.

You can also set up a game server so that it can securely access other AWS resources that you own or control. For more information, see [Connect your Amazon GameLift Servers hosted game server to other AWS resources](#).

(Optional) Report server process health

Add code to your game server to implement the callback function `onHealthCheck()`. Amazon GameLift Servers invokes this callback method periodically to collect health metrics. To implement this callback function, do the following:

- Evaluate the health status of the server process. For example, you might report the server process as unhealthy if any external dependencies have failed.
- Complete the health evaluation and respond to the callback within 60 seconds. If Amazon GameLift Servers doesn't receive a response in that time, it automatically considers the server process to be unhealthy.
- Return a Boolean value: true for healthy, false for unhealthy.

If you don't implement a health check callback, then Amazon GameLift Servers considers the server process to be healthy unless the server doesn't respond.

The Amazon GameLift Servers service uses server process health to end unhealthy processes and clear up resources. If a server process continues to report as unhealthy or doesn't respond for three consecutive health checks, then the service might shut down the process and start a new one. The service collects metrics on a fleet's server process health.

(Optional) Get a TLS certificate

If the server process is running on a fleet that has TLS certificate generation activated, then you can retrieve the TLS certificate to establish a secure connection with a game client and to encrypt client server communication. A copy of the certificate is stored on the instance. To get the file location, call `GetComputeCertificate()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) .

Start a game session

Add code to implement the callback function `onStartGameSession`. Amazon GameLift Servers invokes this callback to start a game session on the server process.

The `onStartGameSession` function takes a [GameSession](#) object as an input parameter. This object includes key game session information, such as maximum players. It can also include game data and player data. The function implementation should do the following tasks:

- Initiate actions to create a new game session based on the `GameSession` properties. At minimum, the game server must associate the game session ID, which game clients reference when connecting to the server process.
- Process game data and player data as needed. This data is in the `GameSession` object.
- Notify the Amazon GameLift Servers service when a new game session is ready to accept players. Call the server API operation `ActivateGameSession()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) . In response to a successful call, the service changes the game session status to `ACTIVE`.

(Optional) Validate a new player

If you're tracking the status of player sessions, then add code to validate a new player when they connect to a game server. Amazon GameLift Servers tracks current players and available game session slots.

For validation, a game client that tries to join a game session must include a player session ID. Amazon GameLift Servers generates this ID when your game starts new game sessions by calling [StartGameSessionPlacement\(\)](#) or [StartMatchmaking\(\)](#). On these requests, an open slot in a game session is reserved for the player session.

When the game server process receives a game client connection request, it calls `AcceptPlayerSession()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) with the player session ID. In response, Amazon GameLift Servers verifies that the player session ID corresponds to an open slot reserved in the game session. After Amazon GameLift Servers validates the player session ID, the server process accepts the connection. The player can then join the game session. If Amazon GameLift Servers doesn't validate the player session ID, then the server process denies the connection.

(Optional) Report a player session ending

If you're tracking the status of player sessions, add code to notify Amazon GameLift Servers when a player leaves the game session. This code should run whenever the server process detects a dropped connection. Amazon GameLift Servers uses this notification to track current players and available slots in the game session.

To handle dropped connections in your code, add a call to the server API operation `RemovePlayerSession()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) with the corresponding player session ID.

End a game session

Add code to the server process shutdown sequence to notify Amazon GameLift Servers when a game session is ending. To recycle and refresh hosting resources, shut down each server process after the game session is complete.

At the start of the server process shutdown code, call the server API operation `ProcessEnding()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) g . This call notifies Amazon GameLift Servers that the server process is shutting down. Amazon GameLift Servers changes the game session status and server process status to `TERMINATED`. After calling `ProcessEnding()`, it's safe for the process to shut down.

Respond to a server process shutdown notification

Add code to shut down the server process in response to a notification from the Amazon GameLift Servers service. The service sends this notification when the server process consistently reports unhealthy, or if the instance where the server process is running is being terminated. Amazon GameLift Servers can stop an instance as part of a capacity scale-down event, or in response to Spot Instance interruption. A Spot Instance interruption provides a two-minute notice, which gives the server process time to disconnect players gracefully, preserve game state data, and perform other cleanup tasks.

To handle a shutdown notification, make the following changes to your game server code:

- Implement the callback function `onProcessTerminate()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) . This function should call code that shuts down the server process.
- Call the server API operation `GetTerminationTime()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) from your game server shutdown code. If Amazon GameLift Servers has issued a call to stop the server process, then `GetTerminationTime()` returns the estimated termination time.
- At the start of your game server shutdown code, call the server API operation `ProcessEnding()` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) . This call notifies the Amazon GameLift Servers service that the server process is shutting down. The service then changes the server process status to `TERMINATED`. After calling `ProcessEnding()`, it's safe for the process to shut down.

Integrate the server SDK for game engines

You can use the managed Amazon GameLift Servers service with most major game engines that support C++ or C# libraries, including O3DE, Unreal Engine, and Unity. Build the version you need for your game; see the README files with each version for build instructions and minimum requirements. For more information on available Amazon GameLift Servers SDKs, supported development platforms and operating systems, see [Get Amazon GameLift Servers development tools](#) for game servers.

In addition to the engine-specific information provided in this topic, find additional help with integrating Amazon GameLift Servers into your game servers, clients and services in the following topics:

- [Add Amazon GameLift Servers to your game server with the server SDK](#) – Detailed instructions on integrating Amazon GameLift Servers into a game server.

- [Integrate Amazon GameLift Servers game client functionality](#) – Detailed instructions on integrating into a game client or service, including creating game sessions and joining players to games.

Unreal Engine

Game servers

Prepare your game servers for hosting on Amazon GameLift Servers by adding the [server SDK for Amazon GameLift Servers for Unreal Engine](#) to your project and implementing the required server functionality. For help setting up the Unreal Engine plugin and adding Amazon GameLift Servers code, see [Integrate Amazon GameLift Servers into an Unreal Engine project](#).

Game clients and services

Enable your game clients and/or game services to interact with Amazon GameLift Servers service, such as to find available game sessions or create new ones, and add players to games. Core client functionality is provided in the [AWS SDK for C++](#). To integrate Amazon GameLift Servers into your Unreal Engine game project, see [Integrate Amazon GameLift Servers game client functionality](#).

Unity

Game servers

Prepare your game servers for hosting on Amazon GameLift Servers by adding the [server SDK for Amazon GameLift Servers for C#](#) to your project and implementing the required server functionality. For help setting up with Unity and adding Amazon GameLift Servers code, see [Integrate Amazon GameLift Servers into a Unity project](#).

Game clients and services

Enable your game clients and/or game services to interact with Amazon GameLift Servers service, such as to find available game sessions or create new ones, and add players to games. Core client functionality is provided in the [AWS SDK for .NET](#). To integrate Amazon GameLift Servers into your Unity game project, see [Integrate Amazon GameLift Servers game client functionality](#).

Other engines

For a full list of the Amazon GameLift Servers SDKs available for game servers and clients, see [the section called “Get development tools”](#).

Integrate Amazon GameLift Servers into an Unreal Engine project

Learn how to integrate the Amazon GameLift Servers SDK for Unreal Engine into your game projects to access the full server SDK feature set.

Tip

For faster deployment, try the Amazon GameLift Servers standalone plugin for Unreal Engine. It provides guided UI workflows to quickly deploy your game server with minimal setup, so you can try out your game components in action. See [Amazon GameLift Servers plugin for Unreal Engine](#).

Additional resources:

- [C++ \(Unreal\) server SDK 5.x for Amazon GameLift Servers -- Actions](#)
- [Get Amazon GameLift Servers development tools](#)

Install the server SDK for Unreal

Get the open-source Amazon GameLift Servers SDK for Unreal Engine from [GitHub](#). The repository's readme files contain prerequisites and installation instructions.

Set up build targets and module rules

Modify your game project files to properly generate build components for use with Amazon GameLift Servers.

To add client and server build targets:

1. Open your game project's code files and locate the file `.../Games/[your application name]Source/[your application name]Target.cs` file. Example: `.../Source/GameLiftUnrealAppTarget.cs`. (If you use Visual Studio, open the project's `.sln` file.)
2. Copy this file to create two new target files in the `Source/` directory.
 - Client target – Rename the new file to `[your application name]Client.Target.cs`. Edit the contents to update the class name and target type values, as illustrated in the following sample code:

```
using UnrealBuildTool;
```

```
using System.Collections.Generic;

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

- Server target – Rename the new file to *[your application name]*Server.Target.cs. Edit the contents to update the class name and target type values, as illustrated in the following sample code:

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

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

3. Regenerate your project files. If you're using Visual Studio, you can right-click your game project's .uproject file and select **Generate Visual Studio Project Files**.

To update the game project module rules:

Update the game project's module rules to take a dependency on the plugin.

1. Open your game project's code files and locate the file `.../Games/[your application name]Source/[your application name].Build.cs` file. Example: `.../Source/GameLiftUnrealApp.Build.cs`. (If you use Visual Studio, open the project's `.sln` file.)
2. Locate the `ModuleRules` class and update as illustrated in the following sample code:

```
using UnrealBuildTool;

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

3. After creating the new target files and modifying the module rules, rebuild your game project.

Add game hosting functionality to your server code

After server SDK installation and setup, the next step is to integrate game hosting functionality into your server code. The server SDK enables your game server to communicate with the Amazon GameLift Servers service, receive instructions for game sessions, report status and health, and perform other actions.

This topic provides sample code that adds the minimum functionality required to host your game with Amazon GameLift Servers.

Step 1: Update the GameMode header file

1. Open your game project's code files and locate the file *Your-application-name*GameMode.h file. Example: GameLiftUnrealAppGameMode.h. If you use Visual Studio, open the .sln file for your game project.
2. Change the header file to include the following example code. Be sure to replace "GameLiftUnrealApp" with your own application name.

Sample code for GameMode.h

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

#pragma once

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

struct FProcessParameters;

DECLARE_LOG_CATEGORY_EXTERN(GameServerLog, Log, All);

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

public:
    AGameLiftUnrealAppGameMode();

protected:
    virtual void BeginPlay() override;

private:
    void InitGameLift();

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

Step 2: Add required server SDK calls to your game server code

Use the sample code in this section to integrate your game server code for use with Amazon GameLift Servers. For details on what the code does, see [Initialize the server process](#) and [C++ \(Unreal\) server SDK 5.x for Amazon GameLift Servers -- Actions](#).

Note

The `WITH_GAMELIFT` preprocessor flag serves two purposes:

- Restricts Amazon GameLift Servers backend API calls to Unreal server builds only
- Ensures compatibility across different Unreal build targets

1. Open the related source file `Your-application-nameGameMode.cpp` file. In our example: `GameLiftUnrealAppGameMode.cpp`.
2. Change the code to align with the following example code. Be sure to replace any instance of "GameLiftUnrealApp" with your own application name.

The code sample provided shows how to add the required elements for integration with Amazon GameLift Servers. These include:

- Initialize an Amazon GameLift Servers API client.
- Implement callback functions to respond to requests from the Amazon GameLift Servers service, including `OnStartGameSession`, `OnProcessTerminate`, and `onHealthCheck`.
- Call `ProcessReady()` to notify the Amazon GameLift Servers service when ready to host game sessions.

Sample code for GameMode.cpp

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

#include "GameLiftUnrealAppGameMode.h"

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

#if WITH_GAMELIFT
```

```

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

#include "GenericPlatform/GenericPlatformOutputDevices.h"

DEFINE_LOG_CATEGORY(GameServerLog);

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

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

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

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

#ifdef WITH_GAMELIFT
    InitGameLift();
#endif
}

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

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

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

```

```

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

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

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

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

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

```

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

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

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

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

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

FString glAnywhereSessionToken = "";
if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereSessionToken="),
glAnywhereSessionToken))
{
```

```

        ServerParametersForAnywhere.m_sessionToken =
TCHAR_TO_UTF8(*glAnywhereSessionToken);
    }

    UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_YELLOW);
    UE_LOG(GameServerLog, Log, TEXT(">>>> WebSocket URL: %s"),
*ServerParametersForAnywhere.m_webSocketUrl);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Fleet ID: %s"),
*ServerParametersForAnywhere.m_fleetId);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Process ID: %s"),
*ServerParametersForAnywhere.m_processId);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Host ID (Compute Name): %s"),
*ServerParametersForAnywhere.m_hostId);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Auth Token: %s"),
*ServerParametersForAnywhere.m_authToken);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Aws Region: %s"),
*ServerParametersForAnywhere.m_awsRegion);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Access Key: %s"),
*ServerParametersForAnywhere.m_accessKey);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Secret Key: %s"),
*ServerParametersForAnywhere.m_secretKey);
    UE_LOG(GameServerLog, Log, TEXT(">>>> Session Token: %s"),
*ServerParametersForAnywhere.m_sessionToken);
    UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
}

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

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

```

```

    UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
    return;
}

ProcessParameters = MakeShared<FProcessParameters>();

//When a game session is created, Amazon GameLift Servers sends an activation
request to the game server and passes along the game session object containing game
properties and other settings.
//Here is where a game server should take action based on the game session object.
//Once the game server is ready to receive incoming player connections, it should
invoke GameLiftServerAPI.ActivateGameSession()
ProcessParameters->OnStartGameSession.BindLambda( [=]
(Aws::GameLift::Server::Model::GameSession InGameSession)
{
    FString GameSessionId = FString(InGameSession.GetGameSessionId());
    UE_LOG(GameServerLog, Log, TEXT("GameSession Initializing: %s"),
*GameSessionId);
    GameLiftSdkModule->ActivateGameSession();
});

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

```

```

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

});

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

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

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

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

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

```



```

}

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

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

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

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

```

Step 3: Rebuild the game project

- Build game project for both of the following target types: *Development Editor* and *Development Server*.

Note

You don't need to rebuild the solution. Instead, build the project under the `/Games/` folder for your app. Otherwise, Visual Studio will rebuild the entire UE5 project, which can take up to an hour.

Package your game server for hosting

With your game server code now integrated with the minimum required server SDK functionality, you're ready to package your game server build using the Unreal Editor.

To package the game server build

1. Open the game project in the Unreal Editor.
2. Follow the Unreal Editor steps to package your game server:
 - Choose your target platform (Windows or Linux).
 - Select your server build target (*[your application name]*Server).

The packaging process generates your game server executable: *[your application name]*Server.exe.

3. Prepare your game server build for deployment to hosting resources. The build should include the following files:
 - Your game server executable
 - If you're using Unreal Engine version 5.5 or older, include the following files for Windows builds. You can find them in your source-built version of the Unreal Engine:
 - VC_redist.x64.exe (UnrealEngine\Engine\Source\Programs\PrereqInstaller\Resources\VCRedist\)
 - UEPrereqSetup_x64.exe or UE5PrereqSetup_x64.exe (UnrealEngine\Engine\Extras\Redist\en-us\)
 - All other required dependencies for your game server.
 - OpenSSL libraries, if needed. You can skip this step if your game server is integrated with Amazon GameLift Servers server SDK version 5.3 or later. [The latest server SDK version is available here.](#)

OpenSSL libraries required for game servers created with server SDK for Unreal, versions 5.2 and earlier

You must include the same version of OpenSSL libraries that were used when packaging the game server in Unreal. These libraries are located in your game engine source. The location varies depending on your development environment:

On Windows:

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

On Linux:

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

Copy the OpenSSL libraries to your game build package in the same directory as the game server executable file.

Next steps

Now that you've prepared a game server build with the minimum required functionality for hosting with Amazon GameLift Servers, consider these potential next steps:

- Deploy your integrated game server for and testing and development. With an Anywhere fleet, you can set up your local machine as a hosting resource and use it to test your game server and game client connections. For cloud-based hosting, deploy your game server to a managed EC2 or managed container fleet. See these topics for guidance:
 - [Set up for iterative development with Amazon GameLift Servers Anywhere](#)
 - [Amazon GameLift Servers Anywhere fleets](#)
 - [Amazon GameLift Servers managed EC2 fleets](#)
 - [Amazon GameLift Servers managed container fleets](#)
- Customize your game server integration by adding optional features. For example, you might want to add player sessions with unique player IDs, set up matchmaking backfill, or manage

game server access to your other AWS resources (such as a database or content storage service). See these topics for guidance:

- [Add Amazon GameLift Servers to your game server with the server SDK](#)
- [C++ \(Unreal\) server SDK 5.x for Amazon GameLift Servers -- Actions](#)
- Customize your game client component to request game sessions, receive connection information, and connect directly to a game server to play a game. See these topics for guidance:
 - [Integrate Amazon GameLift Servers game client functionality](#)

Integrate Amazon GameLift Servers into a Unity project

Learn how to integrate the Amazon GameLift Servers SDK for Unity into your game projects to access the full server SDK feature set.

Tip

For faster deployment, try the Amazon GameLift Servers standalone plugin for Unity. It provides guided UI workflows to quickly deploy your game server with minimal setup, so you can try out your game components in action. See [Amazon GameLift Servers plugin for Unity \(server SDK 5.x\)](#).

Additional resources:

- [C# server SDK 5.x for Amazon GameLift Servers -- Actions](#)
- [the section called "Get development tools"](#)

Install the server SDK for Unity

Get the open-source Amazon GameLift Servers for Unity from [GitHub](#). The repository's readme files contain prerequisites and installation instructions.

Set up an Amazon GameLift Servers Anywhere fleet for testing

You can set up your development workstation as an Amazon GameLift Servers Anywhere hosting fleet to iteratively test your Amazon GameLift Servers integration. With this setup, you can start game server processes on your workstation, send player join or matchmaking requests to Amazon GameLift Servers to start game sessions, and connect clients to the new game sessions. With your

own workstation set up as a hosting server, you can monitor all aspects of your game integration with Amazon GameLift Servers.

For instructions on setting up your workstation, see [Set up local testing with Amazon GameLift Servers Anywhere](#) to complete the following steps:

1. Create a custom location for your workstation.
2. Create an Amazon GameLift Servers Anywhere fleet with your new custom location. If successful, this request returns a fleet ID. Make a note of this value, as you'll need it later.
3. Register your workstation as a compute in the new Anywhere fleet. Provide a unique compute name and specify the IP address for your workstation. If successful, this request returns a service SDK endpoint, in the form of a WebSocket URL. Make a note of this value, as you'll need it later.
4. Generate an authentication token for your workstation compute. This short-lived authentication includes the token and an expiration date. Your game server uses it to authenticate communication with the Amazon GameLift Servers service. Store the authentication on your workstation compute so that your running game server processes can access it.

Add Amazon GameLift Servers server code to your Unity project

Your game server communicates with the Amazon GameLift Servers service to receive instructions and report ongoing status. To accomplish this, you add game server code that uses the Amazon GameLift Servers server SDK.

The provided code example illustrates the basic required integration elements. It uses a `MonoBehavior` to illustrate a simple game server initialization with Amazon GameLift Servers. The example assumes that the game server runs on an Amazon GameLift Servers Anywhere fleet for testing. It includes code to:

- Initialize an Amazon GameLift Servers API client. The sample uses the version of `InitSDK()` with server parameters for your Anywhere fleet and compute. Use the WebSocket URL, fleet ID, compute name (host ID), and authentication token, as defined in the previous topic [Set up an Amazon GameLift Servers Anywhere fleet for testing](#).
- Implement callback functions to respond to requests from the Amazon GameLift Servers service, including `OnStartGameSession`, `OnProcessTerminate`, and `onHealthCheck`.
- Call `ProcessReady()` with a designated port to notify the Amazon GameLift Servers service when the process is ready to host game sessions.

The sample code provided establishes communication with the Amazon GameLift Servers service. It also implements a set of callback functions that respond to requests from the Amazon GameLift Servers service. For more information on each function and what the code does, see [Initialize the server process](#). For more information on the SDK actions and data types used in this code, read [C# server SDK 5.x for Amazon GameLift Servers -- Actions](#).

The sample code shows how to add the required functionality, as described in [Add Amazon GameLift Servers to your game server](#). For more information on server SDK actions, see the [C# server SDK 5.x for Amazon GameLift Servers -- Actions](#).

Integration sample code

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

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

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

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

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

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

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

        //Server parameters are required for an Amazon GameLift Servers Anywhere fleet.
        //They are not required for an Amazon GameLift Servers managed EC2 fleet.
```

```

ServerParameters serverParameters = new ServerParameters(
    webSocketUrl,
    processId,
    hostId,
    fleetId,
    authToken);

//InitSDK establishes a local connection with an Amazon GameLift Servers agent
//to enable further communication.
var initSDKOutcome = GameLiftServerAPI.InitSDK(serverParameters);
if (initSDKOutcome.Success)
{
    //Implement callback functions
    ProcessParameters processParameters = new ProcessParameters(
        //Implement OnStartGameSession callback
        (gameSession) => {
            //Amazon GameLift Servers sends a game session activation request
to the game server
            //with game session object containing game properties and other
settings.
            //Here is where a game server takes action based on the game
session object.
            //When the game server is ready to receive incoming player
connections,
            //it invokes the server SDK call ActivateGameSession().
            GameLiftServerAPI.ActivateGameSession();
        },
        (updateGameSession) => {
            //Amazon GameLift Servers sends a request when a game session is
updated (such as for
            //FlexMatch backfill) with an updated game session object.
            //The game server can examine matchmakerData and handle new
incoming players.
            //updateReason explains the purpose of the update.
        },
        () => {
            //Implement callback function OnProcessTerminate
            //Amazon GameLift Servers invokes this callback before shutting
down the instance hosting this game server.
            //It gives the game server a chance to save its state, communicate
with services, etc.,
            //and initiate shut down. When the game server is ready to shut
down, it invokes the

```

```

        //server SDK call ProcessEnding() to tell Amazon GameLift Servers
it is shutting down.
        GameLiftServerAPI.ProcessEnding();
    },
    () => {
        //Implement callback function OnHealthCheck
        //Amazon GameLift Servers invokes this callback approximately every
60 seconds.

        //A game server might want to check the health of dependencies,
etc.

        //Then it returns health status true if healthy, false otherwise.
        //The game server must respond within 60 seconds, or Amazon
GameLift Servers records 'false'.
        //In this example, the game server always reports healthy.
        return true;
    },
    //The game server gets ready to report that it is ready to host game
sessions
    //and that it will listen on port 7777 for incoming player connections.
    listeningPort,
    new LogParameters(new List<string>()
    {
        //Here, the game server tells Amazon GameLift Servers where to find
game session log files.
        //At the end of a game session, Amazon GameLift Servers uploads
everything in the specified
        //location and stores it in the cloud for access later.
        "/local/game/logs/myserver.log"
    }));

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

```



```
        else
        {
            print("InitSDK failure : " + initSDKOutcome.Error.ToString());
        }
    }

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

Next steps

Now that you've prepared a game server build with the minimum required functionality for hosting with Amazon GameLift Servers, consider these potential next steps:

- Deploy your integrated game server for and testing and development. With an Anywhere fleet, you can set up your local machine as a hosting resource and use it to test your game server and game client connections. For cloud-based hosting, deploy your game server to a managed EC2 or managed container fleet. See these topics for guidance:
 - [Set up for iterative development with Amazon GameLift Servers Anywhere](#)
 - [Amazon GameLift Servers Anywhere fleets](#)
 - [Amazon GameLift Servers managed EC2 fleets](#)
 - [Amazon GameLift Servers managed container fleets](#)

- Customize your game server integration by adding optional features. For example, you might want to add player sessions with unique player IDs, set up matchmaking backfill, or manage game server access to your other AWS resources (such as a database or content storage service). See these topics for guidance:
 - [Add Amazon GameLift Servers to your game server with the server SDK](#)
 - [C++ \(Unreal\) server SDK 5.x for Amazon GameLift Servers -- Actions](#)
- Customize your game client component to request game sessions, receive connection information, and connect directly to a game server to play a game. See these topics for guidance:
 - [Integrate Amazon GameLift Servers game client functionality](#)

Package a game server build for deployment

Get your game server software ready for deployment on your hosting resources. Once deployed, the software is installed on each hosting resource, and then one or more game server process is launched and readied to host game sessions for players.

Getting a game server build ready varies depending on the type of Amazon GameLift Servers hosting options you're using. All game server builds must be integrated with the server SDK for Amazon GameLift Servers, as described in [Integrate a game server with Amazon GameLift Servers](#).

The topics in this section provide guidance on how to get your software ready for deployment to the following scenarios.

- For managed EC2 hosting, package your server software and upload it to Amazon GameLift Servers for deployment.
- For managed containers hosting, build a container image with your server software and store it in Amazon Elastic Container Registry for deployment.
- For hosting with Amazon GameLift Servers Anywhere, package your server software as needed for installation on your own hosting resources.

Note

If you're deploying an Amazon GameLift Servers Realtime configured script, see [Upload a script for Amazon GameLift Servers Realtime](#).

Topics

- [Create a game server build for Amazon GameLift Servers](#)
- [Build a container image for Amazon GameLift Servers](#)

Create a game server build for Amazon GameLift Servers

After you integrate your game server with Amazon GameLift Servers (see [Prepare a game for hosting with Amazon GameLift Servers](#)), install the game server software onto your compute resources for hosting. This process varies depending on the type of Amazon GameLift Servers hosting you're using.

Deploy for managed hosting

If you're using Amazon GameLift Servers managed EC2 hosting, you must package your game server software and upload it to Amazon GameLift Servers. When you create a managed fleet, Amazon GameLift Servers automatically deploys it to each fleet instance.

The topics in this section describe how to package your build files for uploading, create an optional build install script, and then upload the files using the [AWS Command Line Interface \(AWS CLI\)](#) or AWS SDK.

Deploy for Anywhere hosting

If you're using Amazon GameLift Servers Anywhere fleets for self-managed hosting, it's your responsibility to install the game server software onto each compute in a fleet and keep it updated.

When an integrated game server process starts running, it automatically initializes and establishes communication with the Amazon GameLift Servers service. The server process starts game sessions from prompts by Amazon GameLift Servers and reports activity back to the service.

Topics

- [Package your game build files](#)
- [Add a build install script](#)
- [Create an Amazon GameLift Servers build resource for managed hosting](#)

Package your game build files

Before uploading your configured game server to Amazon GameLift Servers, package the game build files into a build directory. This process is a requirement when hosting with an EC2 managed

fleet, and is a best practice when hosting with an Anywhere fleet. The build directory should include all the components that are required to run your game servers and host game sessions. This might include the following:

- **Game server binaries** – The binary files required to run the game server. A build can include binaries for multiple game servers built to run on the same platform. For a list of supported platforms, see [Get Amazon GameLift Servers development tools](#).
- **Dependencies** – Any dependent files that your game server executables require to run. Examples include assets, configuration files, and dependent libraries.

 Note

For game builds created with the server SDK for Amazon GameLift Servers for C++ (including those created with the Unreal plugin), include the OpenSSL DLL for the same version of OpenSSL that you built the server SDK with. See the server SDK README file for more details.

- **Install script** (Optional) – A script file to handle tasks that install your game build on Amazon GameLift Servers hosting servers. Place this file at the root of the build directory. Amazon GameLift Servers runs the install script as part of fleet creation.

You can set up any application in your build, including your install script, to access your resources securely on other AWS services. For information about ways to do this, see [Connect your Amazon GameLift Servers hosted game server to other AWS resources](#).

After you've packaged your build files, make sure that your game server can run on a clean installation of your target OS to verify that all required dependencies are included and that your install script is accurate.

Add a build install script

Create an install script for the operating system (OS) of your game build:

- Windows: Create a batch file named `install.bat`.
- Linux: Create a shell script file named `install.sh`.

Amazon GameLift Servers runs the install script during fleet creation to establish an Amazon Machine Image (AMI) that is used on all future fleet instances. The script is not run at the start of each new instance.

When creating an install script, keep in mind the following:

- The script can't take any user input.
- Amazon GameLift Servers installs the build and recreates the file directories in your build package on a hosting server in the following locations:
 - Windows fleets: C:\game
 - Linux fleets: /local/game
- During the installation process for Linux fleets, the run-as user has limited access to the instance file structure. This user has full rights to the directory where your build files are installed. If your install script performs actions that require administrator permissions, then specify admin access using **sudo**. The run-as user for Windows fleets has administrator permissions by default. Permission failures related to the install script generate an event message that indicates a problem with the script.
- On Linux, Amazon GameLift Servers supports common shell interpreter languages such as bash. Add a shebang (such as `#!/bin/bash`) to the top of your install script. To verify support for your preferred shell commands, remotely access an active Linux instance and open a shell prompt. For more information, see [Connect to fleet instances](#).
- The install script can't rely on a VPC peering connection. A VPC peering connection isn't available until after Amazon GameLift Servers installs the build on fleet instances.
- The install script might run more than once as part of fleet creation. As a best practice, make sure the script is idempotent and robust to being run multiple times.

Example Windows install bash file

This example `install.bat` file installs Visual C++ runtime components required for the game server and writes the results to a log file. The script includes the component file in the build package at the root.

```
vcredist_x64.exe /install /quiet /norestart /log c:\game\vcredist_2013_x64.log
```

Example Linux install shell script

This example `install.sh` file uses bash in the install script and writes results to a log file.

```
#!/bin/bash
echo 'Hello World' > install.log
```

This example `install.sh` file shows how you can use the Amazon CloudWatch agent to collect system-level and custom metrics, and handle log rotation. Because Amazon GameLift Servers runs in a service VPC, you must grant Amazon GameLift Servers permissions to assume an AWS Identity and Access Management (IAM) role on your behalf. To allow Amazon GameLift Servers to assume a role, create a role that includes the AWS managed policy `CloudWatchAgentAdminPolicy`, and use that role when you create a fleet.

```
sudo yum install -y amazon-cloudwatch-agent
sudo yum install -y https://dl.fedoraproject.org/pub/epel/epel-release-
latest-7.noarch.rpm
sudo yum install -y collectd
cat <<'EOF' > /tmp/config.json
{
  "agent": {
    "metrics_collection_interval": 60,
    "run_as_user": "root",
    "credentials": {
      "role_arn": "arn:aws:iam::account#:role/rolename"
    }
  },
  "logs": {
    "logs_collected": {
      "files": {
        "collect_list": [
          {
            "file_path": "/tmp/log",
            "log_group_name": "gllog",
            "log_stream_name": "{instance_id}"
          }
        ]
      }
    }
  },
  "metrics": {
    "namespace": "GL_Metric",
    "append_dimensions": {
      "ImageId": "${aws:ImageId}",
      "InstanceId": "${aws:InstanceId}",
      "InstanceType": "${aws:InstanceType}"
    }
  }
}
```

```
    },
    "metrics_collected": {
        // Configure metrics you want to collect.
        // For more information, see Manually create or edit the CloudWatch agent configuration file.
    }
}
EOF
sudo mv /tmp/config.json /opt/aws/amazon-cloudwatch-agent/bin/config.json
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c file:/opt/aws/amazon-cloudwatch-agent/bin/config.json
sudo systemctl enable amazon-cloudwatch-agent.service
```

Create an Amazon GameLift Servers build resource for managed hosting

When creating a build and uploading your files, you have a couple of options:

- [Create a build from a file directory](#). This is the simplest and most commonly used option.
- [Create a build with files in Amazon Simple Storage Service \(Amazon S3\)](#). With this option, you can manage your build versions in Amazon S3.

With both methods, Amazon GameLift Servers creates a new build resource with a unique build ID and other metadata. The build starts in the **Initialized** status. After Amazon GameLift Servers acquires the game server files, the build moves to **Ready** status.

When the build is ready, you can deploy it to a new Amazon GameLift Servers fleet. For more information, see [Create an Amazon GameLift Servers managed EC2 fleet](#). When Amazon GameLift Servers sets up the new fleet, it downloads the build files to each fleet instance and installs the build files.

Create a build from a file directory

To create a game build stored in any location, including a local directory, use the [upload-build](#) AWS CLI command. This command creates a new build record in Amazon GameLift Servers and uploads files from a location that you specify.

Send an upload request. In a command line window, enter the following **upload-build** command and parameters.

```
aws gamelift upload-build \
```

```
--name user-defined name of build \  
--operating-system supported OS \  
--server-sdk-version server SDK for Amazon GameLift Servers version \  
--build-root build path \  
--build-version user-defined build number \  
--region region name
```

- **operating-system** – The game server build's runtime environment. You must specify an OS value. You can't update this later.
- **server-sdk-version** – The version of the Amazon GameLift Servers server SDK that your game server is integrated with. If you don't provide a value, Amazon GameLift Servers uses the default value 4.0.2. If you specify an incorrect server SDK version, the game server build might fail when calling `InitSdk` to establish a connection to the Amazon GameLift Servers service.
- **build-root** – The directory path of your build files.
- **name** – A descriptive name for the new build.
- **build-version** – The version details for the build files.
- **region** – The AWS Region where you want to create your build. Create the build in the Region where you plan to deploy fleets. If you're deploying your game in multiple Regions, create a build in each Region.

Note

View your current default Region using the [aws configure get region](#). To change your default Region, use the [aws configure set region *region name*](#) command.

Examples

```
aws gamelift upload-build \  
  --operating-system AMAZON_LINUX_2023 \  
  --server-sdk-version "5.0.0" \  
  --build-root "~/mygame" \  
  --name "My Game Nightly Build" \  
  --build-version "build 255" \  
  --region us-west-2
```

```
aws gamelift upload-build \  
  --operating-system WINDOWS_2022 \  
  --name "My Game Nightly Build" \  
  --build-version "build 255" \  
  --region us-west-2
```



```
--server-sdk-version "5.0.0" \  
--build-root "C:\mygame" \  
--name "My Game Nightly Build" \  
--build-version "build 255" \  
--region us-west-2
```

In response to your upload request, Amazon GameLift Servers provides upload progress. On a successful upload, Amazon GameLift Servers returns the new build record ID. Upload time depends on the size of your game files and the connection speed.

Create a build with files in Amazon S3

You can store your build files in Amazon S3 and upload them to Amazon GameLift Servers from there. When you create your build, you specify the S3 bucket location, and Amazon GameLift Servers retrieves the build files directly from Amazon S3.

To create a build resource

1. **Store your build files in Amazon S3.** Create a .zip file containing the packaged build files and upload it to an S3 bucket in your AWS account. Take note of the bucket label and the file name, you'll need these when creating an Amazon GameLift Servers build.
2. **Give Amazon GameLift Servers access to your build files.** Create an IAM role by following the instructions in [Access a game build file in Amazon S3](#). After you've created the role, take note of the new role's Amazon Resource Name (ARN), you'll need this when creating a build.
3. **Create a build.** Use the Amazon GameLift Servers console or the AWS CLI to create a new build record. You must have the `PassRole` permission, as described in [IAM permission examples for Amazon GameLift Servers](#).

Console

1. In the [Amazon GameLift Servers console](#), in the navigation pane, choose **Hosting, Builds**.
2. On the **Builds** page, choose **Create build**.
3. On the **Create build** page, under **Build settings**, do the following:
 - a. For **Name**, enter a script name.
 - b. For **Version**, enter a version. Because you can update the content of a build, version data can help you track updates.

- c. For **Operating system (OS)**, choose the OS of your game server build. You can't update this value later.
 - d. For **Game server build**, enter the **S3 URI** of the build object that you uploaded to Amazon S3, and choose the **Object version**. If you don't remember the Amazon S3 URI and object version, choose **Browse S3** and search for the build object.
 - e. For **IAM role**, choose the role that you created that gives Amazon GameLift Servers access to your S3 bucket and build object.
4. (Optional) Under **Tags**, add tags to the build by entering **Key** and **Value** pairs.
 5. Choose **Create**.

Amazon GameLift Servers assigns an ID to the new build and uploads the designated .zip file. You can view the new build, including the status, on the **Builds** page.

AWS CLI

To define the new build and upload your server build files, use the [create-build](#) command.

1. Open a command line window and switch to a directory where you can use the AWS CLI.
2. Enter the following **create-build** command:

```
aws gamelift create-build \  
  --name user-defined name of build \  
  --server-sdk-version server SDK for Amazon GameLift Servers version \  
  --operating-system supported OS \  
  --build-version user-defined build number \  
  --storage-location "Bucket"=S3 bucket label,"Key"=Build .zip file \  
  name,"RoleArn"=Access role ARN} \  
  --region region name
```

- **name** – A descriptive name for the new build.
- **server-sdk-version** – The version of the server SDK for Amazon GameLift Servers you used to integrate your game server with Amazon GameLift Servers. If you don't provide a value, Amazon GameLift Servers uses the default value 4.0.2.
- **operating-system** – The game server build's runtime environment. You must specify an OS value. You can't update this later.
- **build-version** – The version details for the build files. This information can be useful because each new version of your game server requires a new build resource.

- **storage-location**
 - **Bucket** – The name of the S3 bucket that contains your build. For example, "my_build_files".
 - **Key** – The name of the .zip file that contains your build files. For example, "my_game_build_7.0.1, 7.0.2".
 - **RoleARN** – The ARN assigned to the IAM role that you created. For example, "arn:aws:iam::111122223333:role/GameLiftAccess". For an example policy, see [Access a game build file in Amazon S3](#).
- **region** – Create the build in the AWS Region where you plan to deploy fleets. If you're deploying your game in multiple Regions, create a build in each Region.

 Note

We recommend checking your current default Region using the [configure get](#) command. To change your default Region, use the [configure set](#) command.

Example

```
aws gamelift create-build \  
  --operating-system WINDOWS_2022 \  
  --storage-location  
  "Bucket"="my_game_build_files", "Key"="mygame_build_101.zip", "RoleArn"="arn:aws:iam::111122223333:role/GameLiftAccess"  
gamelift" \  
  --name "My Game Nightly Build" \  
  --build-version "build 101" \  
  --region us-west-2
```

3. To view the new build, use the [describe-build](#) command.

Build a container image for Amazon GameLift Servers

This topic describes how to create a container image with your game server software to use with Amazon GameLift Servers. A game server container image includes the game server executable and any dependencies it needs to run. Game server container images are used with an Amazon GameLift Servers managed containers hosting solution. For details on building the complete solution, see:

- [Development roadmap for hosting with Amazon GameLift Servers managed containers](#)
- [How containers work in Amazon GameLift Servers](#)

Complete the following tasks to get your game server container image ready for deployment to an Amazon GameLift Servers container fleet. Before starting these tasks, finish integrating your game server code with the Amazon GameLift Servers server SDK.

Topics

- [Create a game server container image](#)
- [Push a container image to Amazon ECR](#)

Create a game server container image

Follow these instructions while working on a Linux-based platform or using Windows Subsystem for Linux (WSL) with Docker installed.

To create a game server container image

1. Prepare a working directory with your game server software. On a local machine, create a working directory to organize the files for your game server container. Your container image uses this file structure when deploying the container to Amazon GameLift Servers resources for hosting. For example:

```
[~/]$ mkdir -p work/glc/gamebuild && cd work && find .  
.  
./glc  
./glc/gamebuild
```

Note

If you're trying out this feature and don't have a working game server build yet, try our sample game server, [SimpleServer](#), which is available on GitHub.

2. Create a new Dockerfile using the template provided.
3. Follow the instructions in the Dockerfile template to update it for your own use.
 - Update the base image as needed.

- Set environment variables for your game server build.
4. Build the container image. Run the `docker build`, specifying your own repository name. For example:

```
[~/work/glc]$ docker build -t <local repository name>:<optional tag> .
```

You can view your repositories and images IDs using the `docker images` command, as illustrated in this example:

Dockerfile template for a game server container image

This template contains the minimum instructions that a game server container needs to usable in an Amazon GameLift Servers fleet. Modify the content as needed for your game server.

```
# Base image
# -----
# Add the base image that you want to use,
# Make sure to use an image with the same architecture as the
# Instance type you are planning to use on your fleets.
FROM public.ecr.aws/amazonlinux/amazonlinux
#
# Game build directory
# -----
# Add your gamebuild directory to the env variable below.
# The game build provided here needs to be integrated with server sdk for Amazon
# GameLift Servers.
ENV GAME_BUILD_DIRECTORY="<>ADD_GAME_BUILD_DIRECTORY>" \
#
# Game executable and launch parameters
# -----
# Add the relative path to your executable in the 'GAME_EXECUTABLE' env variable
# below.
# The game build provided over here needs to be integrated with server sdk for Amazon
# GameLift Servers.
# This template assumes that the executable path is relative to the game build
# directory.
# Add any launch parameters to pass into your executable in the 'LAUNCH_PARAMS' env
# variable below.
# Add 'HOME_DIR' to identify where the game executable and logs exist.
GAME_EXECUTABLE="<>ADD NAME OF EXECUTABLE WITHIN THE GAME DIRECTORY>" \
LAUNCH_PARAMS=<ADD LAUNCH PARAMETERS> \
```

```
HOME_DIR="/local/game" \  
  
# Install dependencies as necessary  
RUN yum install -y shadow-utils  
  
RUN mkdir -p $HOME_DIR  
COPY ./$GAME_BUILD_DIRECTORY/ $HOME_DIR  
  
# Change directory to home  
WORKDIR $HOME_DIR  
  
# Set up for 'gamelift' user  
RUN useradd -m gamelift && \  
    echo "gamelift ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers && \  
    chown -R gamelift:gamelift $HOME_DIR  
  
# Add permissions to game build  
RUN chmod +x ./$GAME_EXECUTABLE  
  
USER gamelift  
  
# Check directory before starting the container  
RUN ls -lhrt .  
  
# Check path before starting the container  
RUN echo $PATH  
  
# Start the game build  
ENTRYPOINT ["/bin/sh", "-c", ".$GAME_EXECUTABLE", "$LAUNCH_PARAMS"]
```

Push a container image to Amazon ECR

After you've created a container image for deployment to Amazon GameLift Servers, store the image in a public or private repository in Amazon ECR. This repository is assigned a URI value, which Amazon GameLift Servers uses to take a snapshot of the image for deployment to a container fleet.

Note

If you don't yet have an Amazon ECR private repository, then [create one](#).

To push your container image to Amazon ECR

1. Get your Amazon ECR credentials. Before pushing a container image to Amazon ECR, first acquire your AWS credentials in temporary form and provide them to Docker. Docker needs these credentials to log in.

```
[~/work/glc]$ aws ecr get-login-password --region us-west-2 | docker login --  
username AWS --password-stdin aws_account_id.dkr.ecr.us-west-2.amazonaws.com  
WARNING! Your password will be stored unencrypted in  
/home/user-name/.docker/config.json.  
Configure a credential helper to remove this warning.  
See https://docs.docker.com/engine/reference/commandline/login/#credentials-store  
  
Login Succeeded
```

2. Copy the URI of the [Amazon ECR private repository](#) you want to use.
3. Apply an Amazon ECR tag to your container image.

Example

```
[~/work/glc]$ docker tag <IMAGE ID from above> <Amazon ECR private repository  
URI>:<optional tag>
```

4. Push your container image to Amazon ECR

Example

```
[~/work/glc]$ docker image push <Amazon ECR private repository URI>
```

Game client/server interactions with Amazon GameLift Servers

The components in your Amazon GameLift Servers hosting solution interact with each other in specific ways to run game sessions in response to player demand. This topic describes how components communicate with each other when your game server is hosted on Amazon GameLift Servers managed EC2 fleets, self-managed Amazon GameLift Servers Anywhere fleets, or a hybrid solution.

Hosting solution components include a game server, the Amazon GameLift Servers service, a client-side backend service, and a game client. The game server uses the [Amazon GameLift Servers server SDK](#) to interact with the Amazon GameLift Servers service. The backend service uses the

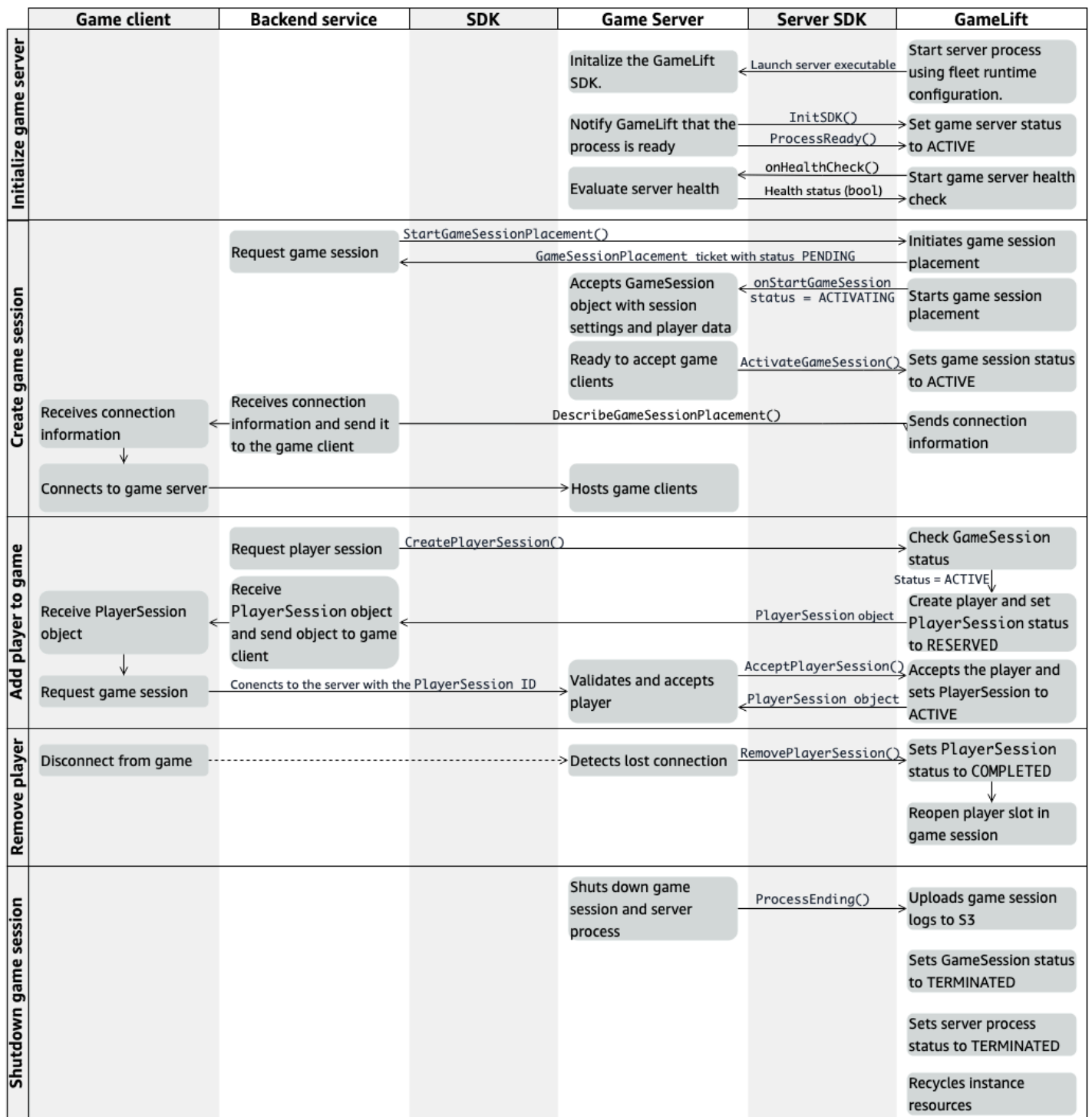
[Amazon GameLift Servers service API](#) (part of the AWS SDK) to interact with the service on behalf of the game client. When joining a game session, the game client connects directly to a game session using the game session's unique IP address and port number.

Topics

- [Interactions diagram](#)
- [Interaction behaviors](#)

Interactions diagram

The following diagram illustrates how your game hosting components interact so that the Amazon GameLift Servers service can track the status of game server availability and start game sessions in response to player demands.



Interaction behaviors

The following sections describe the sequence of events in each of the key interactions.

Initializing a game server process

On startup, a game server process establishes communication with the Amazon GameLift Servers service and reports its status as ready to host a game session.

1. A new process of the game server executable starts running on a hosting resource.
2. The game server process calls the following server SDK operations in sequence:
 - a. `InitSDK()` to initialize the server SDK, authenticate the server process, and establish communication with the Amazon GameLift Servers service.
 - b. `ProcessReady()` to communicate readiness to host a game session. This call also reports the process's connection information, which game clients use to connect to the game session, and other information.

The server process then waits for prompts from the Amazon GameLift Servers service.

3. Amazon GameLift Servers updates the status of the server process to `ACTIVE` and available to host a new game session.
4. Amazon GameLift Servers begins periodically calling the `onHealthCheck` callback to request a health status from the server processes. These calls continue while the server process remains in active status. The server process must respond healthy or not healthy within one minute. If the server process responds not healthy or doesn't respond, at some point the Amazon GameLift Servers service changes the server process's active status and stops sending requests to start game session.

Creating a game session

The Amazon GameLift Servers service starts a new game session in response to a request from a player to play the game.

1. A player using the game client asks to join a game session. Depending on how your game handles the player join process, the game client sends a request to the backend service.
2. If the player join process requires starting a new game session, the backend service sends a request for a new game session to the Amazon GameLift Servers service. This request calls the service API operation `StartGameSessionPlacement()`. (As an alternative, the backend service might call `StartMatchmaking()`, or `CreateGameSession()`.)

3. The Amazon GameLift Servers service responds by creating a new `GameSessionPlacement` ticket with status `PENDING`. It returns ticket information to the backend service, so that it can track the placement ticket status and determine when the game session is ready for players. For more information, see [Set up event notification for game session placement](#).
4. The Amazon GameLift Servers service starts the game session placement process. It identifies which fleets to look at and searches those fleets for an active server process that's not hosting a game session. On locating an available server process, the Amazon GameLift Servers service does the following:
 - a. Creates a `GameSession` object with the game session settings and player data from the placement request, and sets the status to `ACTIVATING`.
 - b. Prompts the server process to start a game session. The service invokes the server process's `onStartGameSession` callback and passes the `GameSession` object.
 - c. Changes the server process's number of game sessions to 1.
5. The server process runs its `onStartGameSession` callback function. When the server process is ready to accept player connections, it calls the server SDK operation `ActivateGameSession()` and waits for player connections.
6. The Amazon GameLift Servers service updates the `GameSession` object with connection information for the server process (as reported in the call to `ProcessReady()`) and sets the game session status to `ACTIVE`. It also updates the `GameSessionPlacement` ticket status to `FULFILLED`.
7. The backend service calls `DescribeGameSessionPlacement()` to check ticket status and get game session information. When the game session is active, the backend service notifies the game client and passes the game session connection information.
8. The game client uses the connection information to connect directly to the game server process and join the game session.

Adding a player to a game

A game can optionally use player sessions to track player connections to game sessions. Player sessions can be created individually or as part of a game session placement request.

1. The backend service calls the service API operation `CreatePlayerSession()` with a game session ID.

2. The Amazon GameLift Servers service checks the game session status (must be ACTIVE), and looks for an open player slot in the game session. If a slot is available, then the service does the following:
 - a. Creates a new `PlayerSession` object and sets the status to RESERVED.
 - b. Responds to the backend service request with player session information.
3. The backend service passes player session information to the game client along with game session connection information.
4. The game client uses the connection information and player session ID to connect directly to the game server process and ask to join the game session.
5. In response to a game client join attempt, the game server process calls the service API operation `AcceptPlayerSession()` to validate the player session ID. The server process either accepts or rejects the connection.
6. The Amazon GameLift Servers service does one of the following:
 - a. If the connection is accepted, then Amazon GameLift Servers sets the `PlayerSession` status to ACTIVE and passes the `PlayerSession` to the game server process.
 - b. If the game server process doesn't call `AcceptPlayerSession()` for the player session ID within a certain time period after the original `CreatePlayerSession()` request, the Amazon GameLift Servers service changes the `PlayerSession` status to TIMEDOUT and reopens the player slot in the game session.

Removing a player

For games that use player sessions, the game server process notifies the Amazon GameLift Servers service when a player disconnects. The service uses this information to track the status of player slots in a game session and can allow new players to use open slots.

1. A player disconnects from the game session.
2. The game server process detects the lost connection and calls the server SDK operation `RemovePlayerSession()`.
3. The Amazon GameLift Servers service changes the player session status to COMPLETED and reopens the player slot in the game session.

Shutting down the game session

At the end of a game session, or when shutting down the game session, the server process notifies the Amazon GameLift Servers service of game session status.

1. The game server process ends the game session and initiates process shut down by calling the server SDK operation `ProcessEnding()`.
2. The game server process calls `Destroy()` to free the server SDK from memory. This step is required for [Telemetry metrics](#) to prevent it from reporting normal process exits as crashes.
3. The Amazon GameLift Servers service does the following:
 - a. Uploads game session logs to Amazon Simple Storage Service (Amazon S3).
 - b. Changes the game session status to `TERMINATED`.
 - c. Changes the server process status to `TERMINATED`.
 - d. Depending on how the hosting solution is designed, the newly available hosting resources are allocated to run a new game server process.

Build a backend service for Amazon GameLift Servers

We recommend that you implement a game client service that authenticates your players and communicates with the Amazon GameLift Servers API. By implementing a custom game client service, you can:

- Customize authentication for your players.
- Control how Amazon GameLift Servers groups players for new game sessions or to add to existing game sessions.
- Collect information from your own resources to provide game sesplayer attributes such as skill rating for matchmaking instead of trusting the client.

Using a game client service also reduces security risks introduced by game clients interacting directly with your Amazon GameLift Servers API.

Authenticating your players

You can use Amazon Cognito and player session IDs to authenticate your game clients. To manage the lifecycle and properties of your player identities, use Amazon Cognito user pools.

If you prefer, build a custom identity solution and host it on AWS. You can also use Lambda authorizers for custom authorization logic with API Gateway.

Additional resources:

- [Using identity pools \(federated identities\)](#) (Amazon Cognito Developer Guide)
- [Getting started with user pools](#) (Amazon Cognito Developer Guide)
- [How to Set Up Player Authentication with Amazon Cognito](#) (AWS for Games Blog)

Integrate Amazon GameLift Servers game client functionality

Integrate Amazon GameLift Servers game hosting functionality into any solution components that need to communicate with the Amazon GameLift Servers service, such as to get game session information or create new game sessions. In most cases, this functionality is built into the backend service component. Add functionality for core tasks, including:

- Request information and status of active game sessions.
- Reserve a slot for a new player in an existing game session.
- Create a new game session for a group of players.
- Request matchmaking for one or more players.
- Provide updated data for existing game sessions.

See [Game client/server interactions with Amazon GameLift Servers](#) for additional details on how game hosting components interact through the Amazon GameLift Servers SDKs.

Set up the Amazon GameLift Servers API

Add the AWS SDK for C++ with Amazon GameLift Servers to a project.

Add code to initialize an Amazon GameLift Servers client and store key settings. This code must run before any code dependent on Amazon GameLift Servers.

1. Set up a client configuration. Use the default client configuration or create a custom client configuration object. For more information, see [AWS::Client::ClientConfiguration](#) (C++) or [AmazonGameLiftConfig](#) (C#).

A client configuration specifies a target region and endpoint to use when contacting Amazon GameLift Servers. Region identifies the set of deployed resources (fleets, queues, and

matchmakers) to use. The default client configuration sets location to the US East (N. Virginia) Region. To use any other Region, create a custom configuration.

2. Initialize an Amazon GameLift Servers client. Use [Aws::GameLift::GameLiftClient\(\)](#) (C++) or [AmazonGameLiftClient\(\)](#) (C#) with a default client configuration or a custom client configuration.
3. If you're using player sessions, add a mechanism to generate a unique identifier for each player. For more information, see [Generate player IDs](#).
4. Collect and store the following information:
 - **Target fleet** – If you're manually creating game sessions on a specific fleet, provide a fleet ID or an alias ID that points to the target fleet. If the fleet is multi-location, specify a fleet location. As a best practice, use fleet aliases so that you can switch players from one fleet to another without having to update your backend service.
 - **Target queue** – For games that use multi-fleet queues to place new game sessions, specify the name of the queue to send placement requests to.
 - **AWS credentials** – All calls to Amazon GameLift Servers must provide credentials for the AWS account that hosts the game. You can acquire these credentials by creating a player user, as described in [Set up programmatic access for your game](#). Depending on how you manage access for the player user, do the following:
 - If you use a role to manage player user permissions, add code to assume the role before calling an Amazon GameLift Servers API. The request to assume the role returns a set of temporary security credentials. For more information, see [Switching to an IAM role \(AWS API\)](#) in the *IAM User Guide*.
 - If you have long-term security credentials, configure your code to locate and use stored credentials. See [Authenticate using long-term credentials](#) in the *AWS SDKs and Tools Reference Guide*. For information on storing credentials, see the AWS API references for [\(C++\)](#) and [\(.NET\)](#).
 - If you have temporary security credentials, add code to regularly refresh the credentials using the AWS Security Token Service (AWS STS), as described in [Using temporary security credentials with the AWS SDKs](#) in the *IAM User Guide*. The code must request new credentials before the old ones expire.

Get active game sessions

Add code to discover available game sessions and manage game session settings and metadata.

Search for active game sessions

Use [SearchGameSessions](#) to get information about a specific game session, all active sessions, or sessions that meet a set of search criteria. This call returns a [GameSession](#) object for each active game session that matches your search request. This object contains the DNS name, IP address, and port, which a game client needs to connect to the game session.

Use search criteria to get a filtered list of active game sessions for players to join. For example, you can filter sessions as follows:

- Exclude game sessions that are full: `CurrentPlayerSessionCount = MaximumPlayerSessionCount`.
- Choose game sessions based on length of time that the session has been running: Evaluate `CreationTime`.
- Find game sessions based on a custom game property: `gameSessionProperties.gameMode = "brawl"`.

Manage game session data

Use any of the following operations to retrieve or update game session information.

- [DescribeGameSessionDetails\(\)](#) – Get a game session's protection status in addition to game session information.
- [UpdateGameSession\(\)](#) – Change a game session's metadata and settings as needed.
- [GetGameSessionLogUrl](#) – Access stored game session logs.

Create game sessions

Add code to start new game sessions on your deployed fleets and make them available to players. There are two options for creating game sessions, depending on how your game hosting solution manages game session placements.

Create a game session with a multi-location queue

Use [StartGameSessionPlacement](#) to place a request for a new game session in a queue. To use this operation, create a queue. This determines where Amazon GameLift Servers places the new game session. For more information about queues and how to use them, see [Configure game session placement](#).

When creating a game session placement, specify the name of the queue to use, a game session name, a maximum number of concurrent players, and an optional set of game properties. You can also optionally provide a list of players to automatically join the game session. If you include player latency data for relevant Regions, then Amazon GameLift Servers uses this information to place the new game session on a fleet that provides the ideal gameplay experience for the players.

To obtain accurate latency measurements, use Amazon GameLift Servers's UDP ping beacons. These endpoints enable you to measure actual UDP network latency between player devices and potential hosting locations, resulting in more accurate placement decisions than using ICMP pings. For more information on using UDP ping beacons to measure latency, refer to [UDP ping beacons](#).

Game session placement is an asynchronous process. After you've placed a request, you can let it succeed or time out. You can also cancel the request at any time using [StopGameSessionPlacement](#). To check the status of your placement request, call [DescribeGameSessionPlacement](#).

Create a game session on a specific fleet

Use [CreateGameSession](#) to create a new session on a specified fleet. This synchronous operation succeeds or fails depending on whether the fleet has resources available to host a new game session. After Amazon GameLift Servers creates the new game session and returns a [GameSession](#) object, you can join players to it.

When you use this operation, provide a fleet ID or alias ID, a session name, and the maximum number of concurrent players for the game. Optionally, you can include a set of game properties. Game properties are defined in an array of key-value pairs.

If you use the Amazon GameLift Servers resource protection feature to limit the number of game sessions that a player can create, then provide the game session creator's player ID.

Join a player to a game session

Add code to reserve a player slot in an active game session and connect game clients to game sessions. This action is available if your game uses player sessions with unique player IDs. For more information about player sessions, see [Amazon GameLift Servers and the player experience](#).

1. Reserve a player slot in a game session

To reserve a player slot, create a player session for the game session. There are two ways to do this:

- If you use [StartGameSessionPlacement](#) or [StartMatchmaking](#) to create a game session, you can include one or more players in the request to create player sessions for them in the new game session.
- To add players to an existing game session, call [CreatePlayerSession](#) or [CreatePlayerSessions](#) with a game session ID.

A player session request must include a unique player ID . For more information, see [Generate player IDs](#). On receiving the request, Amazon GameLift Servers verifies that the game session is accepting new players and has available player slots. If successful, Amazon GameLift Servers reserves a slot for the player, creates the new player session, and returns a [PlayerSession](#) object.

A player session can include a set of custom player data. This data is stored in the newly created player session object. Amazon GameLift Servers passes this object to the game server when the player connects directly to the game session. When requesting multiple player sessions, map each string of player data to a player ID.

2. Connect to a game session

Add code to the backend service to retrieve the `PlayerSession` object (such as by calling [DescribePlayerSessions\(\)](#)) and pass it back to the game client. This object contains the DNS name, IP address, and port. The game client can use this information to establish a direct connection to the server.

- You can connect using the specified port and the DNS name or IP address assigned to the server process.
- If your fleets have TLS certificate generation enabled, then connect using the DNS name and port.
- If your game server validates incoming player connections, then reference the player session ID.

After making the connection, the game client and server process communicate directly without involving Amazon GameLift Servers. The server maintains communication with Amazon GameLift Servers to report player connection status, health status, and more.

If the game server validates incoming players, then it verifies that the player session ID matches a reserved slot in the game session, and either accepts or denies the player connection. When the player disconnects, the server process reports the dropped connection.

Pass custom game data to a game session

Your game client can pass data to a game session. When you create a game session, you can include a set of game properties (key-value pairs) or game session data (string values) in the request. You can also update an existing game session with new or updated game data. This data is passed to the game server process that is hosting the game session, for use by the game server code. You cannot delete game properties.

As an example, let's say that your game defines difficulty levels: Novice, Easy, Intermediate, and Expert. A player chooses to join an Easy, game. Your game client, through the backend service, requests a new game session with the following game property: {"Key": "Difficulty", "Value": "Easy"}. In response, Amazon GameLift Servers prompts an available game server to start a new game session and passes the `GameSession` object. The game server process uses the game property provided to set the game session's difficulty level.

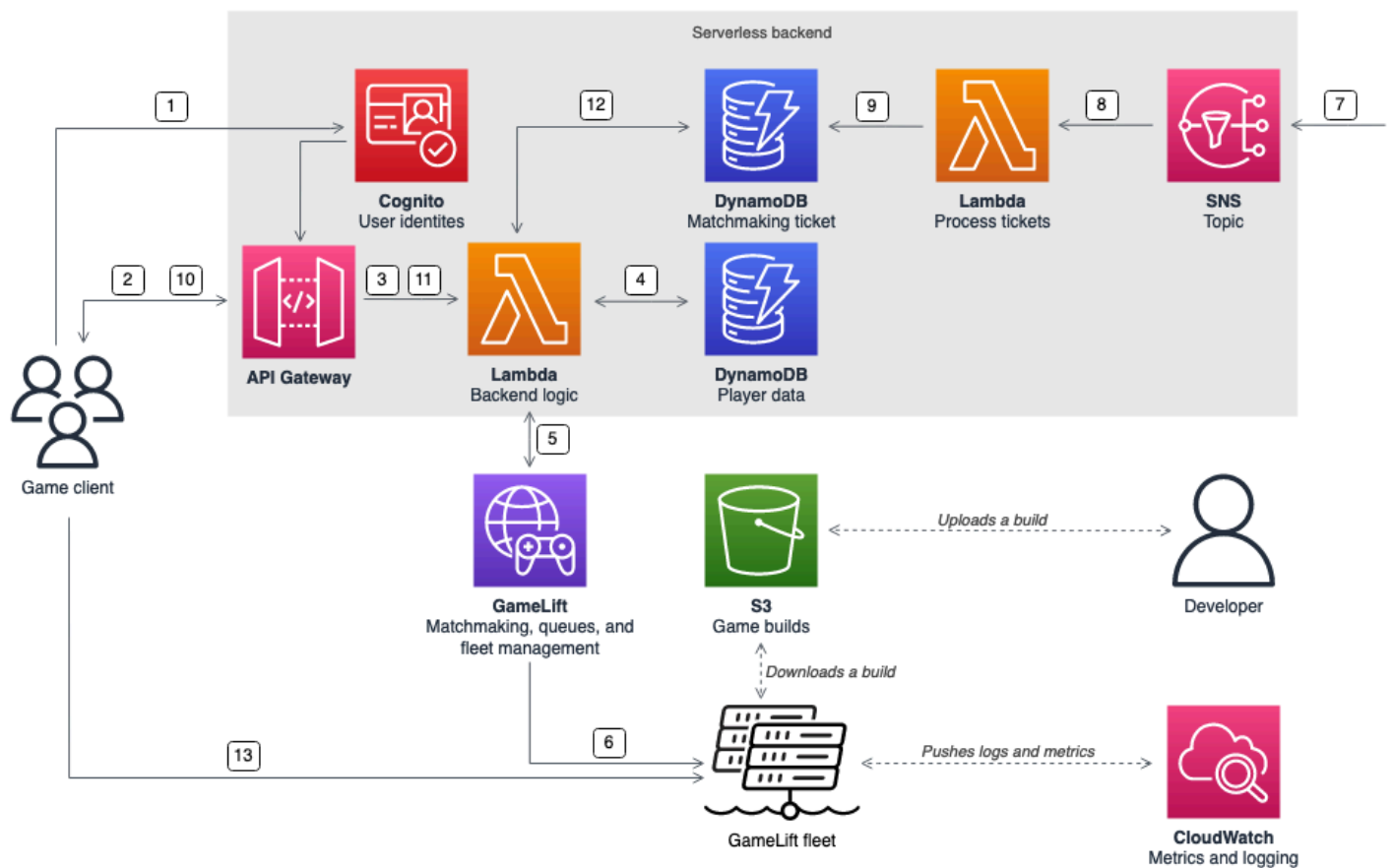
Learn more

- [GameProperty data type](#)
- [SearchGameSessions\(\) examples](#)
- [UpdateGameSession\(\) GameProperties parameter](#)

Standalone game session servers with a serverless backend

Using a serverless client service architecture, the backend can view the status of matchmaking tickets from a highly scalable database instead of by directly accessing the Amazon GameLift Servers API.

The following diagram shows a serverless backend built with AWS services that matches players into games running on Amazon GameLift Servers fleets. The following list provides a description for each numbered callout in the diagram. To try out this example, see [Multiplayer Session-based Game Hosting on AWS](#) on GitHub.



1. The game client requests an Amazon Cognito user identity from an Amazon Cognito identity pool.
2. The game client receives temporary access credentials and requests a game session through an Amazon API Gateway API.
3. API Gateway invokes an AWS Lambda function.
4. The Lambda function requests player data from an Amazon DynamoDB NoSQL table. The function provides the Amazon Cognito identity in the request context data.
5. The Lambda function requests a match through Amazon GameLift Servers FlexMatch matchmaking.
6. FlexMatch matches a group of players with suitable latency, and then requests a game session placement through an Amazon GameLift Servers queue. The queue has fleets with one or more AWS Region locations in it.
7. After Amazon GameLift Servers places the session on one of the fleet's locations, Amazon GameLift Servers sends an event notification to an Amazon Simple Notification Service (Amazon SNS) topic.

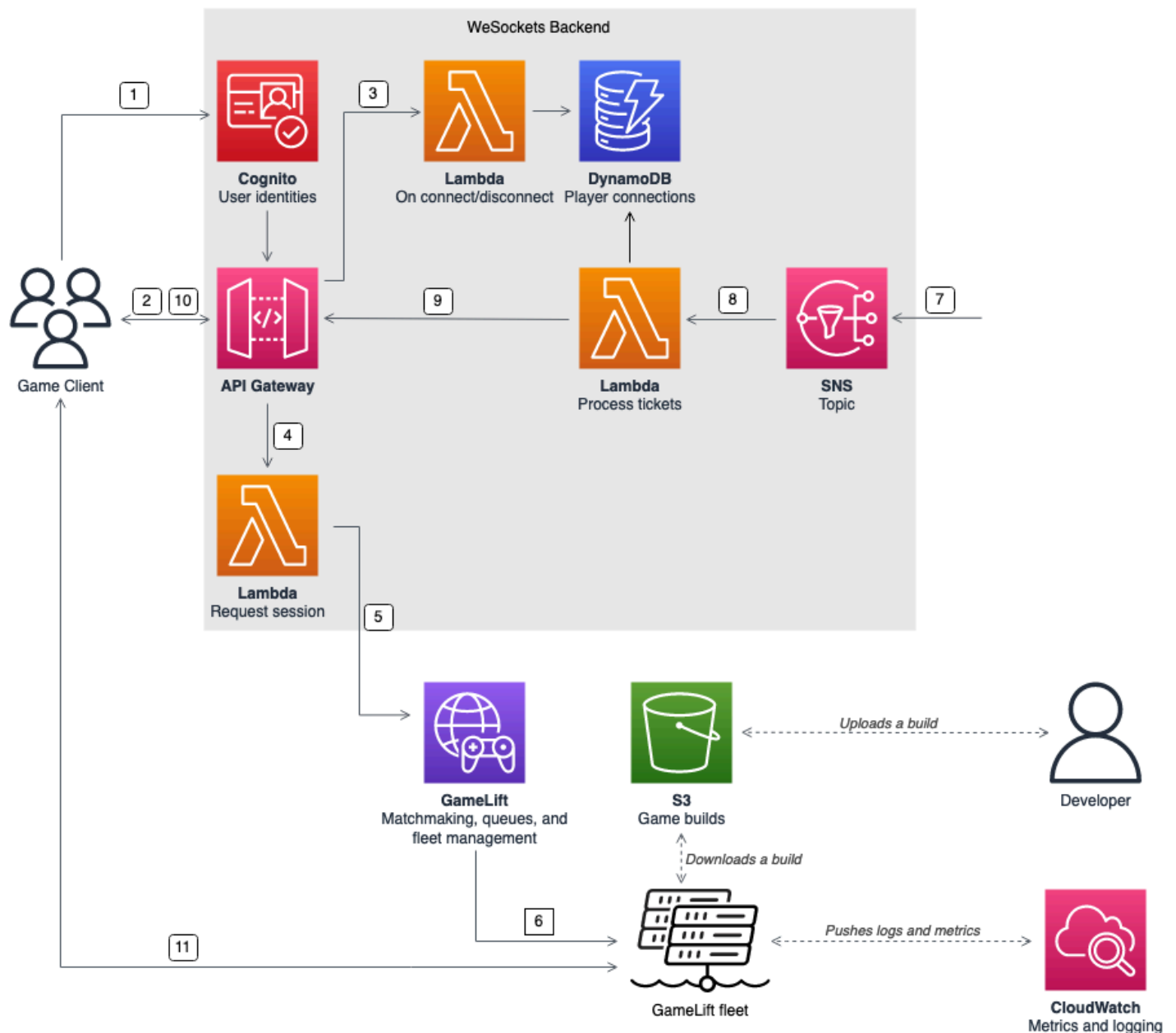
8. A Lambda function receives the Amazon SNS event and processes it.
9. If the matchmaking ticket is a MatchmakingSucceeded event, then the Lambda function writes the result, along with the port and IP address of the game server, to a DynamoDB table.
10. The game client makes a signed request to API Gateway to view the status of the matchmaking ticket on a specific interval.
11. API Gateway uses a Lambda function that checks the matchmaking ticket status.
12. The Lambda function checks the DynamoDB table to see if the ticket is successful. If it has succeeded, the function sends the game server's port and IP address, along with the player session ID, back to the client. If the ticket hasn't succeeded, the function sends a response verifying that the match isn't ready yet.
13. The game client connects to the game server using TCP or UDP by using the port and IP address that the backend service provides. The game client then sends the player session ID to the game server, which then validates the ID using the server SDK for Amazon GameLift Servers.

Standalone game session servers with a WebSocket-based backend

Using an Amazon API Gateway WebSocket-based architecture, you can make matchmaking requests with WebSockets and send push notifications for matchmaking completion using server-initiated messages. This architecture improves performance by having two-way communication between the client and the server.

For more information about using API Gateway WebSock APIs, see [Working with WebSocket APIs](#).

The following diagram shows a WebSocket-based backend architecture that uses API Gateway and other AWS services to match players into games running on Amazon GameLift Servers fleets. The following list provides a description for each numbered callout in the diagram.



1. The game client requests an Amazon Cognito user identity from an Amazon Cognito identity pool.
2. The game client signs a WebSocket connection to an API Gateway API with the Amazon Cognito credentials.
3. API Gateway calls an AWS Lambda function on the connection. The function stores the connection information in an Amazon DynamoDB table.
4. The game client sends a message to a Lambda function, through the API Gateway API over the WebSocket connection, to request a session.

5. A Lambda function receives the message and then requests a match through Amazon GameLift Servers FlexMatch matchmaking.
6. After FlexMatch matches a group of players, FlexMatch requests a game session placement through an Amazon GameLift Servers queue.
7. After Amazon GameLift Servers places the session on one of the fleet's locations, Amazon GameLift Servers sends an event notification to an Amazon Simple Notification Service (Amazon SNS) topic.
8. A Lambda function receives the Amazon SNS event and processes it.
9. If the matchmaking ticket is a MatchmakingSucceeded event, then the Lambda function requests the correct player connection from DynamoDB. The function then sends a message to the game client through the API Gateway API over the WebSocket connection. In this architecture, the game client doesn't actively poll the status of matchmaking.
10. The game client receives the port and IP address of the game server, along with the player session ID, through the WebSocket connection.
11. The game client connects to the game server using TCP or UDP using the port and IP address that the backend service provides. The game client also sends the player session ID to the game server, which then validates the ID using the server SDK for Amazon GameLift Servers.

Deploy hosting fleets for Amazon GameLift Servers

Deploying hosting resources involves creating and configuring the compute infrastructure that will run your game servers. Amazon GameLift Servers offers several fleet types to match different hosting needs, from fully managed AWS Cloud resources to hybrid solutions that combine cloud and on-premises infrastructure.

Choose the fleet type that best fits your requirements for cost, control, scalability, and geographic distribution. You can also combine multiple fleet types in a single hosting solution to optimize for different scenarios or player populations.

Fleet characteristics

An Amazon GameLift Servers fleet is a collection of computing resources that run your game servers and host game sessions for players. Fleets can vary in the type of compute resources you use and how the fleet is managed. A fleet's size—the number of game sessions and players that it can support—depends on the number of compute resources that you give it. All Amazon GameLift Servers fleets have the following characteristics:

- The game server processes that run on all fleets are integrated with the server SDK for Amazon GameLift Servers and communicate with the Amazon GameLift Servers service in the same way. Game servers report their availability to host game sessions and players, respond to prompts to start or stop game sessions, and other interactions.
- Amazon GameLift Servers handles game session placement for all fleets in the same way. Amazon GameLift Servers keeps track of a fleet's game server status and chooses from available game servers to host a new game session. This process is used whether your game places game sessions on a single fleet or uses a [game session queue](#) to balance hosting across multiple fleets. With a queue, you can also customize placement decisions to consider factors such as resource cost and latency.
- All fleets support the use of a FlexMatch matchmaker in collaboration with a game session placement queue. The Amazon GameLift Servers service receives player match requests, forms the matches, and passes them to the game session queue to find available game servers.
- Amazon GameLift Servers collects a wide range of fleet metrics. These include status metrics for computes and server processes, as well as usage metrics for game sessions and player activity. See the complete list of available metrics at [Monitor Amazon GameLift Servers with Amazon CloudWatch](#).

Topics

- [Decision guide: Choose a hosting option](#)
- [Amazon GameLift Servers managed EC2 fleets](#)
- [Amazon GameLift Servers managed container fleets](#)
- [Amazon GameLift Servers Anywhere fleets](#)
- [Build a hybrid hosting solution](#)

Decision guide: Choose a hosting option

Amazon GameLift Servers offers several hosting options, each designed for different hosting scenarios and requirements. This guide helps you choose the type of fleet based on your game's technical needs, operational preferences, and business constraints. For high-level descriptions, see [Amazon GameLift Servers game hosting options](#).

Fleet comparison

The following table compares the key characteristics of each hosting option:

Hosting option	Best For	Management Level	Key Benefits
Managed EC2	Production hosting with full AWS management	Fully managed	Auto-scaling, global reach, minimal operational overhead
Managed containers	Containerized applications requiring orchestration	Fully managed	Container orchestration, resource efficiency, modern deployment
Anywhere	On-premises, hybrid, or development environments	Self-managed	Hardware control, compliance requirements, cost optimization

Key decision factors

Consider these factors when choosing your hosting option:

Operational complexity

How much infrastructure management do you want to handle? Managed fleets require minimal operational overhead, while Anywhere fleets give you full control but require more management.

Scalability requirements

Do you need automatic scaling based on player demand? Managed fleets provide built-in auto-scaling, while Anywhere fleets require manual capacity management.

Geographic distribution

Where are your players located? Managed fleets can be deployed across multiple AWS Regions globally, while Anywhere fleets depend on your own infrastructure locations.

Compliance and security

Do you have specific compliance requirements or need to keep data on-premises? Anywhere fleets allow you to maintain full control over your infrastructure and data location.

Cost considerations

What's your budget and cost optimization strategy? Consider both infrastructure costs and operational overhead when comparing options.

Hybrid solutions

You don't have to choose just one type of fleet for your game hosting solution. Many games use a hybrid approach that combines multiple fleet types to optimize for different scenarios:

- **Development and testing:** Use Anywhere fleets for development and testing, then deploy to managed fleets for production.
- **Regional optimization:** Use managed fleets in AWS Regions with high player density and Anywhere fleets in regions where you have existing infrastructure.
- **Cost optimization:** Use managed fleets for baseline capacity and Anywhere fleets for overflow or specific workloads.

Amazon GameLift Servers managed EC2 fleets

Amazon GameLift Servers managed EC2 fleets provide cloud-based resources for production hosting. With a managed fleet, you get the flexibility, security, and reliability of AWS Cloud resources optimized for multiplayer game hosting. Amazon GameLift Servers provides robust host management tools.

A managed EC2 fleet is a set of Amazon Elastic Compute Cloud (Amazon EC2) instances that Amazon GameLift Servers owns and operates based on your configuration. These instances are physically located in supported AWS Regions or Local Zones. When you create a fleet, you choose an EC2 instance type that meets your game server's requirements for computing power, memory, storage, networking capabilities.

When starting each instance in the fleet, Amazon GameLift Servers deploys your game server build with the required runtime environment. The runtime environment uses the latest Amazon Machine Image (AMI) version available when the fleet is created. All instances in the fleet use the same AMI version.

Note

As a best practice, we recommend replacing your fleets every 30 days to maintain a secure and up-to-date runtime environment for your hosted game servers. This requires creating a new fleet and migrating player traffic to it. For more guidance, see [Security best practices for Amazon GameLift Servers](#).

After installing the runtime environment and your game server build on an instance, Amazon GameLift Servers starts launching game server processes. Each game server process establishes a connection to the Amazon GameLift Servers service, reports readiness to host a game session, and begins communicating health status. Amazon GameLift Servers can then prompt the server process to start a game session.

In addition to fleet deployment, Amazon GameLift Servers handles the following host management tasks so you don't have to:

- Tracks the status of all computes in the fleet and replaces stale or unhealthy computes.
- Handles authentication for communication between server processes and the Amazon GameLift Servers service.
- Automatically starts and stops game server processes on each compute, based on your runtime configuration.
- Offers auto scaling tools to adjust fleet capacity dynamically to meet player demand.
- Reports performance metrics for the fleet's EC2 instances.

See these topics about how to set up and maintain managed EC2 fleets:

- [Development roadmap for hosting with Amazon GameLift Servers managed EC2](#)
- [Create an Amazon GameLift Servers managed EC2 fleet](#)
- [Scaling game hosting capacity with Amazon GameLift Servers](#)
- [Hosting resource customizations](#)
- [Update an Amazon GameLift Servers fleet configuration](#)

Managed EC2 fleet creation workflow

For managed fleets, Amazon GameLift Servers sets up the fleet resource and also deploys a set of compute resources with your game server software installed and running. When the creation workflow is complete and successful, the fleet has one active EC2 instance in the fleet home Region and one each in the fleet's remote locations. All instances have game servers and are ready to host game sessions.

1. Amazon GameLift Servers creates the fleet resource in the fleet's home Region and sets desired capacity in each location to one (1) instance. Fleet and location status is set to **New**.
2. Amazon GameLift Servers begins writing events to the fleet event log.
3. Amazon GameLift Servers sets fleet status to **Downloading** and begins preparing the game server software for deployment.
 - a. Gets the uploaded game server build and extracts the compressed files.
 - b. Runs install scripts, if provided.
 - c. Sets fleet status to **Validating** and begins verifying that no errors occurred when downloading and installing the build files.
4. Amazon GameLift Servers sets the fleet status to **Building**, configures fleet hardware, and allocates one EC2 instance for each fleet instance.
5. Amazon GameLift Servers sets the fleet status to **Activating**. Launches a game server process on each instance (based on the fleet's runtime instructions) and tests connectivity between the build and the Amazon GameLift Servers service.
6. When game server processes on each instance establish a connection and report readiness to host game sessions, Amazon GameLift Servers sets fleet and location statuses to **Active**. At this point, the fleet is considered ready to host game sessions.

Create an Amazon GameLift Servers managed EC2 fleet

This topic describes how to create an Amazon GameLift Servers managed EC2 fleet. Managed fleets use Amazon Elastic Compute Cloud (Amazon EC2) compute instances that are optimized for multiplayer game hosting. You can create managed fleets that deploy computes to globally to AWS Regions and Local Zones supported by Amazon GameLift Servers.

When you create a new managed EC2 fleet resource, you immediately initiate the first phase of fleet creation. Managed fleet creation passes through several phases as Amazon GameLift Servers deploys an EC2 instance, installs a runtime environment and your game server build on the

instance, and begins launching game servers. Depending on the runtime environment your game server build requires, Amazon GameLift Servers deploys the latest version of the Amazon Machine Image (AMI) at the time of fleet creation (and all future instances in the fleet will use the same version). You can monitor a fleet's status in the console or using the AWS Command Line Interface (AWS CLI). When a fleet is ready to host game sessions, the status changes to **ACTIVE**. For more information about managed fleet creation, see these topics:

Note

As a best practice, we recommend replacing your fleets every 30 days to maintain a secure and up-to-date runtime environment for your hosted game servers. This requires creating a new fleet and migrating player traffic to it. For more guidance, see [Security best practices for Amazon GameLift Servers](#).

To create a managed EC2 fleet

Use either the Amazon GameLift Servers console or the AWS Command Line Interface (AWS CLI) to create a managed EC2 fleet.

Console

In the [Amazon GameLift Servers console](#), use the navigation pane to open the **Managed EC2, Fleets** page. Choose **Create managed EC2 fleet** to start the fleet creation workflow.

Step 1 Define managed EC2 fleet details

For a minimal fleet configuration:

- Provide a fleet name.
- Choose **Binary type: Build**. Choose a build from the list of server builds that you've previously uploaded to Amazon GameLift Servers.
- Skip the Additional details and Tags sections to use default settings.

1. Fill out the **Fleet details** section:

- a. Enter a fleet **Name**. Fleet names are descriptive only, useful for sorting or filtering a list of fleets. They don't need to be unique.

- b. Provide a short fleet **Description** to include any custom information you want to capture about the fleet.
 - c. Select **Binary type, Build**, and use the dropdown list to choose a game server build that you've previously uploaded to Amazon GameLift Servers.
2. Set **Additional details** as needed.
 - a. If your game server executable needs to access other AWS resources in your account, specify an IAM **Instance role** with the necessary permissions. For more information, including how to authorize other server-side applications (such as CloudWatch agent), see [Connect your Amazon GameLift Servers hosted game server to other AWS resources](#). This setting can't be changed after you create the fleet.

You must create the role before you create a fleet that uses it. In addition, to create a fleet with an instance role, your AWS user must have IAM `PassRole` permission (see [IAM permission examples for Amazon GameLift Servers](#)).
 - b. Turn on the **Generate a TLS certificate** option to set up authentication and encryption for your game. Game clients use this certificate to authenticate a game server when connecting and encrypt all client/server communication. For each instance in a TLS-enabled fleet. Amazon GameLift Servers also creates a new DNS entry with the certificate. This setting can't be changed after you create the fleet.
 - c. Amazon GameLift Servers emits metric data for each individual fleet. If you want to combine metric data for multiple fleets, specify a **Metric group** name. Use the same metric group name for all fleets that you want to combine metrics for. Use CloudWatch to view the aggregated metric group data.
3. Add **Tags** to the fleet resource. Each tag consists of a key and an optional value, both of which you define. Assign tags to AWS resources that you want to categorize, such as by purpose, owner, or environment. Choose **Add new tag** for each tag that you want to add.
4. Choose **Next** to continue the workflow.

Step 2 Define instance details

In this step, select the type of hardware resources to use for your game servers and specify where you want to deploy them. By choosing multiple locations, you can deploy your game server to a wider geographical location, which puts them closer to your players and

minimizes latency. All EC2 instances in a fleet must be the same type. Not all instance types are available in all locations.

i For a minimal fleet configuration:

- Start with deploying to the home location only. You don't need to add remote locations for initial development and testing.
- Set fleet type set to "On-Demand". Spot fleets require additional setup work.
- Set instance type to "c5.large". This commonly used instance type is available in all AWS Regions. You can optimize your hardware usage later.

1. In **Instance deployment**, specify fleet locations and type.
 - a. Select one or more additional **Locations** where you want to deploy fleet instances. These remote locations are added to the fleet's home location (which is pre-selected), which is the AWS Region where you're creating this fleet. You can select remote locations from all AWS Regions and Local Zones that Amazon GameLift Servers supports.

To learn more about supported locations, including how to use an AWS Region that isn't enabled by default, see [Amazon GameLift Servers service locations](#) for managed hosting. Also review Amazon GameLift Servers [quotas](#) on locations per fleet.
 - b. Choose to use either **On-demand** or **Spot** instances for this fleet. For more information about fleet types, see [On-Demand Instances versus Spot Instances](#).
2. Choose an Amazon EC2 **Instance configuration** that meets your needs and is available in all your selected locations. This list is filtered based on your current location and fleet type selections. You can filter it further by other factors such as instance type family and architecture. After you create the fleet, you can't change the instance type.

Some locations have limited instance type options. If your preferred instance type is not available for all locations, choose the location availability value to view full details. To accommodate all locations, you might need to create separate fleets with different instance types.

For more information about choosing an instance type, see [Instance types](#). To learn more about Amazon EC2 Arm architectures, see [AWS Graviton Processor](#) and [Amazon](#)

[EC2 instance types](#). For a complete list of instance types supported by Amazon GameLift Servers, see the API reference for [EC2InstanceType](#) (`CreateFleet()`).

 Note

Graviton Arm instances require a server build for a Linux AMI. Server SDK 5.1.1 or newer is required for C++ and C#. Server SDK 5.0 or newer is required for Go. These instances do not provide out-of-the-box support for Mono installation on Amazon Linux 2023 (AL2023) or Amazon Linux 2 (AL2).

 Note

If you select an ARM architecture instance, make sure the build you selected runs on ARM architecture. Otherwise the fleet deployment will fail.

3. Choose **Next** to continue the workflow.

Step 4 Configure runtime

In this step, describe how you want each instance in the fleet to run your game server software. Define a separate server process line item for each executable to run on an instance, and decide how many of each server process to run concurrently. Open ports on each instance to allow players to connect directly to game servers. You can update these fleet settings at any time.

 For a minimal fleet configuration:

- Define a single server process line item for your game server executable. If your game server requires other processes to be running, create a definition for each of these as well.
- Use the default number of concurrent processes (1) for each line item.
- Skip game session activation settings.
- Specify a single port number.
- Skip game session resource settings.

1. Create a **Runtime configuration** to instruct Amazon GameLift Servers on how to run server processes on each instance in the fleet. You can change a fleet's runtime configuration at any time after deployment.
 - a. Enter the **Launch path** to an executable file in your build. On Windows instances, game server executables are built to the path `C:\game`. On Linux instances, game servers are built to `/local/game`. Examples: `C:\game\MyGame\server.exe` or `/local/game/MyGame/server.exe`.
 - b. Enter optional **Launch parameters** to pass to your game executable. Example: `+sv_port 33435 +start_lobby`.
 - c. Specify the number of **Concurrent processes** to run on each instance. For a game server executable, each process can host one game session, so concurrent processes determines the number of game sessions the instance can host simultaneously.

Review the Amazon GameLift Servers [quotas](#) on server processes per instance. These quotas apply to the total concurrent processes for all configurations. If you configure the fleet to exceed them, the fleet can't activate.

2. Use the **Game session activation** defaults or customize them for your game. If the runtime configuration calls for multiple concurrent game server process per instance, these settings determine how quickly new game sessions can start up.
 - a. Set **Max concurrent game session activation** to limit the number of game servers on an instance that are preparing a new game session. This setting is useful when launching multiple new game sessions is resource-intensive and might impact the performance of other running game sessions.
 - b. Set the **New activation timeout** to reflect the maximum amount of time a new game session should take to complete activation and report ready to host players. Amazon GameLift Servers terminates a game session activation if it exceeds this value.
3. Open **EC2 port settings** to allow inbound traffic to access server processes on the fleet. These settings aren't required to create a fleet, but you do need set them before players can connect to game sessions on the fleet.

For each port setting, choose the **Type** of data transfer protocol to use for communication between your game client and game server. Provide a **Port range** (in format `nnnnn[-nnnnn]`) and an **IP address range** using CIDR notation (such as `0.0.0.0/0` which allows access to anyone).

If you need to set multiple non-consecutive ranges, create multiple port settings.

4. Specify optional **Game session resource settings**. You can update these settings any time after deployment.
 - a. Turn **Game scaling protection policy** on or off for all instances in the fleet. During a scale-down event, Amazon GameLift Servers won't terminate protected fleet instances if they're hosting active game sessions.
 - b. Set a maximum **Resource creation limit** if you want to restrict the number of game sessions that one player can create during a specified time span.
5. Choose **Next** to continue the workflow.

Step 5 Review and create

Review your settings before creating the fleet. Although some settings can be updated later (see [Update an Amazon GameLift Servers fleet configuration](#)), changes to the following settings aren't allowed after the fleet has been created:

- Compute type: You can't convert a managed EC2 fleet to an Anywhere fleet.
- Game server binary: To deploy an update to your game server build, you must create a new fleet.
- Additional options, including instance role and TLS certificate generation.
- Instance details, including fleet type (Spot or On-Demand) and EC2 instance type.

When you're ready to deploy the new fleet, choose **Create**. Amazon GameLift Servers immediately begins the fleet activation process, assigning a unique ID and placing the fleet in NEW status. Track the fleet's progress from the **Fleets** page. View the details page for the fleet and go to the **Events** tab.

You can adjust a fleet's hosting capacity after the fleet reaches ACTIVE status. Amazon GameLift Servers initially deploys a fleet with a single instance in each fleet location, and you adjust capacity by adding instances to each location. For more information, see [Scaling game hosting capacity with Amazon GameLift Servers](#).

AWS CLI

Use the [create-fleet](#) command to create a fleet of compute type EC2. Amazon GameLift Servers creates the fleet resource in your current default AWS Region (or you can add a `--region` tag to specify a different AWS Region).

Create a minimal managed fleet

The following example request creates a new fleet with the minimal settings that are required to deploy a fleet with running game servers that game clients can connect to. The new fleet has these characteristics:

- It specifies a game server build, which has been uploaded to Amazon GameLift Servers and in READY status.
- It uses c5.large On-Demand Instances with an operating system that matches the selected game build.
- It sets the fleet's home AWS Region to `us-west-2` and deploys instances to that Region only.
- Based on the runtime configuration, each compute in the fleet runs one game server process, which means that each compute can host only one game session at a time. Game session activation timeout is set to the default value of 300 seconds, and there's no limit on the number of concurrent activations.
- Players can connect to game servers using a single port setting of 33435.
- All other features are either turned off or use default settings.

```
aws gamelift create-fleet \  
  --name MinimalFleet123 \  
  --description "A basic test fleet" \  
  --region us-west-2 \  
  --ec2-instance-type c5.large \  
  --fleet-type ON_DEMAND \  
  --build-id build-1111aaaa-22bb-33cc-44dd-5555eeee66ff \  
  --runtime-configuration "ServerProcesses=[{LaunchPath=C:\game\Bin64.dedicated\  
\MultiplayerSampleProjectLauncher_Server.exe, ConcurrentExecutions=10}]" \  
  --ec2-inbound-permissions  
  "FromPort=33435,ToPort=33435,IpRange=0.0.0.0/0,Protocol=UDP"
```

Create a fully configured managed fleet

The following example request creates a production fleet with settings for all optional features. The new fleet has these characteristics:

- It specifies a game server build, which has been uploaded to Amazon GameLift Servers and in READY status.
- It uses c5.large On-Demand Instances with the operating system that matches the selected game build.
- It sets the fleet's home AWS Region to us-west-2 and deploys instances to the home Region and one remote location sa-east-1.
- Based on the runtime configuration:
 - Each compute in the fleet runs 10 game server processes with the same launch parameters, which means that each compute can host up to 10 game sessions simultaneously.
 - On each compute, only two game sessions can be activating at the same time. Activating game sessions must be ready to host players within 300 seconds (5 minutes) or be terminated.
- Players can connect to game servers using a port in the following range 33435 to 33535.
- It generates a TLS certificate for encrypted communication between game client and server.
- All game sessions in the fleet have game session protection turned on.
- Individual players are limited to creating three new game sessions within a 15-minute period.
- Metrics for this fleet are included in the metric group AMERfleets, which (for this example) aggregates metrics for a group of fleets in North, Central, and South America.

```
aws gamelift create-fleet \  
  --name ProdFleet123 \  
  --description "A fully configured prod fleet" \  
  --ec2-instance-type c5.large \  
  --region us-west-2 \  
  --locations "Location=sa-east-1" \  
  --fleet-type ON_DEMAND \  
  --build-id build-1111aaaa-22bb-33cc-44dd-5555eeee66ff \  
  --certificate-configuration "CertificateType=GENERATED" \  
  --runtime-configuration "GameSessionActivationTimeoutSeconds=300,  
MaxConcurrentGameSessionActivations=2, ServerProcesses=[{LaunchPath=C:\game  
\Bin64.dedicated\MultiplayerSampleProjectLauncher_Server.exe, Parameters=+sv_port  
33435 +start_lobby, ConcurrentExecutions=10}]" \  
  --new-game-session-protection-policy "FullProtection" \  

```

```
--resource-creation-limit-policy "NewGameSessionsPerCreator=3,  
PolicyPeriodInMinutes=15" \  
--ec2-inbound-permissions  
"FromPort=33435,ToPort=33535,IpRange=0.0.0.0/0,Protocol=UDP" \  
--metric-groups "AMERfleets"
```

If the create-fleet request is successful, Amazon GameLift Servers returns a set of fleet attributes that includes the configuration settings you requested and a new fleet ID. Amazon GameLift Servers then initiates the fleet activation process and sets the fleet status and the location statuses to **New**. You can track the fleet's status and view other fleet information using these CLI commands:

- [describe-fleet-events](#)
- [describe-fleet-attributes](#)
- [describe-fleet-capacity](#)
- [describe-fleet-port-settings](#)
- [describe-fleet-utilization](#)
- [describe-runtime-configuration](#)
- [describe-fleet-location-attributes](#)
- [describe-fleet-location-capacity](#)
- [describe-fleet-location-utilization](#)

You can change the fleet's capacity and other configuration settings as needed using these commands:

- [update-fleet-attributes](#)
- [update-fleet-capacity](#)
- [update-fleet-port-settings](#)
- [update-runtime-configuration](#)
- [create-fleet-locations](#)
- [delete-fleet-locations](#)

Amazon GameLift Servers managed container fleets

Amazon GameLift Servers managed container fleets provide cloud-based resources for hosting your containerized game server software. With a managed container fleet, you get the flexibility, security, and reliability of AWS Cloud resources, optimized for multiplayer game hosting. Amazon GameLift Servers provides robust host management tools.

Speed up onboarding with these tools for managed containers:

- The [containers starter kit](#) streamlines integration and fleet setup. It adds essential game session management features to your game server, and uses pre-configured templates to build a container fleet and an automated deployment pipeline for your game server. After deployment, use the Amazon GameLift Servers console and API tools to monitor fleet performance, manage game sessions, and analyze metrics.
- For Unreal Engine or Unity developers, use the game engine [Amazon GameLift Servers plugins and server SDKs](#) to integrate your game server and build a container fleet from inside your game engine's development environment. The plugin's guided workflows help you create a fast, simple solution with cloud-based hosting using managed containers. You can build on this foundation to create a custom hosting solution for your game.

A managed container fleet is a set of Amazon Elastic Compute Cloud (Amazon EC2) instances running Linux, which Amazon GameLift Servers owns and operates based on your configuration. These instances are physically located in supported AWS Regions or Local Zones. When you create a container fleet, you choose an EC2 instance type that meets your game server's requirements for computing power, memory, storage, networking capabilities.

For a managed container fleet, you store Linux-based container images in an Amazon Elastic Container Registry (Amazon ECR) repository and create a container group definition to describe your container architecture. When you create a fleet, Amazon GameLift Servers provisions a fleet instance with the latest version of the Linux Amazon Machine Image (AMI) and uses the container group definition to deploy your container images. All instances in a container fleet will use the same AMI version, even if you update a container group definition or change a container image.

Note

As a best practice, we recommend replacing your fleets every 30 days to maintain a secure and up-to-date runtime environment for your hosted game servers. This requires creating

a new fleet and migrating player traffic to it. For more guidance, see [Security best practices for Amazon GameLift Servers](#).

After deploying the containerized instance, containers start launching game server processes. Each game server process establishes a connection to the Amazon GameLift Servers service, reports readiness to host a game session, and begins communicating health status. Amazon GameLift Servers can then prompt the server process to start a game session.

In addition to fleet deployment, Amazon GameLift Servers handles the following host management tasks so you don't have to:

- Tracks the status of all containers in the fleet and replaces stale or unhealthy ones.
- Handles authentication for communication between server processes and the Amazon GameLift Servers service.
- Offers auto scaling tools that adjust fleet capacity dynamically to meet player demand.
- Reports performance metrics for the fleet's EC2 instances, containers, and server processes.

See these topics about how to set up and maintain managed container fleets:

- [Development roadmap for hosting with Amazon GameLift Servers managed containers](#)
- [Create an Amazon GameLift Servers managed container fleet](#)
- [Customize an Amazon GameLift Servers container fleet](#)
- [Scaling game hosting capacity with Amazon GameLift Servers](#)
- [Update an Amazon GameLift Servers managed container fleet](#)

How containers work in Amazon GameLift Servers

Amazon GameLift Servers container fleets are designed to give you flexibility in how you deploy and scale your containerized applications. It uses the Amazon Elastic Container Service (Amazon ECS) to manage task deployment and execution for your Amazon GameLift Servers fleets. This topic describes the basic structural elements for running containers on an Amazon GameLift Servers managed fleet, illustrates common architectures, and outlines some core concepts.

Speed up onboarding with these tools for managed containers:

- The [containers starter kit](#) streamlines integration and fleet setup. It adds essential game session management features to your game server, and uses pre-configured templates to build a container fleet and an automated deployment pipeline for your game server. After deployment, use the Amazon GameLift Servers console and API tools to monitor fleet performance, manage game sessions, and analyze metrics.
- For Unreal Engine or Unity developers, use the [Amazon GameLift Servers plugins](#) to integrate your game server and build a container fleet from inside your game engine's development environment. The plugin's guided workflows help you create a fast, simple solution with cloud-based hosting using managed containers. Then build on this foundation to create a custom hosting solution for your game.

Container fleet components

Fleet

A container fleet is a collection of Amazon EC2 instances to host your containerized game servers. These instances are managed by Amazon GameLift Servers on your behalf. When you create a fleet, you configure how the container architecture with your game server software is deployed to each fleet instance. You can create a container fleet with instances in one or multiple geographic locations. You can use Amazon GameLift Servers scaling tools to automatically scale a container fleet's capacity to host game sessions and players.

Instance

An Amazon EC2 instance is the virtual server that provides compute capacity for game hosting. With Amazon GameLift Servers, you can choose from a range of instance types. Each instance type offers a different combination of CPU, memory, storage, and networking capacity.

When you create a container fleet, Amazon GameLift Servers deploys your containers based on the instance type you choose and your fleet configuration. Each deployed fleet instance is identical and runs your containerized game server software in the same way. The number of instances in a fleet determines the fleet's size and game hosting capacity.

Container group

Amazon GameLift Servers uses the concept of a container group to describe and manage a set of containers. A container group is similar to a container "task" or "pod". Within each container

group, you can define how containers behave, set up dependencies, and share available CPU and memory resources.

Each fleet instance can have the following types of container groups:

- A **game server container group** manages the containers that run your game server application and supporting software. A container fleet must have one of this type of container group to host game sessions and players. A game server container group can be replicated across a fleet instance. The number of game server group replicas per fleet instance depends on your software's compute requirements and the compute resources available on the instance.
- A **per-instance container group**, which is optional, gives you the ability to run additional software on each fleet instance. They are useful for running background services or utility programs, such as for monitoring. Your game server software doesn't directly depend on processes in a per-instance group. Only one copy of a per-instance container group is deployed to each fleet instance.

Each container group in a container fleet has one container that's designated "essential". An essential container drives the lifecycle of a container group. If the essential container fails, the entire container group restarts.

Container

The container is the most basic element of a container-based architecture. It includes a container image with software executables and dependent files. Define a container to configure how the software runs and interacts with Amazon GameLift Servers.

Amazon GameLift Servers defines two types of containers:

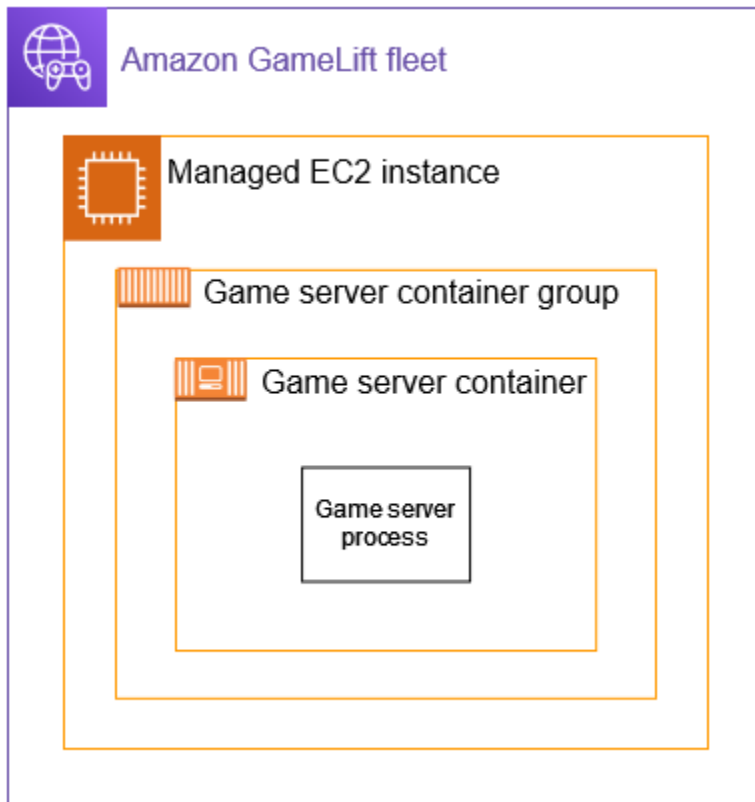
- A **game server container** includes everything you need to run your game server processes and host game sessions for players. It includes your game server build and dependent software. Define one game server container for a fleet's game server container group. The game server container is automatically considered essential to the container group.
- A **support container** runs additional software to support your game server. It is similar to the concept of a "sidecar" container. It gives you the option to run and scale supporting software alongside your game servers but manage as separate containers. In a game server container group, you can define zero or more support containers. In a per-instance container group, all containers are support containers. Any support container can be designated essential.

Compute

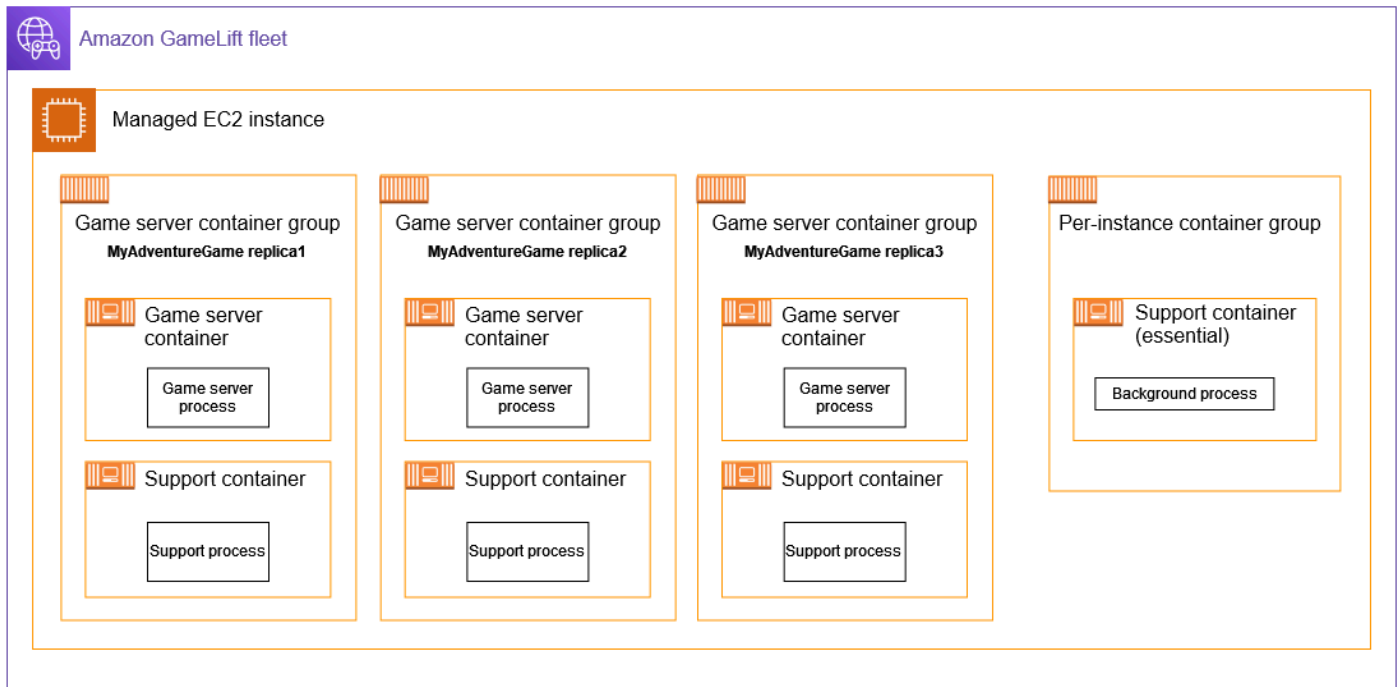
A compute represents a copy of a game server container group on a fleet instance.

Common architectures

The following diagram illustrates the simplest container fleet structure. In this structure, each instance in the fleet maintains one copy of the game server container group. The container group has a single game server container that runs one game server process. In this example, the container fleet is configured to place one copy of the game server container group per instance. With this architecture, each instance runs one game server process.



This second diagram illustrates a more complex container fleet architecture. In this structure, the fleet has both a game server container group and a per-instance container group. The game server container group has separate containers for the game server process and a support process. The fleet is configured to place three copies of the game server container group on each fleet instance. The per-instance container group is never replicated. In this example, the container fleet is configured to place three copies of the game server container group per instance. With this architecture, each instance runs three game server processes.



Core features

This section summarizes how Amazon GameLift Servers implements some basic container concepts. For instructions on how to work with container fleets, see the relevant topics in this guide.

Active fleet updates

Managed containers provides advanced support to help you manage the life cycle of your hosted software and container architecture. You can update your container definitions, including container images, and deploy changes to your existing fleets. This feature makes it faster and easier to iterate changes to your containers during development. It also provides features to help you build, deploy, and track your software version updates over time. These features include:

- **Manage container group definition updates and versioning.** You can update nearly all properties of a container group definition, including container images and configuration settings. Whenever you update a container, Amazon GameLift Servers automatically assigns a version number to the update and by default maintains all versions. You can access any specific version, and you can delete versions as desired. When creating a container fleet, you can specify the container group definition and version to deploy.
- **Update existing container fleets with new container group definitions and configuration settings.** You can deploy container updates to fleets that are already deployed to fleet instances. You can

track the status of update deployments across each fleet location using the AWS Management Console or the AWS SDK and CLI.

- Configure how you want fleet updates to deploy across an active fleet.
 - Game session protection. Choose to protect fleet instances with active game sessions until after the game sessions end (safe deployment). Or choose to replace fleet instances regardless of game session activity (unsafe deployment). Use unsafe deployments during development and testing phases to reduce deployment time.
 - Minimum healthy percentage. Specify the percentage of healthy tasks that you want to maintain during deployment. This feature lets you decide how many fleet instances are impacted during a deployment. A low value prioritizes deployment speed, while a high value ensures that game server availability remains high throughout the deployment.
 - Deployment failure strategy. Decide what actions to take if a deployment fails. A deployment failure means that some of the updated containers have failed status checks and are considered impaired. You can set deployments to automatically roll back all fleet instances to the previously deployed state. Alternatively you can choose to maintain some of the impaired fleet instances for use in debugging.

The ability to update active fleets is highly useful when you want to deploy an update to your game server software. After you've built a new container image for your game server, deploying it is a two-step process: first, update the container group definition with the new image, and second, update the container fleet. Amazon GameLift Servers handles all other tasks as needed.

Container packing

When developing your container structure for deployment in a container fleet, a common goal is to optimize your use of available computing power. To achieve this goal, you want to pack as many game server container groups as possible onto each fleet instance.

Amazon GameLift Servers helps you do this by calculating the maximum game server container groups per instance, based on the following information:

- The fleet's instance type and its vCPU and memory resources.
- The vCPU and memory requirements for all containers in the game server container group.

The vCPU and memory requirements for all containers in a per-instance container group, if there is one.

When you create a container fleet, you can use the calculated maximum or you can specify a desired number. As a best practice, plan to experiment with your containerized game server software to determine resource requirements for optimal game server performance.

Capacity scaling

Fleet capacity measures the number of game sessions that a fleet can host concurrently. You can also measure capacity based on the number of concurrent players that a fleet can support. To increase or decrease a fleet's hosting capacity, you add or remove fleet instances.

A container fleet is configured to run a specific number of concurrent game server processes on each fleet instance. (You can calculate this based on (1) the game server container groups per instance and (2) the number of game server processes that run in each container group.) The number of concurrent game servers per instance tells you what the impact is of adding or removing each fleet instance. For example, if your container fleet runs 1 game server process in each game server container group, and each fleet instance holds 100 game server containers groups, you increase or decrease your fleet's capacity to host concurrent game sessions by increments of 100. If each game session has 10 player slots, then you increase or decrease your fleet's capacity to host players by increments of 1000.

With container fleets, you can use any of the capacity scaling methods provided by Amazon GameLift Servers. These include:

- Set fleet capacity manually by setting a desired fleet instance count.
- Set up automatic scaling by targeting a desired buffer of available instances (target tracking). This method automatically maintains a certain amount of idle hosting resources so that incoming players can get into games quickly. As player demand increases or decreases, the size of this buffer is continually adjusted.
- Set up automatic scaling with custom scaling rules (advanced feature). This method lets you scale based on fleet metrics that you choose.

Game client/server connections

With Amazon GameLift Servers managed fleets, game clients connect directly to your cloud-hosted game servers. When a game client asks to join a game, Amazon GameLift Servers finds a game session and provides connection information (IP and port) to the game client. You can control external access to fleet instances by opening certain port ranges (inbound permissions) for the

fleet. Inbound permissions determine which ports are open to incoming traffic. You can quickly shut down all ports, limit to a few, or open all ports.

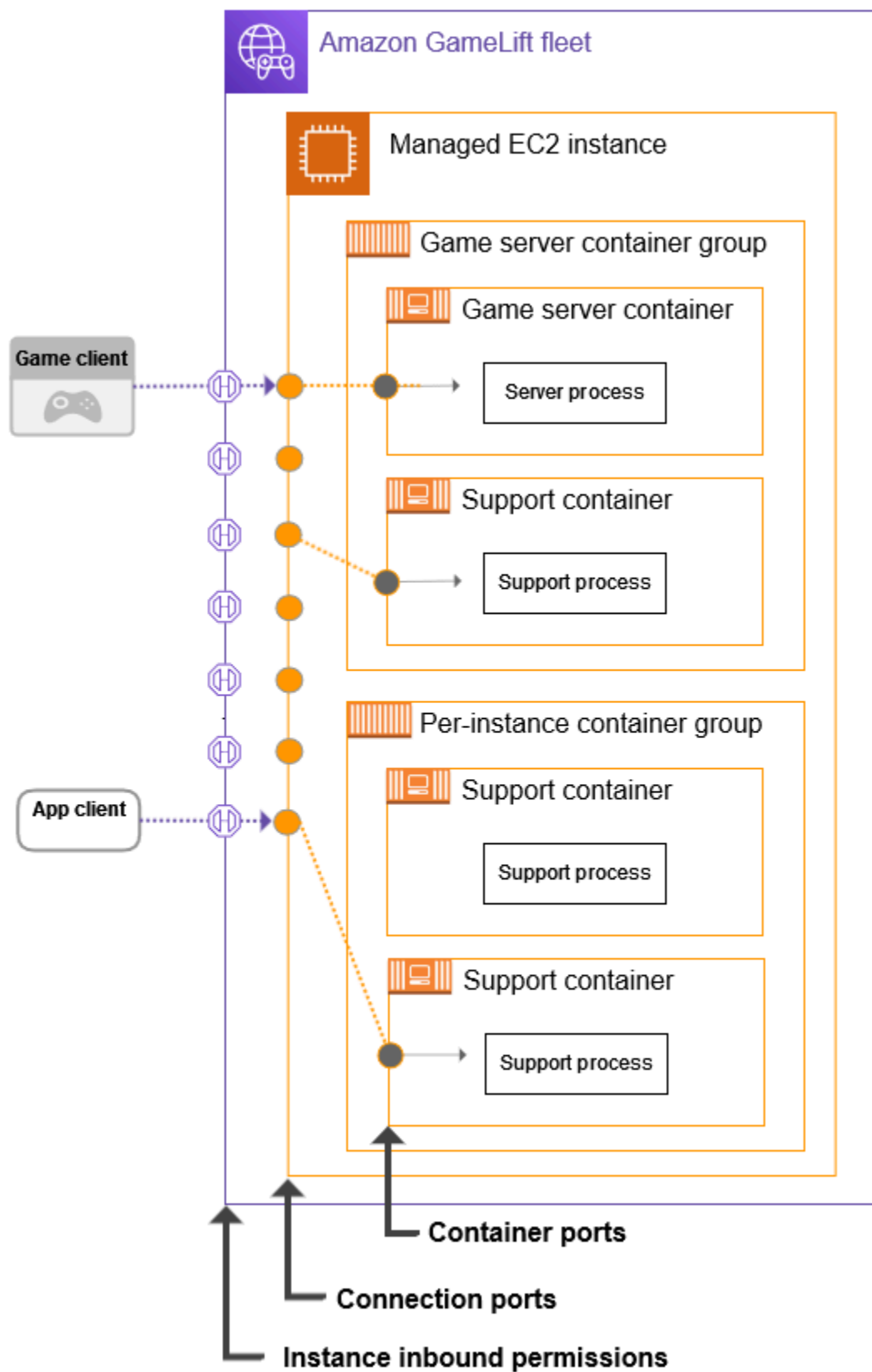
Managed container fleets require an additional setting that allows access to processes that are running in a container. When you create a container definition, you specify a set of ports, one for each process that takes a connection. This includes:

- All game server processes that will run concurrently in the game server container. All game server processes must allow game clients to connect in order to join a game session.
- Any process in a support container that an external source needs to connect to. For example, you could remotely connect to a testing app.

When you set the internal-facing container port settings, Amazon GameLift Servers uses them to calculate the external-facing inbound permissions that game clients and other applications can connect to. Amazon GameLift Servers also manages the mapping between inbound permissions and individual container ports that gives players access to a game session in a container. This internal mapping provides a layer of security by protecting your game servers from direct access to the container ports. You have the option to customize a fleet's external-facing port settings as needed. For more information about manually setting container fleet ports, see [Configure network connections](#).

You can modify a container fleet's port settings at any time. This change requires a fleet update deployment.

The following diagram illustrates the role of port connections across a container fleet. As shown, you set ports on individual containers, and Amazon GameLift Servers uses this information to configure enough ports on the fleet instance to map to each container port. Both the external-facing instance inbound permissions and the internal-facing connection ports are calculated by Amazon GameLift Servers for your fleet, unless you choose to set them manually.



Container logging

In managed container fleets, the standard output (and standard error) streams are captured for all containers. This includes your game server's game session logs. You can configure a container fleet to use one of several options to handle output streams:

- Save container output as an Amazon CloudWatch log stream. Each log stream references the fleet ID and container. If you choose this logging option for the fleet, you specify a CloudWatch log group, which organizes all the log streams from the fleet. You can then use CloudWatch features to search and analyse log data as needed.
- Save container output to an Amazon Simple Storage Service (Amazon S3) storage bucket. You can view, share, or download the content as needed.
- Turn off logging. In this scenario, container output isn't saved.

Amazon GameLift Servers sends log data from managed container fleets to the CloudWatch or Amazon S3 services in your AWS account. To view your data, use the AWS Management Console or other tools by signing in to your AWS account and working with the individual services. You extend limited access to Amazon GameLift Servers to take these actions by creating a service role for your container fleets.

You can modify a container fleet's logging configuration at any time. This change requires a fleet update deployment.

Container fleets and the Amazon GameLift Servers Agent

A commonly used container architecture runs a single process in each container. In an Amazon GameLift Servers container fleet, the game server container group has one game server container, which runs one game server process. With this architecture, Amazon GameLift Servers manages the lifecycle of the single game server process in each game server container group on a fleet instance.

If you opt to build a container architecture that runs multiple game server processes in each game server container group, you need a way to manage the lifecycle of all processes. This includes tasks such as starting, shutting down, and replacing processes as needed, managing a desired number of processes to run concurrently, and handling failure states.

You can choose to use the Amazon GameLift Servers Agent for these tasks. For a container fleet, the Agent implements runtime instructions that specify which executables to run (and how many), provide launch parameters, and set rules around game server activation. For example, runtime

instructions might tell the Agent to maintain ten game server processes for production use, and one game server process with special launch parameters for testing.

To use the Agent with your container fleets, add the Agent to your container image and include a set of runtime instructions. For more information about the Agent, see [Work with the Amazon GameLift Servers Agent](#).

Create an Amazon GameLift Servers managed container fleet

Create an Amazon GameLift Servers managed container fleet to deploy and host your containerized game server in the AWS Cloud. When you create a container fleet, specify container group definitions that specify one or more container images (at least one that includes your game server build) and configuration settings.

When you create a new managed container fleet resource, you immediately initiate the first phase of fleet creation. Managed fleet creation passes through several phases as Amazon GameLift Servers provisions an EC2 instance, installs a runtime environment, deploys your container groups onto the instance, and begins launching game server processes. Depending on the runtime environment your game server build requires, Amazon GameLift Servers deploys the latest version of the Amazon Machine Image (AMI) at the time of fleet creation (and all future instances in the fleet will use the same version). You can monitor a fleet's status in the console or using the AWS Command Line Interface (AWS CLI). When a fleet is ready to host game sessions, the status changes to ACTIVE. For help with fleet creation issues, see [Debug Amazon GameLift Servers fleet issues](#).

You can opt to create an empty container fleet and then add or update the fleet's container group definitions later. If you create a fleet without a container group definition, the fleet will not reach active status.

Note

As a best practice, we recommend replacing your fleets every 30 days to maintain a secure and up-to-date runtime environment for your hosted game servers. This requires creating a new fleet and migrating player traffic to it. For more guidance, see [Security best practices for Amazon GameLift Servers](#).

Use the [Amazon GameLift Servers console](#) or the AWS Command Line Interface (AWS CLI) to create a container fleet.

Console

In the [Amazon GameLift Servers console](#), select the AWS Region where you want to create the fleet. The container group definitions must be in the same region where you want to create the fleet.

Open the console's left navigation bar and choose **Managed containers: Fleets**. On the Fleets page, choose **Create container fleet**.

Step 1: Define managed container fleet details

1. In the **Container fleet details** section, enter a fleet description.
2. Specify an **IAM role** for the fleet. This role has permissions that Amazon GameLift Servers must have to manage the container fleet on your behalf. For help creating the required service role, see [Set up an IAM service role for Amazon GameLift Servers](#).
3. Choose a **Log configuration** option. The CloudWatch option is selected by default. Provide the required information based on your selected option.
4. Add container groups to the fleet. This is an optional step. You can choose to create a fleet without a container group with a plan to add them later. A fleet without any container groups won't deploy any fleet instances and can't host any games yet, but the fleet resource is created.
 - Select a game server container group definition. Optionally specify the version of the definition that you want to deploy. If you don't specify the version number, Amazon GameLift Servers automatically uses the latest version.
 - Optionally add a per-instance container group definition and version. If you don't specify the version number, Amazon GameLift Servers automatically uses the latest version.
5. In the **Additional details**, you can set some optional customizations. None of these settings are required to create the container fleet.

Step 2: Define instance details

1. In **Instance deployment**, select one or more remote locations to deploy instances to. The home Region is automatically selected (this is the Region that you're creating the fleet in). If you select additional locations, fleet instances are also deployed in these locations.

 Important

To use Regions that aren't enabled by default, enable them in your AWS account.

- Fleets with Regions that aren't enabled that you created before February 28, 2022 are not affected by this requirement.
- To create new multi-location fleets or to update existing multi-location fleets, first enable any Regions or Local Zones that you choose to use.

For more information about Regions that aren't enabled by default and how to enable them, see [Managing AWS Regions](#) in the *AWS General Reference*. See [Getting started with Local Zones](#) in the *AWS Local Zones User Guide*.

2. Select an **Instance configuration** for the fleet. The console automatically calculates the minimum vCPU and memory required (based on the total limits you set for each container group). It filters the complete list of available instance types base on resource requirements and the locations you entered. You can add additional filters as needed.

 Note

For Graviton ARM instances, make sure the container images in your container group definitions are built for ARM architecture. Otherwise the fleet deployment will fail.

For more information about choosing an instance type, see [Configure a container fleet](#). The size of the instance type you choose will impact how game server container groups are packed onto each fleet instance. Depending on your choice, consider reviewing your setting for desired game server container groups per instance.

Step 4: Review and create

- Review your fleet configuration settings.

You can update the fleet's metadata and configuration at any time, regardless of fleet status. For more information, see [Update an Amazon GameLift Servers fleet configuration](#).

You can update fleet capacity after the fleet has reached ACTIVE status. For more information, see [Scaling game hosting capacity with Amazon GameLift Servers](#). You can also add or remove remote locations.

When you're finished reviewing, choose **Create**.

If your request is successful, the console displays the detail page for the new fleet resource. Initially the status is NEW, as Amazon GameLift Servers starts the fleet creation process. You can track the new fleet's status on the **Fleets** page. A fleet is ready to host game sessions when it reaches status ACTIVE.

AWS CLI

To create a container fleet with the AWS CLI, open a command line window and use the `create-container-fleet` command. For more information about this command, see [create-container-fleet](#) in the *AWS CLI Command Reference*.

The example `create-container-fleet` request shown below creates a new container fleet with the following characteristics:

- The `ContainerGroupsConfiguration` specifies a game server container group definition only: `MyAdventureGameContainerGroup`. The number of game server container groups that will be deployed to each fleet instance is calculated by Amazon GameLift Servers.
- The fleet uses `c5.large` On-Demand instances by default.
- By default, the fleet opens a set of connection ports and inbound permissions ports as calculated by Amazon GameLift Servers. It deploys container groups to the following locations:

```
aws gamelift create-container-fleet \  
  --fleet-role-arn arn:aws:iam::MyAccount:role/MyContainersRole \  
  --game-server-container-group-definition-name "rn:aws:gamelift:us-  
west-2:111122223333:containergroupdefinition/MyAdventureGameContainerGroup:2" \  
  --fleet-name MyAdventureGameContainerGroup
```

If the `create-fleet` request is successful, Amazon GameLift Servers returns a set of fleet attributes that includes the configuration settings you requested and a new container fleet ID. Amazon GameLift Servers then sets the fleet status and location statuses to **New** and initiates

the fleet activation process. You can track the fleet's status and view other fleet information using these CLI commands:

- [describe-fleet-events](#)
- [describe-container-fleet](#)
- [describe-fleet-capacity](#)
- [describe-fleet-port-settings](#)
- [describe-fleet-utilization](#)
- [describe-fleet-location-capacity](#)
- [describe-fleet-location-utilization](#)

You can change the fleet's capacity and other configuration settings as needed using these commands:

- [update-container-fleet](#)
- [update-fleet-capacity](#)
- [update-fleet-port-settings](#)
- [create-fleet-locations](#)
- [delete-fleet-locations](#)

Create a container group definition for an Amazon GameLift Servers container fleet

A container group definition describes how to deploy your containerized game server applications to a container fleet. It's a blueprint that tells Amazon GameLift Servers what container images to deploy to the fleet and how to run them. When you create a container fleet, you specify the container group definitions to deploy to the fleet. For more information about container groups, see [Container fleet components](#).

Before you start

Tips on what to do before you start creating a container group definition:

- Finalize your container images and push them to an Amazon Elastic Container Registry (Amazon ECR) repository in the same AWS Region where you plan to create the container group. Amazon

GameLift Servers captures a snapshot of each image at the time you create container group definition, and uses the snapshot when deploying to a container fleet. See [Build a container image for Amazon GameLift Servers](#).

- Create your container definitions as JSON files. A container group definition includes one or more container definitions. You can use the JSON files if you create a container group definition using the AWS CLI for Amazon GameLift Servers.
- Verify that your AWS user has IAM permissions to access the Amazon ECR repository. See [IAM permission examples for Amazon GameLift Servers](#).

Create a game server container group definition

A game server container group runs your game server software. A game server container group has one game server container, which runs the game server executable. It can also have one or more support containers to run additional software to support your game server. (These are sometimes referred to as “sidecar” containers.)

This topic describes how to create a simple game server container group definition using the Amazon GameLift Servers console or AWS CLI tools. For more detailed information on optional features, see [Customize an Amazon GameLift Servers container fleet](#).

Note

You can change most container group definition and container definition settings after creating them. If you make changes to a container definition, Amazon GameLift Servers captures a new snapshot of the updated container images.

To create a simple game server container group definition:

The following instructions describe how to create a container group definition with the minimal required parameters and using the Amazon GameLift Servers default values.

Console

In the [Amazon GameLift Servers console](#), select the AWS Region where you want to create the container group.

Open the console's left navigation bar and choose **Managed containers: Group definitions**. On the Container groups definition page, choose **Create group definition**.

Step 1: Define container group definition details

1. Enter a container group definition name. The name must be unique to the AWS account and Region.
2. Select the **Game server** container group type.
3. For **Total memory limit**, enter the maximum memory resources to make available for all containers in the container group. For help calculating this value, see [Set resource limits](#).
4. For **Total vCPU limit**, enter the maximum computing power to make available for all containers in the container group. For help calculating this value, see [Set resource limits](#).

Step 2: Add container definitions

At minimum, a game server container group has one game server container. In the console, the first container definition you create is the game server container. This step describes how to define the minimum required settings for a game server container definition.

1. Enter a container definition **Name**. Each container defined for the group must have a unique name value.
2. Link to a container image with your game server build. Enter the **Amazon ECR image URI** for a container image in a public or private repository. You can use any of the following formats:
 - Image URI only: [AWS account].dkr.ecr.[AWS Region].amazonaws.com/[repository ID]
 - Image URI + digest: [AWS account].dkr.ecr.[AWS Region].amazonaws.com/[repository ID]@[digest]
 - Image URI + tag: [AWS account].dkr.ecr.[AWS Region].amazonaws.com/[repository ID]:[tag]
3. Specify the Amazon GameLift Servers **Server SDK version** that the game server build uses. For a container fleet, this value must be 5.2.0 or greater.
4. In **Internal container port range**, set the protocol and define a port range. The range size must be greater than the number of concurrent game server processes that will run in this container. If the game server container runs only one server process per container, this port range only needs a few ports. For more details, see [Configure network connections](#).
5. Add more containers as needed to run additional support software. Additional containers are automatically designated support containers. A game server container group can have

only one game server containers and up to eight support containers. Provide the following minimal required settings:

- Container definition **Name**
- **ECR image URI**.
- **Internal container ports** (Include this only if the container has processes that need network access.)

Step 3: Configure dependencies

- If your container group definition has more than one container, you can optionally set dependencies between the containers. For more information, see [Set container dependencies](#).

Step 3: Review and create

1. Review all your container group definition settings. Use **Edit** to make changes to any section, including each of your container definitions for the group.
2. When you're finished reviewing, choose **Create**.

If your request is successful, the console displays the detail page for the new container group definition resource. Initially the status is `COPYING`, as Amazon GameLift Servers starts taking snapshots of all the container images for the group. When this phase is complete, the container group definition status changes to `READY`. A container group definition must be in `READY` status before you can create a container fleet with it.

AWS CLI

When you use the AWS CLI to create a container group definition, maintain your container definition configurations in a separate JSON file. You can reference the file in your CLI command. See [Create a container definition JSON file](#) for schema examples.

Create a container group definition

To create a new container group definition, use the `create-container-group-definition` CLI command. For more information about this command, see [create-container-group-definition](#) in the *AWS CLI Command Reference*.

This example illustrates a request for a game server container group definition. It assumes that you've created a JSON file with the container definitions for this group.

```
aws gamelift create-container-group-definition \  
  --name MyAdventureGameContainerGroup \  
  --operating-system AMAZON_LINUX_2023 \  
  --container-group-type GAME_SERVER \  
  --total-memory-limit-mebibytes 4096 \  
  --total-vcpu-limit 1 \  
  --game-server-container-definition file://MyAdventureGameContainers.json
```

Create a container definition JSON file

When you create a container group definition, you also define the containers for the group. A container definition specifies the Amazon ECR repository where the container image is stored, and optional configurations for network ports, limits for CPU and memory usage, and other settings. We recommend creating a single JSON file with the configurations for all the containers in a container group. Maintaining a file is useful for storing, sharing, version tracking these critical configurations. If you use the AWS CLI to create your container group definitions, you can reference the file in the command.

To create a container definition

1. Create and open a new .JSON file. For example:

```
[~/work/glc]$ vim SimpleServer.json
```

2. Create a separate container definition for each of the containers for the group. Copy the following example content and modify it as needed for your containers. For details on the syntax of a container definition, see [ContainerDefinitionInput](#) in the *Amazon GameLift Servers API Reference*.
3. Save the file locally so that you can refer to it in an AWS CLI command.

Example: Game server container definition

Example

This example describes the essential container for your game server container group. The essential replica container includes your game server application, the Amazon GameLift Servers Agent, and

can include other supporting software for your game hosting. The definition must include a name, image URI, and a port configuration. This example also sets some container-specific resource limits.

```
{
  "ContainerName": "MyAdventureGameServer",
  "ImageUri": "111122223333.dkr.ecr.us-east-1.amazonaws.com/gl-
containers:myadventuregame-server",
  "PortConfiguration": {
    "ContainerPortRanges": [
      {
        "FromPort": 2000,
        "Protocol": "TCP",
        "ToPort": 2010
      }
    ]
  },
  "ServerSdkVersion": "5.2.0"
}
```

Discover containers on a container fleet

On an Amazon GameLift Servers container fleet, each instance can run multiple containers. For example, you can run several replicas of your game server container group. You can also run a per-instance container group that provides shared, instance-wide services.

The container discovery server enables your game server processes to discover the other containers running on the same instance. Containers can then communicate with one another across container group definitions over the instance's local network.

The discovery server is a lightweight HTTP service. Amazon GameLift Servers runs it locally on every container fleet instance. The service provides the following network information for each container running on that instance:

- The container name that you define in the container group definition.
- The unique identifier of the container.
- The container's local IP address on the instance.
- The container group type, either `GAME_SERVER` (a replica of the game server container group) or `PER_INSTANCE` (the per-instance container group).

Note

The container discovery server is only available on managed container fleets.

Retrieve container network information

You can retrieve container network information in two ways: from a game server process with the Amazon GameLift Servers server SDK, or from any container by calling the discovery server directly over HTTP.

Retrieve container information using the server SDK

If your game server process uses the Amazon GameLift Servers server SDK, call the `ListContainersNetworkInfo()` action to retrieve container network information. The server SDK locates the discovery server and makes the request for you. The action returns the network information for every container running on the same instance. On any other compute type, the action returns an unsupported compute type error.

`ListContainersNetworkInfo()` is available in server SDK version 5.5 or later for the following languages:

- [ListContainersNetworkInfo\(\)](#) (Go)
- [ListContainersNetworkInfo\(\)](#) (C++)
- [ListContainersNetworkInfo\(\)](#) (C#)

The action is also available in the Amazon GameLift Servers plugins for Unreal Engine and Unity.

Retrieve container information by calling the discovery server directly

Some containers don't use the server SDK, such as per-instance containers. These containers call the discovery server directly over HTTP. Amazon GameLift Servers pre-sets the `GAMELIFT_CONTAINER_DISCOVERY_SERVER_ENDPOINT_V1` environment variable in these containers. This variable holds the address of the discovery server, including the API version path, and it's set in both the game server and per-instance container group definitions.

To retrieve container network information, send an HTTP GET request to this endpoint. The following example uses `curl`:

```
curl "$GAMELIFT_CONTAINER_DISCOVERY_SERVER_ENDPOINT_V1"
```

The discovery server returns a JSON array with one entry for each container running on the instance:

```
[
  {
    "containerName": "game-server",
    "ipAddress": "172.17.0.3",
    "containerId": "b896159563df43736e5e93ad0547a24fa572648d55b70c70747642afca398b18",
    "containerGroupType": "GAME_SERVER"
  },
  {
    "containerName": "per-instance",
    "ipAddress": "172.17.0.4",
    "containerId": "3c239ae037aec28bb42212ffa2b9e02524c0c4c361db9bc31dd4c5aecc0b7943",
    "containerGroupType": "PER_INSTANCE"
  }
]
```

Request rate limits

The discovery server limits each container on the instance to 10 requests per second. It throttles requests that exceed this limit and returns an error. To stay within this limit, query the discovery server once at startup. As an alternative, query it only when you need to refresh your view of the instance. Cache the results in your container and reuse the cached values rather than querying the discovery server on a recurring basis.

Periodically refresh your cache to keep it current. A container's local IP address can change if the container crashes or exits (for example, when its game session ends).

Example: collecting metrics from game server containers

A common use of the discovery server is to send telemetry between containers on the same instance without leaving the instance's local network. Consider a fleet that runs multiple game server containers (GAME_SERVER) and a single per-instance container (PER_INSTANCE). The per-instance container runs a metrics collector that all of the game servers on the instance share.

You can implement metrics collection with either a push model or a pull model:

- **Push (metrics emission)** – At startup, each game server container calls `ListContainersNetworkInfo()`. The container finds the entry with the `PER_INSTANCE` container group type. It then reads the local IP address from that entry. The game server sends its metrics directly to the per-instance collector over the instance's local network.
- **Pull (metrics scraping)** – The reverse approach also works. The per-instance container doesn't use the server SDK, so it calls the discovery server endpoint directly. It finds the IP addresses of all `GAME_SERVER` containers on the instance. It then scrapes a metrics endpoint that each game server container exposes over the local network.

In both cases, communication stays on the instance's local network. Each container queries the discovery server only occasionally rather than on a recurring basis.

Related topics

- [Add Amazon GameLift Servers to your game server with the server SDK](#)
- [Create a container group definition for an Amazon GameLift Servers container fleet](#)
- [Customize an Amazon GameLift Servers container fleet](#)
- [How containers work in Amazon GameLift Servers](#)

Amazon GameLift Servers Anywhere fleets

Use Anywhere fleets when you want to take advantage of Amazon GameLift Servers features with your own hosting resources. Anywhere fleets are commonly used as test environments for iterative development or alongside managed fleets in a hybrid hosting solution.

An Anywhere fleet consists of a set of compute resources (virtual or physical) that you supply and manage. Computes can reside in any geographic location with connectivity, from a local laptop to remote outposts. When setting up an Anywhere fleet, you add computes to the fleet by registering them through Amazon GameLift Servers. Each compute is registered with its IP address (or DNS name) so that Amazon GameLift Servers can establish a connection with it.

You deploy game server software to an Anywhere fleet by installing it on each compute and launching game server processes. Each launched game server process establishes a connection to the Amazon GameLift Servers service and reports readiness to host a game session. You can use your existing configuration management and deployment tooling to handle initial deployment and host management tasks. There are a few additional tasks required for use with Amazon GameLift Servers, including:

- Register and deregister computes to add or remove them from the fleet.
- Maintain up-to-date authentication tokens on all computes. Server processes on the compute use them when connecting to the Amazon GameLift Servers service.

Note

Optionally deploy your Anywhere fleet with the Amazon GameLift Servers Agent to automate these key management tasks. See [Work with the Amazon GameLift Servers Agent](#).

See these topics about how to set up and maintain Anywhere fleets:

- [Development roadmap for hosting with Amazon GameLift Servers Anywhere](#)
- [Development roadmap for hybrid hosting with Amazon GameLift Servers](#)
- [Set up for iterative development with Amazon GameLift Servers Anywhere](#)
- [Create an Amazon GameLift Servers Anywhere fleet](#)
- [Update an Amazon GameLift Servers fleet configuration](#)

Anywhere fleet creation workflow

For Anywhere fleets, Amazon GameLift Servers sets up the fleet resource only. You set up and register computes with the fleet, and you install game server software and start game server processes to host game sessions.

1. Amazon GameLift Servers creates the fleet resource in the fleet's home Region. Fleet status and custom location status are set to **New**.
2. Amazon GameLift Servers begins writing events to the fleet event log.
3. After the fleet resource is created, Amazon GameLift Servers sets the fleet status to **Active**. At this point, you can register new computes with the fleet.

Create an Amazon GameLift Servers Anywhere fleet

This topic describes how to create an Amazon GameLift Servers Anywhere fleet. With an Anywhere fleet, you can use core Amazon GameLift Servers game session management features while

hosting game sessions with your own compute resources. Create an Anywhere fleet for your on-premises hardware or other cloud-based resources.

Anywhere fleets are commonly used alongside Amazon GameLift Servers managed fleets in a hybrid hosting solution. They also provide useful test environments when developing a game for hosting with Amazon GameLift Servers. See these topics to learn more about when and how to incorporate Amazon GameLift Servers Anywhere fleets into a game hosting solution:

- [Hybrid hosting](#)
- [Deploy hosting fleets for Amazon GameLift Servers](#)
- [Set up for iterative development with Amazon GameLift Servers Anywhere](#)

Because Anywhere fleets are self-managed, setting up a fleet requires some additional work. To get an Anywhere fleet ready to host game sessions and players, you need to complete the following tasks:

Topics

- [Before you start](#)
- [Create a custom location](#)
- [Create an Anywhere fleet](#)
- [Add a compute to the fleet](#)
- [Start a game server](#)

Before you start

Before creating an Anywhere fleet, do the following tasks. For more detailed guidance, see the [Development roadmap for hosting with Amazon GameLift Servers Anywhere](#) or [Development roadmap for hybrid hosting with Amazon GameLift Servers](#).

- **Integrate your game server code with the Amazon GameLift Servers server SDK version 5.x (or higher).** You don't need to complete all game integration tasks, just those required for a game server build. A common practice is to set up your local machine as an Anywhere fleet and use a command line interface to test your game server integration (see [Set up local testing with Amazon GameLift Servers Anywhere](#)). You can incorporate additional components (such as an Amazon GameLift Servers enabled game client) as you develop them.

- **Package your game server software for installation onto your Anywhere fleet computes.** The package should include your integrated game server build and all support software needed to run your game server.
- **Decide whether to use the Amazon GameLift Servers Agent with your Anywhere fleet.** The Agent is an on-compute process management tool that automates some of the key tasks related to managing server processes and computes for use with Amazon GameLift Servers. For more information, see [Work with the Amazon GameLift Servers Agent](#).

Create a custom location

Create a custom location to represent the physical location of your compute resources. When creating an Anywhere fleet, you must have at least one custom location already defined. You can create additional custom locations and add them to an existing fleet at any time.

To create a custom location

Use either the Amazon GameLift Servers console or the AWS Command Line Interface (AWS CLI) to create a custom location.

Console

In the [Amazon GameLift Servers console](#), use the navigation pane to open the **Locations** page. Choose **Create location** to open the Create dialog box.

1. In the dialog box, enter a **Location name**. As a best practice, use a name that describes a meaningful location for a set of compute resources. It might be geographic locations, a data center name, or other location identifier. Amazon GameLift Servers appends the name of your custom location with **custom-**.
2. (Optional) Add tags to your custom location. Each tag consists of a key and an optional value, both of which you define. Assign tags to AWS resources that you want to categorize in useful ways, such as by purpose, owner, or environment. Choose **Add new tag** for each tag that you want to add.
3. Choose **Create**.

AWS CLI

Create a custom location using the [create-location](#) command. Provide a `location-name` value, which must start with `custom-`. As a best practice, use a name that describes a

meaningful location for a set of compute resources. It might be geographic locations, a data center name, or other location identifier.

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

Output

```
{  
  "Location": {  
    "LocationName": "custom-location-1",  
    "LocationArn": "arn:aws:gamelift:us-east-1:111122223333:location/custom-  
location-1"  
  }  
}
```

Create an Anywhere fleet

Create an Anywhere fleet for a set of compute resources that you own. A new Anywhere fleet starts empty; you add computes to the fleet by registering them.

On creation, a new Anywhere fleet quickly moves through fleet statuses from NEW to ACTIVE. You can add computes to the fleet after it reaches ACTIVE.

To create an Anywhere fleet

Use either the Amazon GameLift Servers console or the AWS Command Line Interface (AWS CLI) to create an Anywhere fleet.

Console

In the [Amazon GameLift Servers console](#), use the navigation pane to open the **Fleets** page. Choose **Create fleet** to start the fleet creation workflow.

Step 1 Choose compute type

Select the **Anywhere** option and choose **Next**.

Step 2 Define fleet details

In this step, specify some key fleet-wide settings.

1. Fill out the **Fleet details** section:
 - a. Enter a fleet **Name**. We recommend using a fleet naming pattern that makes it easier to identify fleet types when viewing lists of fleets.
 - b. Provide a short **Description** of the fleet.
2. Set these optional **Additional details** as needed. You can update these fleet settings later.
 - a. When creating a fleet for production or pre-prod testing, use this setting to specify a per-hour **Cost** value for the fleet's computes. Amazon GameLift Servers can use this information during the game session placement process to select hosting resources based on cost.
 - b. If you want to combine metric data for this fleet and others, specify a **Metric group** name. Use the same metric group name for all fleets that you want to combine together. View metrics for the metric group to see the aggregated data.
3. Add optional tags to your custom location. Each tag consists of a key and an optional value, both of which you define. Assign tags to AWS resources that you want to categorize in useful ways, such as by purpose, owner, or environment. Choose **Add new tag** for each tag that you want to add.
4. Choose **Next** to continue the workflow.

Step 3 Select custom locations

In this step, identify the physical location of the computes that you plan to add to this fleet. You can specify one or more locations now, and you can add or remove locations later as needed.

1. In **Custom locations**, select one or more locations for the fleet's computes. The list includes all custom locations that have been defined in your currently selected AWS Region. To define a new custom location that you want to add to the fleet, choose **Create location**.
2. Choose **Next** to continue the workflow.

Step 4 Review and create

Review your settings before creating the fleet.

When you're ready to deploy the new fleet, choose **Create**. Amazon GameLift Servers immediately begins the fleet activation process, assigning a unique ID and placing the fleet in NEW status. You can track the fleet's progress on the **Fleets** page.

AWS CLI

Use the [create-fleet](#) command to create a fleet of compute type ANYWHERE. Provide a name and at least one custom location. Amazon GameLift Servers creates the Anywhere fleet resource in your current default AWS Region (or you can add a `--region` tag to specify a different AWS Region).

The following example request creates a new fleet with the minimal required settings. Replace *FleetName* and *custom-location* with your own information.

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

Example response

```
{  
  "FleetAttributes": {  
    "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",  
    "FleetArn": "arn:aws:gamelift:us-west-2:111122223333:fleet/  
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",  
    "Name": "HardwareAnywhere",  
    "CreationTime": "2023-02-23T17:57:42.293000+00:00",  
    "Status": "ACTIVE",  
    "MetricGroups": [  
      "default"  
    ],  
    "CertificateConfiguration": {  
      "CertificateType": "DISABLED"  
    },  
    "ComputeType": "ANYWHERE"  
  }  
}
```

On creation, a new Anywhere fleet quickly moves to fleet status ACTIVE. You can add computes to the fleet after it reaches ACTIVE.

Notice that the response doesn't include the fleet locations. You can retrieve full fleet details by calling [describe-fleet-attributes](#) and [describe-fleet-location-attributes](#).

Add a compute to the fleet

To add a compute resource to a fleet and get it ready to host game sessions, do the following tasks:

- Register the compute with the fleet. Registration tells Amazon GameLift Servers what physical hosting resources are part of the fleet.
- Request an authentication token for the compute. Each game server that runs on the compute needs this token to connect to the Amazon GameLift Servers service. Authentication tokens are temporary and must be regularly refreshed.

Note

If you're deploying your game server software with the Amazon GameLift Servers Agent, you can skip this step. The Agent automatically registers each compute and maintains a valid authentication token for the compute. See [Work with the Amazon GameLift Servers Agent](#).

You can register a compute and request an authentication token by using the AWS CLI or making programmatic calls to the AWS SDK for Amazon GameLift Servers. These actions are not available through the Amazon GameLift Servers console.

As a best practice, we recommend automating both of these tasks by adding a startup script to each compute. The startup script automatically calls both the `register-compute` and `get-compute-auth-token` commands. You can also automate tasks to regularly refresh the auth token throughout the life of the compute and deregister the compute on shut down.

Each of the startup actions returns compute-specific values that you need to store on the compute. When a game server process launches on the compute, it must pass these values as server parameters when initializing a connection with the Amazon GameLift Servers service (see [ServerParameters](#) in the server SDK reference). We recommend that you set these compute-specific values (or their stored locations) as environment variables. If you're using the Amazon GameLift Servers Agent, this task is handled for you. The compute-specific values are as follows:

- `register-compute` returns a value for `GameLiftServiceSdkEndpoint`. Set this value to the `webSocketUrl` server parameter.
- `compute-auth-token` returns the authentication token. Set this value to the `authToken` server parameter.

AWS CLI

The following instructions describe how manually submit each request using the AWS CLI.

To register a compute

Call [register-compute](#) to register a compute. Identify the ID of the fleet to add the compute to. Provide the following compute information: a meaningful name, IP address, and location. The compute's location must be a custom location that's already associated with the fleet. If you want to use a different custom location, use the Amazon GameLift Servers console to update the fleet or call the AWS CLI command [create-fleet-locations](#) to add a custom location to the fleet.

In the following example, replace the placeholder values for your compute and fleet. The `fleet-id` value is returned when you create an Anywhere fleet. You can retrieve full fleet details by calling [describe-fleet-attributes](#) and [describe-fleet-location-attributes](#).

```
aws gamelift register-compute \  
  --compute-name HardwareAnywhere \  
  --fleet-id arn:aws:gamelift:us-east-1:111122223333:fleet/  
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \  
  --ip-address 10.1.2.3 \  
  --location custom-location-1
```

Example output

```
{  
  "Compute": {  
    "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",  
    "FleetArn": "arn:aws:gamelift:us-west-2:111122223333:fleet/  
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",  
    "ComputeName": "HardwareAnywhere",  
    "ComputeArn": "arn:aws:gamelift:us-west-2:111122223333:compute/  
HardwareAnywhere",
```

```

    "IpAddress": "10.1.2.3",
    "ComputeStatus": "Active",
    "Location": "custom-location-1",
    "CreationTime": "2023-02-23T18:09:26.727000+00:00",
    "GameLiftServiceSdkEndpoint": "wss://us-west-2.api.amazongamelift.com"
  }
}

```

To request an authentication token

Call [get-compute-auth-token](#) to request a valid authentication token. register a compute. Identify the fleet ID and compute name.

In the following example, replace the placeholder values for your compute and fleet. The `fleet-id` value is returned when you create an Anywhere fleet. You can retrieve full fleet details by calling [describe-fleet-attributes](#). To find compute information, call [list-compute](#) with the fleet ID to see all computes that are registered to the fleet.

```

aws gamelift get-compute-auth-token \
  --fleet-id arn:aws:gamelift:us-east-1:111122223333:fleet/
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \
  --compute-name HardwareAnywhere

```

Example output

```

{
  "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
  "FleetArn": "arn:aws:gamelift:us-east-1:111122223333:fleet/
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
  "ComputeName": "HardwareAnywhere",
  "ComputeArn": "arn:aws:gamelift:us-east-1:111122223333:compute/
HardwareAnywhere",
  "AuthToken": "0c728041-3e84-4aaa-b927-a0fb202684c0",
  "ExpirationTimestamp": "2023-02-23T18:47:54+00:00"
}

```

Start a game server

After you've created an Anywhere fleet and added one or more computes to the fleet, you're ready to start running your game servers.

Step 1 Install your game server software

Get your game server build and all dependent software installed onto each compute in your Anywhere fleet. The game server build must be integrated with Amazon GameLift Servers server SDK version 5.x (or higher) with the minimum required functionality to communicate with the Amazon GameLift Servers service.

Step 2 Get your computes ready to run a game server

Ensure that each compute is registered and has a valid authentication token. If you're using scripts to manage these tasks, make sure that the scripts run on each compute before starting any game server processes.

If you've deployed the Amazon GameLift Servers Agent with your game server software, make sure that the Agent executable launches.

Step 3 Launch a game server process

Run an instance of your game server executable on a compute. If your game server build is properly integrated, the game server process calls the server SDK action `InitSDK()` with a set of valid server parameters. When the server process is ready to host a game session, it calls `ProcessReady()`.

Note

If you deployed your game server software with the Amazon GameLift Servers Agent, you can skip this step. The Agent automatically launches game server processes based on the runtime instructions you provide.

You can monitor progress by viewing server process metrics for activating and active server processes. See [Amazon GameLift Servers metrics for fleets](#). If your game server process fails to initialize, verify that the process is retrieving the right server parameter values for the compute it's running on.

Build a hybrid hosting solution

A hybrid hosting solution combines multiple sources of game hosting resources to host your game, including Amazon GameLift Servers managed fleets that run in the AWS Cloud and resources that you supply and manage on your own. This topic describes some common patterns for building a

hybrid solution, and provides tips on how to successfully combine self-managed game hosting with cloud-based game hosting managed by Amazon GameLift Servers.

Common hybrid patterns

- **Cost optimization:** Use the most cost-effective fleet type for each scenario, such as Anywhere fleets for baseline capacity and managed fleets for peak demand.
- **Geographic flexibility:** Deploy managed fleets in high-traffic AWS Regions and Anywhere fleets where you have existing infrastructure or specific compliance requirements.
- **Risk mitigation:** Reduce dependency on any single hosting approach by distributing load across multiple fleet types and providers.
- **Gradual migration:** Transition from on-premises or other hosting solutions to AWS gradually while maintaining service continuity.

Development and production split

Use different fleet types for development and production environments:

- **Development:** Anywhere fleets for cost-effective development and testing
- **Production:** Managed Amazon EC2 or container fleets for scalable, reliable production hosting

Regional optimization

Optimize fleet types based on AWS Regions characteristics:

- **High-traffic AWS Regions:** Managed fleets with auto-scaling for variable demand
- **Specialized AWS Regions:** Anywhere fleets for compliance, data sovereignty, or existing infrastructure

Capacity tiering

Use different fleet types for different capacity tiers:

- **Baseline capacity:** Anywhere fleets or reserved instances for predictable load
- **Burst capacity:** Managed fleets with auto-scaling for peak demand
- **Overflow capacity:** Spot instances or additional AWS Regions for extreme peaks

Implementation considerations

When building a hybrid solution, consider these key factors:

Game session queue configuration

Configure your game session queues to include all fleet types and set appropriate priorities and latency preferences to ensure optimal placement across your hybrid infrastructure.

Monitoring and observability

Implement comprehensive monitoring across all fleet types to maintain visibility into performance, capacity, and costs across your hybrid solution.

Operational complexity

Account for the increased operational complexity of managing multiple fleet types, including different deployment processes, monitoring tools, and troubleshooting procedures.

Network connectivity

Ensure reliable network connectivity between your different hosting environments, especially for Anywhere fleets that may be on-premises or in different cloud providers.

Getting started with hybrid hosting

To implement a hybrid hosting solution:

1. **Start simple:** Begin with a single fleet type and add others gradually as your requirements become clearer.
2. **Plan your architecture:** Design your hybrid architecture based on your specific requirements for cost, performance, compliance, and operational complexity.
3. **Configure queues:** Set up game session queues that span your different fleet types with appropriate priorities and placement strategies.
4. **Test thoroughly:** Test game session placement and failover scenarios across your hybrid infrastructure before going to production.
5. **Monitor and optimize:** Continuously monitor performance and costs across all fleet types and adjust your configuration as needed.

Tips

- Use the same game client and server components with both managed and self-managed hosting resources. Provide a unified player experience across all hosting resources.
- Use the same FlexMatch matchmakers to place matches across all hosting resources.
- Centrally manage your hybrid hosting resources together while you deploy them across the globe.
- As player demand fluctuates, manage game session loads seamlessly across managed and self-managed resources.
- With the Amazon GameLift Servers Agent, you can use the same tools to manage game server life cycles on all types of hosting resources.
- Gather game and player metrics and logs across all hosting resources. Take advantage of Amazon GameLift Servers features and other AWS services to combine data and develop cohesive observability solutions.

Configure game session placement

Game session placement is the process of finding available game servers to host new game sessions. Amazon GameLift Servers uses game session queues to intelligently place game sessions across your fleets, taking into account factors like player latency, fleet capacity, and cost optimization.

A well-configured game session queue ensures that players are connected to the best available game server, providing optimal performance while making efficient use of your hosting resources. Queues can span multiple fleets and AWS Regions to provide global coverage and redundancy.

A game session queue is the primary mechanism that Amazon GameLift Servers uses to search for available game servers and choose them to host new game sessions. Queues offers a far more efficient way to process large numbers of game session requests and find placements for them across multiple fleets of hosting resources. If your hosting solution uses more than one fleet, and you're processing high volumes of requests, you probably need a queue.

When your game wants to start a new game session for players, it sends a placement request to the Amazon GameLift Servers service, which funnels it to the queue. The queue's configuration determines when and how the requests are processed. When processing a placement request, Amazon GameLift Servers searches a set of fleets for a game server to host the game session. Placement succeeds when Amazon GameLift Servers finds an available game server and prompts it to start a game session.

Topics

- [Queue characteristics](#)
- [Best practices for Amazon GameLift Servers game session queues](#)
- [Create a game session queue](#)
- [Set up event notification for game session placement](#)

Queue characteristics

An Amazon GameLift Servers game session queue is an AWS cloud resource. You can create a queue in any AWS Region that Amazon GameLift Servers supports (see [Amazon GameLift Servers service locations](#)). Game session placement requests are sent to that location and processed there.

Automating game session placement with queues offers significant benefits for both game developers and players. These include:

- **Queues deliver the "best possible" placement.** When processing game session placement requests, a queue uses the Amazon GameLift Servers FleetIQ algorithm to prioritize placements based on a set of defined preferences, including cost, location, and player latency.
- **Queues support Spot fleets to help reduce game hosting costs.** You can configure your queues with AWS Spot fleets, which often offer significantly lower hosting costs, as well as On-Demand fleets. Because low cost is one of the key criteria for placements, queues can always take advantage of differences in cost.
- **Queues can place new games faster during high demand.** By configuring a queue with multiple fleets, you're providing more flexible options for game session placement. But additional fleets also provide backup capacity as needed when demand increases. For any placement request, if Amazon GameLift Servers can't place a game session in the most preferred location, it automatically moves on to evaluate other locations.
- **Queues can make game server availability more resilient.** Outages can happen. With a multi-fleet queue, a slowdown or outage doesn't have to affect player access to your game. By configuring your queue with fleets that have capacity in different AWS Regions and availability zones, you can help make sure that players can always find a game session to join.
- **Get metrics on game session placements and queue performance.** Amazon GameLift Servers emits queue metrics, including statistics on placement successes and failures, the number of requests in the queue, and average time that requests spend in the queue. You can view these metrics in the Amazon GameLift Servers console or in CloudWatch.

To get started by creating a basic starter queue, see [Create a game session queue](#).

Topics

- [Best practices for Amazon GameLift Servers game session queues](#)
- [Create a game session queue](#)
- [Set up event notification for game session placement](#)

Best practices for Amazon GameLift Servers game session queues

A game session queue contains a list of fleets where Amazon GameLift Servers can place new game sessions. Each fleet can have hosting resources deployed in multiple geographic locations. When choosing a placement, the queue selects a fleet and a fleet location based on a set of priorities that you set for the fleet.

Consider the following guidelines and best practices:

- **Add fleets in locations that cover your players.** You can add fleets and aliases in any available location. Location is important if you're making placements based on reported player latency.
- **Use aliases for all fleets.** Assign an alias to each fleet in a queue, and use the alias names when setting destinations in your queue.
- **Use the same or a similar game build or script for all fleets.** The queue might put players into game sessions on any fleet in the queue. Players must be able to play in any game session on any fleet.
- **Create fleets in at least two locations.** By having game servers hosted in at least one other location, you mitigate the impact of Regional outages on your players. You can keep your backup fleets scaled down, and use autoscaling to increase capacity if usage increases.
- **Prioritize your game session placement.** A queue prioritizes placement choices based on several elements, including destination list order.
- **Create your queue in the same location as your client service.** By putting your queue in a location near your client service, you can minimize communication latency.
- **Use fleets with multiple locations.** Use the queue filter configuration to prevent the queue from placing game sessions in specified locations. You can use at least two multi-location fleets with different home locations to mitigate the impact of game placements during a Regional outage.
- **Use the same TLS certificate setting for all fleets.** Game clients that connect to game sessions in your fleets must have compatible communication protocols.

Create a game session queue

Queues are used to place new game sessions across multiple fleets and locations. Your game starts new game sessions by submitting placement requests to a queue. A queue is configured with instructions for how to process requests. Learn more about initiating game session placement requests in [Create game sessions](#).

To create a game session queue

These instructions illustrate how to create a simple working queue with minimal configuration settings and default settings. There are a number of options for customizing a queue configuration. These options help you make the best possible placements based on the needs of your game. To learn more about customizing queues for your game, see [Customize a game session queue](#). You can update most queue configuration settings at any time.

You can create a game session queue using either the Amazon GameLift Servers console or the AWS CLI.

Console

In the [Amazon GameLift Servers console](#), select an AWS Region to work in. Open the console's left navigation bar and choose **Queues**.

1. On the **Queues** page, choose **Create queue** to start the workflow.
2. Under **Queue settings** enter the following settings:
 - a. Enter a queue name. This name must be unique to the AWS Region that you're creating the queue in.
 - b. Keep the default **Timeout** setting, which is 600 seconds (or 10 minutes). This value controls how long Amazon GameLift Servers tries to place a new game session before stopping. Amazon GameLift Servers searches for available resources until the request times out. You can update a queue's timeout setting at any time.
 - c. Skip the **Player latency policies** section. A queue uses latency policies only when it receives placement requests that include player latency data. You can add latency policies to a queue at any time. For more on creating latency policies, see [Create a player latency policy](#).
3. Skip the **Game session placement locations** section to use the default setting of **All locations**. This setting lets you create an allow list of locations where the queue can make

placements (also called a filter configuration). For more about prioritizing by location and filter configurations, see [Prioritize placements by location](#).

4. Under **Destination order**, add one or more fleets to the queue. You can identify fleets by using fleet IDs or ARNs, or by using a fleet alias. When adding multiple fleets, keep in mind that they should all be running similar game builds and be compatible with any game client that uses this queue. In addition, all fleets in a queue must have the same certificate configuration.
 - a. Select the **Region** where the fleet or alias was created. For a multi-location fleet, this is the "home" region.
 - b. For destination **Type**, select either a fleet or an alias.
 - c. Your region and type selections populate a dropdown list of existing fleets or aliases. Select one to designate as a queue destination.
 - d. To specify another fleet or alias for the queue, choose **Add destination** and repeat the previous steps.
 - e. After you've added a list of destinations, use the drag-and-drop feature to reorder the destinations. Amazon GameLift Servers uses this order when prioritizing placements by destination.
5. Skip the **Game session placement priority** section to keep the default priority order. This setting lets you customize how Amazon GameLift Servers chooses where to look for available hosting resources for new game session placements. For more information about prioritizing placements, see [Prioritize game session placement](#). You can update a queue's placement priorities at any time.
6. Under **Location order**, keep the default values. This setting is used when prioritizing by fleet location. It provides location order to use. When using the default priority settings, location is used as a tiebreaker when the preferred destination is a fleet with multiple locations.
7. Skip the optional **Event notification settings** section. Event notifications are required for queues that process a high volume of placement requests. For queues that process low volumes, such as for development or testing purposes, you can track the status of placement requests by polling with [DescribeGameSessionPlacement](#). For more details, see [Set up event notification for game session placement](#). You can update a queue's event notification settings at any time.
8. Choose **Create** to generate a new queue with minimal customization.

AWS CLI

Example Create a queue

The following example creates a game session queue with these configurations:

- A five minute timeout.
- Two fleet destinations.
- Filter to only allow placements in these locations: us-east-1, us-east-2, us-west-2, and ca-central-1.
- Priority order based on cost and then locations in a specified order.

```
aws gamelift create-game-session-queue \  
  --name "sample-test-queue" \  
  --timeout-in-seconds 300 \  
  --destinations DestinationArn="arn:aws:gamelift:us-east-1:111122223333:fleet/  
fleet-772266ba-8c82-4a6e-b620-a74a62a93ff8" DestinationArn="arn:aws:gamelift:us-  
east-1:111122223333:fleet/fleet-33f28fb6-aa8b-4867-85b4-ceb217bf5994" \  
  --filter-configuration "AllowedLocations=us-east-1, ca-central-1, us-east-2, us-  
west-2" \  
  --priority-configuration PriorityOrder="COST","LOCATION",LocationOrder="us-  
east-1","us-east-2","ca-central-1","us-west-2" \  
  --notification-target "arn:aws:sns:us-east-1:111122223333:gamelift-test.fifo"
```

Note

You can get fleet and alias ARN values by calling either [describe-fleet-attributes](#) or [describe-alias](#) with the fleet or alias ID.

If the create-game-session-queue request is successful, Amazon GameLift Servers returns a [GameSessionQueue](#) object with the new queue configuration. You can now submit requests to the queue using [StartGameSessionPlacement](#).

Example Create a queue with player latency policies

The following example creates a game session queue with these configurations:

- A ten minute timeout

- Three fleet destinations
- A set of player latency policies

```
aws gamelift create-game-session-queue \  
  --name "matchmaker-queue" \  
  --timeout-in-seconds 600 \  
  --destinations DestinationArn=arn:aws:gamelift:us-east-1::alias/alias-a1234567-  
b8c9-0d1e-2fa3-b45c6d7e8910 \  
                DestinationArn=arn:aws:gamelift:us-west-2::alias/alias-b0234567-  
c8d9-0e1f-2ab3-c45d6e7f8901 \  
                DestinationArn=arn:aws:gamelift:us-west-2::fleet/fleet-f1234567-  
b8c9-0d1e-2fa3-b45c6d7e8912 \  
  --player-latency-policies  
  "MaximumIndividualPlayerLatencyMilliseconds=50,PolicyDurationSeconds=120" \  
  
  "MaximumIndividualPlayerLatencyMilliseconds=100,PolicyDurationSeconds=120" \  
  "MaximumIndividualPlayerLatencyMilliseconds=150" \  

```

If the `create-game-session-queue` request is successful, Amazon GameLift Servers returns a [GameSessionQueue](#) object with the new queue configuration.

Set up event notification for game session placement

You can use event notifications to monitor the status of individual placement requests. We recommend setting up event notifications for all games with high-volume placement activity.

There are two options for setting up event notifications.

- Have Amazon GameLift Servers publish event notifications to an Amazon Simple Notification Service (Amazon SNS) topic using a queue.
- Use automatically published Amazon EventBridge events and its suite of tools for managing events.

For a list of game session placement events emitted by Amazon GameLift Servers, see [Game session placement events](#).

Important

For high-volume placement systems, we recommend using standard (non-FIFO) Amazon SNS topics rather than FIFO topics. FIFO topics have lower publishing limits than standard topics, which can lead to throttling exceptions during high load. If you experience throttling with FIFO topics, you may lose queue placement notifications.

Set up an SNS topic

For Amazon GameLift Servers to publish all events generated by a game session queue to a topic, set the notification target field to a topic.

To set up an SNS topic for Amazon GameLift Servers event notification

1. Sign in to the AWS Management Console and open the Amazon SNS console at <https://console.aws.amazon.com/sns/v3/home>.
2. From the SNS **Topics** page, choose **Create topic** and follow the instructions to create your topic.
3. Under **Access policy**, do the following:
 - a. Choose the **Advanced** method.
 - b. Add the following bolded section of the JSON object to the existing policy.

JSON

```
{
  "Version": "2012-10-17",
  "Id": "__default_policy_ID",
  "Statement": [
    {
      "Sid": "__default_statement_ID",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "SNS:GetTopicAttributes",
        "SNS:SetTopicAttributes",
```

```

        "SNS:AddPermission",
        "SNS:RemovePermission",
        "SNS:DeleteTopic",
        "SNS:Subscribe",
        "SNS:ListSubscriptionsByTopic",
        "SNS:Publish"
    ],
    "Resource": "arn:aws:sns:us-east-1:111122223333:your_topic_name",
    "Condition": {
        "StringEquals": {
            "AWS:SourceAccount": "your_account"
        }
    }
},
{
    "Sid": "__console_pub_0",
    "Effect": "Allow",
    "Principal": {
        "Service": "gamelift.amazonaws.com"
    },
    "Action": "sns:Publish",
    "Resource": "arn:aws:sns:us-east-1:111122223333:your_topic_name",
    "Condition": {
        "ArnLike": {
            "aws:SourceArn": "arn:aws:gamelift:us-east-1:111122223333:gamesessionqueue/your_queue_name"
        }
    }
}
]
}

```

- c. (Optional) Add additional access control to the topic by adding conditions to the resource policy.
4. Choose **Create topic**.
5. After you've created your SNS topic, add it to queues during queue creation, or edit an existing queue to add it.

Set up an SNS topic with server-side encryption

With server-side encryption (SSE), you can store sensitive data in encrypted topics. SSE protects the contents of messages in Amazon SNS topics using keys that are managed in AWS Key

Management Service (AWS KMS). For more information about server-side encryption with Amazon SNS, see [Encryption at rest](#) in the *Amazon Simple Notification Service Developer Guide*.

To set up an SNS topic with server-side encryption, review the following topics:

- [Creating key](#) in the *AWS Key Management Service Developer Guide*
- [Enabling SSE for a topic](#) in the *Amazon Simple Notification Service Developer Guide*

When creating your KMS key, use the following KMS key policy:

```
{
  "Effect": "Allow",
  "Principal": {
    "Service": "gamelift.amazonaws.com"
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "*",
  "Condition": {
    "ArnLike": {
      "aws:SourceArn":
        "arn:aws:gamelift:your_region:your_account:gamesessionqueue/your_queue_name"
    },
    "StringEquals": {
      "kms:EncryptionContext:aws:sns:topicArn":
        "arn:aws:sns:your_region:your_account:your_sns_topic_name"
    }
  }
}
```

Set up EventBridge

Amazon GameLift Servers automatically posts all game session placement events to EventBridge. With EventBridge you can set up rules to have events routed to targets for processing. For example, you can set a rule to route the event `PlacementFulfilled` to an AWS Lambda function that handles tasks that precede connecting to a game session. For more information about EventBridge, see [What is Amazon EventBridge?](#) in the *Amazon EventBridge User Guide*.

The following are some examples of EventBridge rules to use with Amazon GameLift Servers queues:

Matches events from all Amazon GameLift Servers queues

```
{
  "source": [
    "aws.gamelift"
  ],
  "detail-type": [
    "GameLift Queue Placement Event"
  ]
}
```

Matches events from a specific queue

```
{
  "source": [
    "aws.gamelift"
  ],
  "detail-type": [
    "GameLift Queue Placement Event"
  ],
  "resources": [
    "arn:aws:gamelift:your_region:your_account:gamesessionqueue/your_queue_name"
  ]
}
```

Reference implementation with AWS CDK

For a complete reference implementation of event-based game session placement using Amazon SNS, AWS Lambda, and Amazon DynamoDB, see the [Event-based game session placement guidance](#) in the Amazon GameLift Toolkit. This guidance includes an AWS CDK template for deploying a complete event-driven game session placement system with an AWS Lambda function for processing game session placement events, an Amazon DynamoDB table for tracking game session placement state, and best practices for Amazon GameLift Servers queue configuration.

Customize to your game hosting solution

With a basic game hosting solution in place, use the following topics to customize and enhance it to improve the player experience, optimize costs, and add advanced functionality. This section

covers various customization options organized by the component they primarily affect. Choose the customizations that best fit your game's requirements and player base.

Topics

- [Game server build customizations](#)
 - [Connect your Amazon GameLift Servers hosted game server to other AWS resources](#)
 - [Let your game server access Amazon GameLift Servers fleet data](#)
 - [Set up VPC peering for Amazon GameLift Servers](#)
- [Player sessions and matchmaking customizations](#)
 - [Generate player IDs](#)
 - [Add FlexMatch matchmaking to Amazon GameLift Servers](#)
- [Game session placement customizations](#)
 - [Customize a game session queue](#)
 - [Prioritize game session placement](#)
 - [Build a queue for Spot Instances](#)
- [Hosting resource customizations](#)
 - [Choose compute resources for a managed fleet](#)
 - [Customize an Amazon GameLift Servers container fleet](#)
 - [Reduce game hosting costs with Spot fleets](#)
 - [Optimize game server runtime configuration on managed Amazon GameLift Servers](#)
 - [Work with the Amazon GameLift Servers Agent](#)
 - [Abstract an Amazon GameLift Servers fleet designation with an alias](#)

Game server build customizations

Game server build customizations offer opportunities to enhance functionality to the game server, including taking advantage of other AWS services. These customizations extend your game server capabilities beyond basic hosting to support advanced features and integrations.

Connect your Amazon GameLift Servers hosted game server to other AWS resources

When you're creating a game server build for deployment on Amazon GameLift Servers fleets, you might want the applications in your game build to communicate directly and securely with other AWS resources that you own. Because Amazon GameLift Servers manages your game hosting fleets, you must give Amazon GameLift Servers limited access to these resources and services.

Some example scenarios include:

- Use an Amazon CloudWatch agent to collect metrics, logs, and traces from managed EC2 fleets and Anywhere fleets.
- Send instance log data to Amazon CloudWatch Logs.
- Obtain game files stored in an Amazon Simple Storage Service (Amazon S3) bucket.
- Read and write game data (such as game modes or inventory) stored in an Amazon DynamoDB database or other data storage service.
- Send signals directly to an instance using Amazon Simple Queue Service (Amazon SQS).
- Access custom resources that are deployed and running on Amazon Elastic Compute Cloud (Amazon EC2).

Amazon GameLift Servers supports these methods for establishing access:

- [Access AWS resources with an IAM role](#)
- [Access AWS resources with VPC peering](#)

Access AWS resources with an IAM role

Use an IAM role to specify who can access your resources and set limits on that access. Trusted parties can "assume" a role and get temporary security credentials that authorize them to interact with the resources. When the parties make API requests related to the resource, they must include the credentials.

To set up access controlled by an IAM role, do the following tasks:

1. [Create the IAM role](#)
2. [Modify applications to acquire credentials](#)
3. [Associate a fleet with the IAM role](#)

Create the IAM role

In this step, you create an IAM role, with a set of permissions to control access to your AWS resources and a trust policy that gives Amazon GameLift Servers rights to use the role's permissions.

For instructions on how to set up the IAM role, see [Set up an IAM service role for Amazon GameLift Servers](#). When creating the permissions policy, choose specific services, resources, and actions that your applications need to work with. As a best practice, limit the scope of the permissions as much as possible.

After you create the role, take note of the role's Amazon Resource Name (ARN). You need the role ARN during fleet creation.

Modify applications to acquire credentials

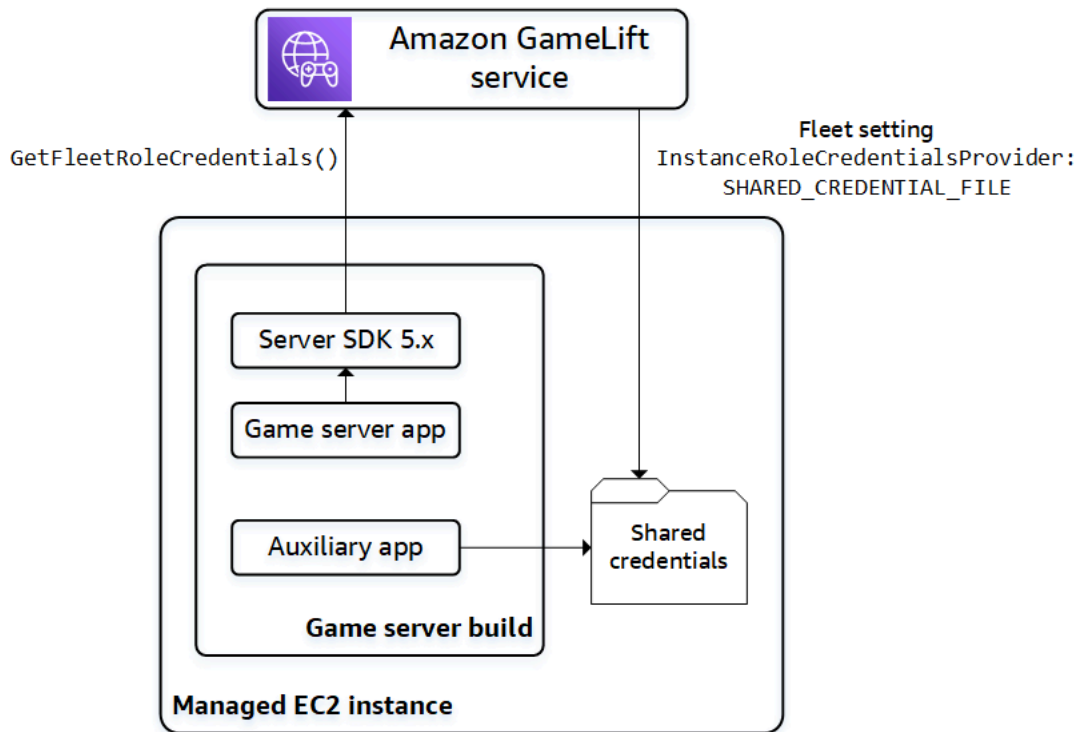
In this step, you configure your applications to acquire security credentials for the IAM role and use them when interacting with your AWS resources. See the following table to determine how to modify your applications based on (1) the type of application, and (2) the server SDK version your game uses to communicate with Amazon GameLift Servers.

	Game server applications	Other applications
Using server SDK version 5.x	Call the server SDK method <code>GetFleetRoleCredentials()</code> from your game server code.	Add code to the application to pull credentials from a shared file on the fleet instance.
Using server SDK version 4 or earlier	Call AWS Security Token Service (AWS STS) AssumeRole with the role ARN.	Call AWS Security Token Service (AWS STS) AssumeRole with the role ARN.

Note

For container fleets, `FleetRoleArn` credentials are injected into each container. Your applications can use the default AWS credentials provider to access these credentials. You can still call `GetFleetRoleCredentials()`, which returns the same credentials. These fleet role credentials are only accessible inside containers.

For games integrated with server SDK 5.x, this diagram illustrates how applications in your deployed game build can acquire credentials for the IAM role.



Call `GetFleetRoleCredentials()` (server SDK 5.x)

In your game server code, which should already be integrated with the Amazon GameLift Servers server SDK 5.x, call `GetFleetRoleCredentials` ([C++](#)) ([C#](#)) ([Unreal](#)) ([Go](#)) to retrieve a set of temporary credentials. When the credentials expire, you can refresh them with another call to `GetFleetRoleCredentials`.

Use shared credentials (server SDK 5.x)

For non-server applications that are deployed with game server builds using server SDK 5.x, add code to get and use credentials stored in a shared file. Amazon GameLift Servers generates a credentials profile for each fleet instance. The credentials are available for use by all applications on the instance. Amazon GameLift Servers continually refreshes the temporary credentials.

You must configure a fleet to generate the shared credentials file on fleet creation.

In each application that needs to use the shared credentials file, specify the file location and profile name, as follows:

Windows:


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

Linux:

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

Example: Set up a CloudWatch agent to collect metrics for Amazon GameLift Servers fleet instances

If you want to use an Amazon CloudWatch agent to collect metrics, logs, and traces from your Amazon GameLift Servers fleets, use this method to authorize the agent to emit the data to your account. In this scenario, take the following steps:

1. Retrieve or write the CloudWatch agent `config.json` file.
2. Update the `common-config.toml` file for the agent to identify the credentials file name and profile name, as described above.
3. Set up your game server build install script to install and start the CloudWatch agent.

Use AssumeRole() (server SDK 4)

Add code to your applications to assume the IAM role and get credentials to interact with your AWS resources. Any application that runs on an Amazon GameLift Servers fleet instance with server SDK 4 or earlier can assume the IAM role.

In the application code, before accessing an AWS resource, the application must call the AWS Security Token Service (AWS STS) [AssumeRole](#) API operation and specify the role ARN. This operation returns a set of temporary credentials that authorizes the application to access to the AWS resource. For more information, see [Using temporary credentials with AWS resources](#) in the *IAM User Guide*.

Associate a fleet with the IAM role

After you've created the IAM role and updated the applications in your game server build to get and use the access credentials, you can deploy a fleet. When you configure the new fleet, set the following parameters.

For container fleets:

- [FleetRoleArn](#) – Set this parameter to the ARN of the IAM role.

For other fleet types:

- [InstanceRoleArn](#) – Set this parameter to the ARN of the IAM role.
- [InstanceRoleCredentialsProvider](#) – To prompt Amazon GameLift Servers to generate a shared credentials file for each fleet instance, set this parameter to `SHARED_CREDENTIAL_FILE`.

You must set these values when you create the fleet. They can't be updated later.

Access AWS resources with VPC peering

You can use Amazon Virtual Private Cloud (Amazon VPC) peering to communicate between applications running on a Amazon GameLift Servers instance and another AWS resource. A VPC is a virtual private network that you define that includes a set of resources managed through your AWS account. Each Amazon GameLift Servers fleet has its own VPC. With VPC peering, you can establish a direct network connection between the VPC for your fleet and for your other AWS resources.

Amazon GameLift Servers streamlines the process of setting up VPC peering connections for your game servers. It handles peering requests, updates route tables, and configures the connections as required. For instructions about how to set up VPC peering for your game servers, see [Set up VPC peering for Amazon GameLift Servers](#).

Note

VPC peering is not supported for container fleets.

Let your game server access Amazon GameLift Servers fleet data

There are some situations where your custom game build or Amazon GameLift Servers Realtime script may require information about the Amazon GameLift Servers fleet. For example, your game build or script might include code to:

- Monitor activity based on fleet data.
- Roll up metrics to track activity by fleet data. (Many games use this data for LiveOps activities.)

- Provide relevant data to custom game services, such as for matchmaking, additional capacity scaling, or testing.

Fleet information is available as a JSON file on each instance in the following locations:

- Windows: C:\GameMetadata\gamelift-metadata.json
- Linux: /local/gamemetadata/gamelift-metadata.json

The `gamelift-metadata.json` file includes the [attributes of an Amazon GameLift Servers fleet resource](#).

Example JSON file:

```
{
  "buildArn": "arn:aws:gamelift:us-west-2:123456789012:build/build-1111aaaa-22bb-33cc-44dd-5555eeee66ff",
  "buildId": "build-1111aaaa-22bb-33cc-44dd-5555eeee66ff",
  "fleetArn": "arn:aws:gamelift:us-west-2:123456789012:fleet/fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
  "fleetDescription": "Test fleet for Really Fun Game v0.8",
  "fleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
  "name": "ReallyFunGameTestFleet08",
  "fleetType": "ON_DEMAND",
  "instanceRoleArn": "arn:aws:iam::123456789012:role/S3AccessForGameLift",
  "instanceType": "c5.large",
  "serverLaunchPath": "/local/game/reallyfungame.exe"
}
```

Set up VPC peering for Amazon GameLift Servers

This topic provides guidance on how to set up a VPC peering connection between your Amazon GameLift Servers-hosted game servers and your other non-Amazon GameLift Servers resources. Use Amazon Virtual Private Cloud (VPC) peering connections to enable your game servers to communicate directly and privately with your other AWS resources, such as a web service or a repository. You can establish VPC peering with any resources that run on AWS and are managed by an AWS account that you have access to.

Note

VPC peering is an advanced feature. To learn about preferred options for enabling your game servers to communicate directly and privately with your other AWS resources, see [Connect your Amazon GameLift Servers hosted game server to other AWS resources](#).

If you're already familiar with Amazon VPCs and VPC peering, understand that setting up peering with Amazon GameLift Servers game servers is somewhat different. You don't have access to the VPC that contains your game servers—it is controlled by the Amazon GameLift Servers service—so you can't directly request VPC peering for it. Instead, you first pre-authorize the VPC with your non-Amazon GameLift Servers resources to accept a peering request from the Amazon GameLift Servers service. Then you trigger Amazon GameLift Servers to request the VPC peering that you just authorized. Amazon GameLift Servers handles the tasks of creating the peering connection, setting up the route tables, and configuring the connection.

IAM permissions for VPC peering

Amazon GameLift Servers uses the caller's credentials to update peer-VPC resources. The IAM user that calls `CreateVpcPeeringAuthorization` or `CreateVpcPeeringConnection` must have the following Amazon EC2 permissions enabled:

- `ec2:AcceptVpcPeeringConnection`
- `ec2:AuthorizeSecurityGroupEgress`
- `ec2:AuthorizeSecurityGroupIngress`
- `ec2:CreateRoute`
- `ec2:DescribeRouteTables`
- `ec2:DescribeSecurityGroups`

To set up VPC peering for an existing fleet**1. Get AWS account ID(s) and credentials.**

You need an ID and sign-in credentials for the following AWS accounts. You can find AWS account IDs by signing into the [AWS Management Console](#) and viewing your account settings. To get credentials, go to the IAM console.

- AWS account that you use to manage your Amazon GameLift Servers game servers.
- AWS account that you use to manage your non-Amazon GameLift Servers resources.

If you're using the same account for Amazon GameLift Servers and non-Amazon GameLift Servers resources, you need ID and credentials for that account only.

2. Get identifiers for each VPC.

Get the following information for the two VPCs to be peered:

- VPC for your Amazon GameLift Servers game servers – This is your Amazon GameLift Servers fleet ID. Your game servers are deployed in Amazon GameLift Servers on a fleet of EC2 instances. A fleet is automatically placed in its own VPC, which is managed by the Amazon GameLift Servers service. You don't have direct access to the VPC, so it is identified by the fleet ID.
- VPC for your non-Amazon GameLift Servers AWS resources – You can establish a VPC peering with any resources that run on AWS and are managed by an AWS account that you have access to. If you haven't already created a VPC for these resources, see [Getting started with Amazon VPC](#). Once you have created a VPC, you can find the VPC ID by signing into the [AWS Management Console](#) for Amazon VPC and viewing your VPCs.

Note

When setting up a peering, both VPCs must exist in the same region. The VPC for your Amazon GameLift Servers fleet game servers is in the same region as the fleet.

3. Authorize a VPC peering.

In this step, you are pre-authorizing a future request from Amazon GameLift Servers to peer the VPC with your game servers with your VPC for non-Amazon GameLift Servers resources. This action updates the security group for your VPC.

To authorize the VPC peering, call the service API [CreateVpcPeeringAuthorization\(\)](#) or use the AWS CLI command `create-vpc-peering-authorization`. Make this call using the account that manages your non-Amazon GameLift Servers resources. Identify the following information:

- Peer VPC ID – This is for the VPC with your non-Amazon GameLift Servers resources.
- Amazon GameLift Servers AWS account ID – This is the account that you use to manage your Amazon GameLift Servers fleet.

Once you've authorized a VPC peering, the authorization remains valid for 24 hours unless revoked. You can manage your VPC peering authorizations using the following operations:

- [DescribeVpcPeeringAuthorizations\(\)](#) (AWS CLI `describe-vpc-peering-authorizations`).
- [DeleteVpcPeeringAuthorization\(\)](#) (AWS CLI `delete-vpc-peering-authorization`).

4. Request a peering connection.

With a valid authorization, you can request that Amazon GameLift Servers establish a peering connection.

To request a VPC peering, call the service API [CreateVpcPeeringConnection\(\)](#) or use the AWS CLI command `create-vpc-peering-connection`. Make this call using the account that manages your Amazon GameLift Servers game servers. Use the following information to identify the two VPCs that you want to peer:

- Peer VPC ID and AWS account ID – This is the VPC for your non-Amazon GameLift Servers resources and the account that you use to manage them. The VPC ID must match the ID on a valid peering authorization.
- Fleet ID – This identifies the VPC for your Amazon GameLift Servers game servers.

5. Track the peering connection status.

Requesting a VPC peering connection is an asynchronous operation. To track the status of a peering request and handle success or failure cases, use one of the following options:

- Continuously poll with `DescribeVpcPeeringConnections()`. This operation retrieves the VPC peering connection record, including the status of the request. If a peering connection is successfully created, the connection record also contains a CIDR block of private IP addresses that is assigned to the VPC.
- Handle fleet events associated with VPC peering connections with [DescribeFleetEvents\(\)](#), including success and failure events.

Once the peering connection is established, you can manage it using the following operations:

- [DescribeVpcPeeringConnections\(\)](#) (AWS CLI `describe-vpc-peering-connections`).
- [DeleteVpcPeeringConnection\(\)](#) (AWS CLI `delete-vpc-peering-connection`).

To set up VPC peering with a new fleet

You can create a new Amazon GameLift Servers fleet and request a VPC peering connection at the same time.

1. Get AWS account ID(s) and credentials.

You need an ID and sign-in credentials for the following two AWS accounts. You can find AWS account IDs by signing into the [AWS Management Console](#) and viewing your account settings. To get credentials, go to the IAM console.

- AWS account that you use to manage your Amazon GameLift Servers game servers.
- AWS account that you use to manage your non-Amazon GameLift Servers resources.

If you're using the same account for Amazon GameLift Servers and non-Amazon GameLift Servers resources, you need ID and credentials for that account only.

2. Get the VPC ID for your non-Amazon GameLift Servers AWS resources.

If you haven't already created a VPC for these resources, do so now (see [Getting started with Amazon VPC](#)). Be sure that you create the new VPC in the same region where you plan to create your new fleet. If your non-Amazon GameLift Servers resources are managed under a different AWS account or user/user group than the one you use with Amazon GameLift Servers, you'll need to use these account credentials when requesting authorization in the next step.

Once you have created a VPC, you can locate the VPC ID in Amazon VPC console by viewing your VPCs.

3. Authorize a VPC peering with non-Amazon GameLift Servers resources.

When Amazon GameLift Servers creates the new fleet and a corresponding VPC, it also sends a request to peer with the VPC for your non-Amazon GameLift Servers resources. You need to pre-authorize that request. This step updates the security group for your VPC.

Using the account credentials that manage your non-Amazon GameLift Servers resources, call the service API [CreateVpcPeeringAuthorization\(\)](#) or use the AWS CLI command `create-vpc-peering-authorization`. Identify the following information:

- Peer VPC ID – ID of the VPC with your non-Amazon GameLift Servers resources.
- Amazon GameLift Servers AWS account ID – ID of the account that you use to manage your Amazon GameLift Servers fleet.

Once you've authorized a VPC peering, the authorization remains valid for 24 hours unless revoked. You can manage your VPC peering authorizations using the following operations:

- [DescribeVpcPeeringAuthorizations\(\)](#) (AWS CLI `describe-vpc-peering-authorizations`).
 - [DeleteVpcPeeringAuthorization\(\)](#) (AWS CLI `delete-vpc-peering-authorization`).
4. Follow the instructions for [creating a new fleet using the AWS CLI](#). Include the following additional parameters:
- *peer-vpc-aws-account-id* – ID for the account that you use to manage the VPC with your non-Amazon GameLift Servers resources.
 - *peer-vpc-id* – ID of the VPC with your non-Amazon GameLift Servers account.

A successful call to [create-fleet](#) with the VPC peering parameters generates both a new fleet and a new VPC peering request. The fleet's status is set to **New** and the fleet activation process is initiated. The peering connection request's status is set to **initiating-request**. You can track the success or failure of the peering request by calling [describe-vpc-peering-connections](#).

When requesting both a new fleet and a VPC peering connection, both actions either succeed or fail. If a fleet fails during the creation process, the VPC peering connection will not be established. Likewise, if a VPC peering connection fails for any reason, the new fleet will fail to move from status **Activating** to **Active**.

Note

The new VPC peering connection is not completed until the fleet is ready to become active. This means that the connection is not available and can't be used during the game server build installation process.

The following example creates both a new fleet and a peering connection between a pre-established VPC and the VPC for the new fleet. The pre-established VPC is uniquely identified by the combination of your non-Amazon GameLift Servers AWS account ID and the VPC ID.

```
$ AWS gamelift create-fleet
  --name "My_Fleet_1"
  --description "The sample test fleet"
  --ec2-instance-type "c5.large"
  --fleet-type "ON_DEMAND"
  --build-id "build-1111aaaa-22bb-33cc-44dd-5555eeee66ff"
  --runtime-configuration "GameSessionActivationTimeoutSeconds=300,
                           MaxConcurrentGameSessionActivations=2,
                           ServerProcesses=[{LaunchPath=C:\game\Bin64.dedicated
\MultiplayerSampleProjectLauncher_Server.exe,
                                           Parameters=+sv_port 33435 +start_lobby,
                                           ConcurrentExecutions=10}]"
  --new-game-session-protection-policy "FullProtection"
  --resource-creation-limit-policy "NewGameSessionsPerCreator=3,
                                   PolicyPeriodInMinutes=15"
  --ec2-inbound-permissions
  "FromPort=33435,ToPort=33435,IpRange=0.0.0.0/0,Protocol=UDP"

  "FromPort=33235,ToPort=33235,IpRange=0.0.0.0/0,Protocol=UDP"
  --metric-groups "EMEAfleets"
  --peer-vpc-aws-account-id "111122223333"
  --peer-vpc-id "vpc-a11a11a"
```

Copyable version:

```
AWS gamelift create-fleet --name "My_Fleet_1" --description "The
sample test fleet" --fleet-type "ON_DEMAND" --metric-groups
"EMEAfleets" --build-id "build-1111aaaa-22bb-33cc-44dd-5555eeee66ff"
--ec2-instance-type "c5.large" --runtime-configuration
"GameSessionActivationTimeoutSeconds=300,MaxConcurrentGameSessionActivations=2,ServerProcesses
\game\Bin64.dedicated\MultiplayerSampleProjectLauncher_Server.exe,Parameters=
+sv_port 33435 +start_lobby,ConcurrentExecutions=10}]" --new-game-session-
protection-policy "FullProtection" --resource-creation-limit-policy
"NewGameSessionsPerCreator=3,PolicyPeriodInMinutes=15" --ec2-inbound-
permissions "FromPort=33435,ToPort=33435,IpRange=0.0.0.0/0,Protocol=UDP"
"FromPort=33235,ToPort=33235,IpRange=0.0.0.0/0,Protocol=UDP" --peer-vpc-aws-account-id
"111122223333" --peer-vpc-id "vpc-a11a11a"
```

Troubleshooting VPC peering issues

If you're having trouble establishing a VPC peering connection for your Amazon GameLift Servers game servers, consider these common root causes:

- An authorization for the requested connection was not found:
 - Check the status of a VPC authorization for the non-Amazon GameLift Servers VPC. It might not exist or it might have expired.
 - Check the regions of the two VPCs you're trying to peer. If they're not in the same region, they can't be peered.
- The CIDR blocks (see [Invalid VPC peering connection configurations](#)) of your two VPCs are overlapping. The IPv4 CIDR blocks that are assigned to peered VPCs cannot overlap. The CIDR block of the VPC for your Amazon GameLift Servers fleet is automatically assigned and can't be changed, so you'll need to change the CIDR block for of the VPC for your non-Amazon GameLift Servers resources. To resolve this issue:
 - Look up this CIDR block for your Amazon GameLift Servers fleet by calling `DescribeVpcPeeringConnections()`.
 - Go to the Amazon VPC console, find the VPC for your non-Amazon GameLift Servers resources, and change the CIDR block so that they don't overlap.
- The new fleet did not activate (when requesting VPC peering with a new fleet). If the new fleet failed to progress to **Active** status, there is no VPC to peer with, so the peering connection cannot succeed.
- A VPC peering connection is not found while the API call throws no exception. Check the fleet events under the fleet being used to create the VPC peering connection for any `FLEET_VPC_PEERING_FAILED` events.
- A VPC peering connection creation fails with fleet event `FLEET_VPC_PEERING_FAILED` and a message indicating the role is not authorized to access a VPC. Verify that the IAM role or user used to create the VPC peering authorization contains the required EC2 permissions listed in [Actions, resources, and condition keys for Amazon GameLift Servers](#).

Player sessions and matchmaking customizations

Player session and matchmaking customizations give you the opportunity to develop sophisticated player management workflows, including a nuanced matchmaking system that helps you deliver balanced, engaging multiplayer experiences.

Generate player IDs

Amazon GameLift Servers uses a player session to represent a player connected to a game session. Amazon GameLift Servers creates a player session each time a player connects to a game session using a game client integrated with Amazon GameLift Servers. When a player leaves a game, the player session ends. Amazon GameLift Servers doesn't reuse player sessions.

Important

When using FlexMatch matchmaking, if you create a new matchmaking request that contains a player ID that is already included in an existing active matchmaking request, the existing request is automatically cancelled. However, a `MatchmakingCancelled` event is not sent for the cancelled request. To monitor the status of existing matchmaking requests, use [DescribeMatchmaking](#) to poll the request status at infrequent intervals (30-60 seconds). The cancelled request will show a status of `CANCELLED` with the reason `Cancelled due to duplicate player`.

The following code example randomly generates unique player IDs:

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

For more information about player sessions, see [Game and player sessions in the Amazon GameLift Servers console](#).

Add FlexMatch matchmaking to Amazon GameLift Servers

Use Amazon GameLift Servers FlexMatch to add player matchmaking functionality to your Amazon GameLift Servers hosted games. You can use FlexMatch with either custom game servers or Amazon GameLift Servers Realtime.

FlexMatch pairs the matchmaking service with a customizable rules engine. You design how to match players together based on player attributes and game modes that make sense for your game. FlexMatch manages the nuts and bolts of evaluating players who are looking for a game, forming matches with one or more teams, and starting game sessions to host the matches.

To use the full FlexMatch service, you must have your hosting resources set up with queues. Amazon GameLift Servers uses queues to locate the best possible hosting locations for games across multiple regions and computing types. In particular, Amazon GameLift Servers queues can use latency data, when provided by game clients, to place game sessions so that players experience the lowest possible latency when playing.

For more information on FlexMatch including detailed help with integrating matchmaking into your games, see these [Amazon GameLift Servers FlexMatch Developer Guide](#) topics:

- [How Amazon GameLift Servers FlexMatch works](#)
- [FlexMatch integration steps](#)

Game session placement customizations

Game session placement customizations give you the opportunity to fine-tune your placement system to give your player base the best possible gameplay experience. You can optimize placement for player compatibility and preferences, as well as for operational considerations such as cost efficiency, geographic distribution, and resilience against service disruptions.

Customize a game session queue

This topic describes how to customize your game session queues to make the best possible decisions about game session placement. For more information about game session queues and how they work, see [Configure game session placement](#).

These Amazon GameLift Servers features require queues:

- [Matchmaking with FlexMatch](#)
- [Build a queue for Spot Instances](#)

Topics

- [Define a queue's scope](#)
- [Build a multi-location queue](#)
- [Evaluate queue metrics](#)

Define a queue's scope

Your game's player population might have groups of players who shouldn't play together. For example, if you publish your game in two languages each language should have its own game servers.

To set up game session placement for your player population, create a separate queue for each player segment. Scope each queue to place players into the correct game servers. Some common ways to scope queues include:

- **By geographic locations.** When deploying your game servers in multiple geographic areas, you might build queues for players in each location to reduce player latency.
- **By build or script variations.** If you have more than one variation of your game server, you might be supporting player groups that can't play in the same game sessions. For example, game server builds or scripts might support different languages or device types.
- **By event types.** You might create a special queue to manage games for participants in tournaments or other special events.

Design multiple queues

Depending on your game and players, you might want to create more than one game session queue. When your game client service requests a new game session, it specifies which game session queue to use. To help you determine whether to use multiple queues, consider:

- **Variations of your game server.** You can create a separate queue for each variation of your game server. All fleets in a queue must deploy compatible game servers. This is because players who use the queue to join games must be able to play on any of the queue's game servers.
- **Different player groups.** You can customize how Amazon GameLift Servers places game sessions based on player group. For example, you might need queues customized for certain game modes that require a special instance type or runtime configuration. Or, you might want a special queue to manage placements for a tournament or other event.
- **Game session queue metrics.** You can set up queues based on how you want to collect game session placement metrics. For more information, see [Amazon GameLift Servers metrics for queues](#).

Build a multi-location queue

We recommend a multi-location design for all queues. This design can improve placement speed and hosting resiliency. A multi-location design is required to use player latency data to put players into game sessions with minimal latency. If you're building multi-location queues that use Spot Instance fleets, follow the instructions in [Reduce game hosting costs with Spot fleets](#).

One way to create a multi-location queue is to add a [multi-location fleet](#) to a queue. That way, the queue can place game sessions in any of the fleet's locations. You can also add other fleets with different configurations or home locations for redundancy. If you're using a multi-location Spot Instance fleet, follow best practices and include an On-Demand Instance fleet with the same locations.

The following example outlines the process of designing a basic multi-location queue. In this example, we use two fleets: one Spot Instance fleet and one On-Demand Instance fleet. Each fleet has the following AWS Regions for placement locations: us-east-1, us-east-2, ca-central-1, and us-west-2.

To create a basic multi-location queue with multi-location fleets

1. Choose a location to create the queue in. You can minimize request latency by placing the queue in a location near where you deployed the client service. In this example, we create the queue in us-east-1.
2. Create a new queue and add your multi-location fleets as queue destinations. The destination order determines how Amazon GameLift Servers places game sessions. In this example, we list the Spot Instance fleet first and the On-Demand Instance fleet second.
3. Define the queue's game session placement priority order. This order determines where the queue searches first for an available game server. In this example, we use the default priority order.
4. Define the location order. If you don't define the location order, Amazon GameLift Servers uses the locations in alphabetical order.

Game session placement locations

Locations where the queue can place new game sessions.

Locations

Choose locations ▼

ca-central-1
Canada (Central) ✕

us-west-2
US West (Oregon) ✕

us-east-2
US East (Ohio) ✕

us-east-1
US East (N. Virginia) ✕

Destination order

An ordered list of fleets and aliases that the queue can use for game session placement.

	Region	Type	Name	
⋮	us-east-1 ▼	Fleet ▼	TestFleet-SPOT ▼	Remove
⋮	us-east-1 ▼	Fleet ▼	TestFleet-ONDEMAND ▼	Remove

Add Destination

Game session placement priority

The values that GameLift uses to prioritize game session placement. The default order is latency, cost, destination, and location.



Latency

Prioritize locations with the lowest average player latency.



Cost

Prioritize destinations with the lowest current hosting cost.



Destination

Prioritize based on the defined destination order.



Location

Prioritize based on the defined location order.

▼ Location order

An ordered list of locations that the queue can use for game session placement.

Location

ca-central-1



Remove

us-east-1



Remove

us-east-2



Remove

us-west-2



Remove

Add locations

Evaluate queue metrics

Use metrics to evaluate how well your queues are performing. You can view metrics related to queues in the [Amazon GameLift Servers console](#) or in Amazon CloudWatch. For a list and descriptions of queue metrics, see [Amazon GameLift Servers metrics for queues](#).

Queue metrics can provide insight about the following:

- **Overall queue performance** – Queue metrics indicate how successfully a queue responds to placement requests. These metrics can also help you identify when and why placements fail. For queues with manually scaled fleets, the `AverageWaitTime` and `QueueDepth` metrics can indicate when you should adjust capacity for a queue.
- **FleetIQ algorithm performance** – For placement requests using the FleetIQ algorithm, metrics show how often the algorithm finds ideal game session placement. The placement may prioritize using resources with the lowest player latency or resources with the lowest cost. There are also error metrics that identify common reasons why Amazon GameLift Servers can't find an ideal placement. For more information about metrics, see [Monitor Amazon GameLift Servers with Amazon CloudWatch](#).
- **Location specific placements** – For multi-location queues, metrics show successful placements by location. For queues that use the FleetIQ algorithm, this data provides useful insight into where player activity occurs.

When evaluating metrics for FleetIQ algorithm performance, consider the following tips:

- To track the queue's rate of finding an ideal placement, use the `PlacementsSucceeded` metric in combination with the FleetIQ metrics for lowest latency and lowest price.
- To boost a queue's rate of finding an ideal placement, review the following error metrics:
 - If the `FirstChoiceOutOfCapacity` is high, adjust capacity scaling for the queue's fleets.
 - If the `FirstChoiceNotViable` error metric is high, look at your Spot Instance fleets. Spot Instance fleets are considered not viable when the interruption rate for a particular instance type is too high. To resolve this issue, change the queue to use Spot Instance fleets with different instance types. We recommend that you include Spot Instance fleets with different instance types in each location.

Prioritize game session placement

Amazon GameLift Servers uses an algorithm to determine how to prioritize a queue's destinations and determine where to place a new game session. The algorithm is based on an ordered set of criteria. You can use the default priority order, or you can customize the order. You can edit a queue's priority order at any time.

Default priority order

1. **Latency** – If the game session placement request includes location-specific latency data for players, Amazon GameLift Servers calculates the average player latency in each location and attempts to place a game session in a fleet location with the lowest average.
2. **Cost** – If a request doesn't include latency data, or if multiple fleets have equal latency, then Amazon GameLift Servers evaluates the hosting cost of each fleet. A fleet's hosting cost varies based on fleet type (Spot or On-Demand), instance type, and location.
3. **Destination** – If multiple fleets have equal latency and costs, then Amazon GameLift Servers prioritizes fleets based on the destination order as listed in the queue configuration.
4. **Location** – For queues with multi-location fleets, if all other criteria are equal, then Amazon GameLift Servers prioritizes the fleet's locations based on alphabetical order.

Customize how a queue prioritizes game session placements

You can choose to customize how a queue prioritizes the placement criteria. The queue applies the custom prioritization to all game session placement requests that it receives.

Note

If you create a custom priority configuration and don't include all four criteria, Amazon GameLift Servers automatically appends any missing criteria in the default order.

To customize a queue's priority configuration

Use the [Amazon GameLift Servers console](#) or the AWS Command Line Interface (AWS CLI) to create a custom priority configuration.

Console

In the [Amazon GameLift Servers console](#), you can customize a queue's priorities when creating a new queue or updating an existing queue. Select an AWS Region to work in.

Open the console's left navigation bar and choose **Queues**. On the Queues page, select an existing queue and choose **Edit**.

1. Go to the section **Game session placement priority**. Drag and drop each priority criteria to create the order you want.

2. Go to the section **Location order**. Add any locations that you want to prioritize. This list is useful when the queue has fleets with multiple locations. At a minimum, you must specify one location. The locations you specify here are prioritized first, followed by all other locations in the queue's destinations.
3. Choose **Save changes**.

AWS CLI

Use the [update-game-session-queue](#) command with the `--priority-configuration` option to customize a queue's priority order. Amazon GameLift Servers updates a queue in your current default AWS Region, or you can add a `--region` tag to specify a different AWS Region.

The following example request adds or updates the priority configuration for a specified queue

```
aws gamelift update-game-session-queue \  
  --name "example-queue-with-priority" \  
  --priority-configuration \  
    PriorityOrder="COST", "LOCATION", "DESTINATION", LocationOrder="us-east-1", "us-  
east-2", "ca-central-1", "us-west-2" \  

```

Prioritize placements by player latency

If you want to give your players the best possible player experience and ensure minimal latency, take the following steps when setting up your game session placement system:

- Set your queue to prioritize latency when choosing where to place game sessions. Latency is at the top of the priority list by default. You can also customize your queue's priority configuration and choose where to put latency in priority order.
- Set up player latency policies for your queue. Latency policies let you set hard limits on the amount of latency to allow in a game session placement. If Amazon GameLift Servers can't place a game session without exceeding the limits, the placement request will time out and fail. You can set up a single latency policy, or you can create a series of policies that gradually relax the latency limit over time. With a series of policies, you can specify very low initial latency limits, and still accommodate players with higher latencies after a short delay. For details on creating latency policies, see [Create a player latency policy](#).
- When making game session placement requests (see [StartGameSessionPlacement](#)), include latency data for each player. Player latency data includes a value for every possible location

where a game session might be placed. For example, for a queue that places game sessions in AWS Regions us-east-2 and ca-central-1, latency data might look like this:

```
"PlayerLatencies": [
  { "LatencyInMilliseconds": 100, "PlayerId": "player1", "RegionIdentifier": "us-east-2" },
  { "LatencyInMilliseconds": 100, "PlayerId": "player1", "RegionIdentifier": "ca-central-1" },
  { "LatencyInMilliseconds": 150, "PlayerId": "player2", "RegionIdentifier": "us-east-2" },
  { "LatencyInMilliseconds": 150, "PlayerId": "player2", "RegionIdentifier": "ca-central-1" }
]
```

To obtain accurate latency measurements, use Amazon GameLift Servers's UDP ping beacons. These endpoints enable you to measure actual UDP network latency between player devices and each of the potential hosting locations, resulting in more accurate placement decisions than using ICMP pings. For more information on using UDP ping beacons to measure latency, refer to [UDP ping beacons](#).

Prioritize placements by location

You can configure a queue to make game session placements based on a prioritized list of geographic locations. Location is one of the criteria that determines how a queue chooses where to place a new game session. By default, location is prioritized fourth, after latency, cost, and destination.

For game session placement, destination and location have somewhat different meanings:

- *Destination* refers to a specific fleet and includes all the fleet's hosting resources, wherever they're deployed. When prioritizing by destination, Amazon GameLift Servers might make a placement with any location in the fleet. Multi-location managed fleets and Anywhere fleets can have hosting resources that are deployed to one or more locations.
- *Location* refers to a specific geographic position where a fleet's hosting resources are deployed. A fleet can have multiple locations, which might include AWS Regions, Local Zones, or custom locations (for an Anywhere fleet). A single-location managed fleet has one location and it is always an AWS Region. A multi-location managed fleet has a home Region and can have remote locations. An Anywhere fleet has one or more custom locations.

When prioritizing placements by location, Amazon GameLift Servers looks for any queue destinations that include the priority location and searches them for an available hosting resource. If there are multiple destinations with the priority location, Amazon GameLift Servers moves on to the next priority criteria (cost, latency, destination).

There are several ways that you can influence how a queue's locations are prioritized

- Configure how the queue handles all game session placement requests:
 - **Add a priority configuration to the queue.** A queue's priority configuration includes an ordered list of locations. You can specify one or more locations to prioritize. This list doesn't exclude any locations, it simply tells Amazon GameLift Servers where to look first for an available hosting resource. A common use for an ordered location list is when you want to funnel most traffic to one or more specific geographic locations and use additional locations as backup capacity. Add a priority configuration by calling [UpdateGameSessionQueue](#).
 - **Add a filter configuration to the queue.** A filter configuration is an allow list for the queue. It tells Amazon GameLift Servers to ignore any locations that aren't on the list when looking for an available hosting resource. There are two common uses for a filter configuration. First, for fleets with multiple locations, you might use a filter to exclude some of the fleet's locations. Second, you might want to temporarily disallow placements on a certain location; for example, a location might be experiencing transitory issues. Because you can update a queue's filter configuration at any time, you can easily add and remove locations as needed. Add a filter configuration by calling [UpdateGameSessionQueue](#).
- Use special instructions for individual placement requests:
 - **Include a priority override list in a game session placement request.** You can provide an alternate priority list of locations with any [StartGameSessionPlacement](#) request. This list effectively replaces the queue's configured prioritization for locations for that one request only. It doesn't impact any other requests. This override feature has a few requirements:
 - Use an override list only with a queue that has a priority configuration in place with `LOCATION` as the first priority.
 - Don't include player latency data in the same placement request. Including latency data sets up conflicts when prioritizing locations that Amazon GameLift Servers can't resolve.
 - Decide how you want Amazon GameLift Servers to proceed if it can't find an available resource on the priority override list. Choose between falling back to the queue's other locations, or limit placements to the override list. By default, Amazon GameLift Servers falls back to attempt placement on the queue's other locations.

- Update the queue's filter configuration as needed, such as adding locations on the override list. The override list doesn't invalidate the filter list.

Create a player latency policy

If your placement requests include player latency data, Amazon GameLift Servers finds game sessions in locations with the lowest average latency for all players. Placing game sessions based on average player latency prevents Amazon GameLift Servers from placing most players in games with high latency. However, Amazon GameLift Servers still places players with extreme latency. To accommodate these players, create player latency policies.

A player latency policy prevents Amazon GameLift Servers from placing a requested game session anywhere that players in the request would experience latency over the maximum value. Player latency policies can also prevent Amazon GameLift Servers from matching game session requests with higher latency players.

Tip

To manage latency specific rules, such as requiring similar latency across all players in a group, you can use [Amazon GameLift Servers FlexMatch](#) to create latency-based matchmaking rules.

For example, consider this queue with a 5-minute timeout and the following player latency policies:

1. Spend 120 seconds searching for a location where all player latencies are less than 50 milliseconds.
2. Spend 120 seconds searching for a location where all player latencies are less than 100 milliseconds.
3. Spend the remaining queue time until timeout searching for a location where all player latencies are less than 200 milliseconds.

Create queue

Queue settings

Name

The name must be unique and have 1-128 characters. Valid characters: A-Z, a-z, 0-9, and - (hyphen).

Timeout

Specify how long GameLift tries to place a game session before stopping.

 seconds

Must be 10-600 seconds.

 We recommend setting player latency policies, unless you're using GameLift FlexMatch.



Player latency policies - *optional*

Add policies to help place players into games with lower latency. Use multiple policies to reduce latency requirements per policy so that each player eventually finds a match.

0 seconds left to allocate 100%

Period start

Seconds

Period end

Seconds

Max player latency

Milliseconds

Remove

Seconds

Seconds

Milliseconds

Remove

Seconds

Seconds

Milliseconds

Remove

Add policy

Build a queue for Spot Instances

You can achieve of potentially significant savings in hosting costs by using Spot fleets. For more details about Spot fleets and how to use them, see [On-Demand Instances versus Spot Instances](#).

If your game hosting solution includes Spot fleets, you must use a game session placement queue. Amazon GameLift Servers uses queues to search across multiple game hosting resources and select the best one available to host a new game session. With Spot fleets, queues are particularly important for minimizing hosting costs and avoiding possible Spot interruptions. This topic helps you set up a resilient queue that can continue to host games for players even in the event of interruptions, slowdowns and outages. You can customize how the queue prioritizes available hosting resources based on several factors including hosting cost.

Are you using FlexMatch for matchmaking? You can use a queue with Spot fleets to make game session placements for your matches.

Implementation tasks for Spot fleets

When creating or updating your game hosting solution to use Spot fleets, complete the following tasks. For more detailed guidance on how to build a queue that optimizes Spot availability and resiliency, see [Reduce game hosting costs with Spot fleets](#).

1. Choose and create a set of fleet destinations for your game session queue.

Start by deciding where you want your queue to place game sessions. A queue can search across multiple fleets to find the best possible placement. Each fleet has one instance type but can have multiple geographic locations. Queues with fleets that offer variety in both location and instance type are more likely to make successful placements. See these best practices for designing an effective and resilient Spot-optimized queue.

2. Create your Spot-optimized game session queue.

Create a queue and configure it for your Spot fleets. See [Create a game session queue](#) for help creating and configuring the new queue. You can use the Amazon GameLift Servers console or the AWS CLI to create or edit a queue.

- Add the fleet destinations from Step 1.
- Prioritize the destination order as appropriate. By default, Amazon GameLift Servers prioritizes by cost before destination, so destination order is used only when the lowest costs between destinations are equal.
- If you want to prioritize game hosting cost before player latency, provide a custom placement priority. See [Prioritize game session placement](#).

3. Update other components in your solution to use the new queue.

When your solution uses a Spot-optimized queue to start new game sessions, the queue automatically avoids placing game sessions with fleets that have a high likelihood of interruption. It instead searches all viable fleets for resources that match your defined priorities, including player latency, hosting cost, and destination order.

- If you're not using FlexMatch – Update your backend service to specify the new Spot-optimized queue in game session requests. The backend service makes API requests to Amazon GameLift Servers on behalf of your game client (using `StartGameSessionPlacement()`), and each request must specify a queue name. For help implementing game session placements in your game client, see [Create game sessions](#).
- If you are using FlexMatch – Update your matchmaking configuration to send game session requests to the new Spot-optimized queue. When the matchmaking system forms a player match, it sends a game session placement request to the designated queue to start a new game session for the match. Only matchmaking configurations with FlexMatch mode set to "Managed" can designate a placement queue. You can update a matchmaking configuration using the AWS CLI or the Amazon GameLift Servers console (see [Edit a matchmaking configuration](#)).

4. Review the performance of your Spot fleets and queues.

View Amazon GameLift Servers metrics in the Amazon GameLift Servers console or with Amazon CloudWatch to review performance. For more information about Amazon GameLift Servers metrics, see [Monitor Amazon GameLift Servers with Amazon CloudWatch](#). Key metrics include:

- Interruption rate – Use the `InstanceInterruptions` and `GameSessionInterruptions` metrics to track the number and frequency of Spot-related interruptions for instances and game sessions. Game sessions on reclaimed instances have a status of `TERMINATED` and a status reason of `INTERRUPTED`.
- Queue effectiveness – Track placement success rates, average wait time, and queue depth to confirm that Spot fleets don't impact your queue performance.
- Fleet usage – Monitor data on instances, game sessions and player sessions. Usage for your On-Demand fleets can be an indicator that queues are avoiding placements into your Spot fleets to avoid disruption.

Best practices for queues with Spot fleets

Use the following best practices when creating fleets and queues for Spot instances.

- **Expand your queue's geographic coverage.** Even if your players are clustered in a single AWS Region, add adjacent locations to your Spot fleet. This approach improves the queue's ability to maintain capacity during regional slowdowns, outages, and Spot interruptions. Multi-location fleets work with both Spot and On-Demand instances.
- **Diversify your queue's instance type coverage.** Amazon GameLift Servers evaluates Spot viability based on instance type, so having Spot fleets with a variety of instance types reduces the chance that multiple Spot fleets are nonviable at the same time. Include at least two Spot fleets with different instances types each location.

Note

Pricing is based on the instances you use, not the number of fleets. Running five fleets with 10 instances each is the same as running one fleet with 50 instances of similar cost. Pricing varies by instance type, size, and location.

Tips for grouping Spot instance types:

- Use instance types in the same family, such as `m6g.medium`, `m6g.large`, and `m6g.xlarge`. Larger instance types cost more, but can also host more game sessions at a time.
- Select widely available instances types. Typically, older generation families (such as C5, M5, and R5) and common sizes (such as `.large`, `.xlarge`, and `.2xlarge`) have better availability.
- Check the 30-90 day pricing history in the Amazon GameLift Servers console. Look for instance types with consistent availability patterns.
- Use the Amazon GameLift Servers console, fleet creation tool, to explore location coverage for instance types.
- **Add On-Demand fleets for backup capacity.** Game hosting can switch to On-Demand fleets whenever Spot fleets are unavailable. Put at least one On-Demand fleet in each location to maintain low player latency. Add auto-scaling to your backup On-Demand fleets, so you can keep them scaled down until they're needed.
- **Assign aliases to all fleet destinations.** Create aliases for each of your queue's destinations. Aliases make it easier and more efficient whenever you need to replace fleets.

- **Apply a queue prioritization strategy.** You can customize how a queue prioritizes where to place game sessions (see [Prioritize game session placement](#) for more details). For Spot-optimized queues, prioritizing by cost ensures that low-cost Spot fleets are used whenever possible.

You can also prioritize certain fleets by specifying a destination order. For example, some users designate a set of primary fleets for regular use and also a set of secondary fleets as backup. In this scenario, set the queue's destination order to list the primary fleets first. Then configure the queue's priority order with destination followed by cost.

Hosting resource customizations

This section provides advanced options for configuring and managing your Amazon GameLift Servers infrastructure to meet specific performance, cost, and operational requirements. In particular, the topics in this section describe how you can customize your Amazon GameLift Servers managed hosting resources to best fit your game and your players.

Some of the decisions you want to consider:

- Where to deploy hosting resources for your players? Gameplay latency is a major factor in selecting your fleet's geographic locations, but there are other factors that vary by location, including resource type availability and cost.
- What EC2 instance types will best support your game? Choose from available instance types that use the best combination of compute architecture, memory, storage, and networking capacity.
- What size of instance type do you need? Choose an instance type size based on the resource requirements (memory and CPU) of your game server software and other factors.
- Should your fleet use On-Demand or Spot Instances? Consider whether you can take advantage of lower Spot pricing, and whether Amazon GameLift Servers sufficiently mitigates the chance of Spot interruptions to your game sessions.
- How do you want your game server software to run on each fleet instance? The runtime configuration tells Amazon GameLift Servers what server software to run and how.
- For container fleets, does the default configuration work for your game? Amazon GameLift Servers does a lot of the work for you to optimize your container fleet configurations, but you can customize most configuration settings.

Choose compute resources for a managed fleet

For Amazon GameLift Servers managed hosting, including managed EC2 and managed containers, the service deploys your game servers to fleets of computing resources in the AWS Cloud. When you create a managed fleet, you want to configure the hosting resources to best suit your game. This topic discusses key decision points when choosing and configuring your game hosting fleets.

Note

If you're building a hybrid solution with both Anywhere and Amazon GameLift Servers managed fleets, use these topics to design managed fleets to supplement your own self-managed resources. See [Deploy hosting fleets for Amazon GameLift Servers](#).

Topics

- [Geographic locations](#)
- [Operating systems](#)
- [Instance types](#)
- [On-Demand Instances versus Spot Instances](#)
- [Service quotas](#)

Geographic locations

Consider where you plan to deploy your game servers. In general, you want to put your game servers as close as possible to your players to deliver the best possible player experience. For Amazon GameLift Servers managed hosting, you can choose to put game servers in any of the supported AWS Regions and Local Zones. If you're building a hybrid solution, consider how managed fleet deployment can supplement the locations of your self-managed Amazon GameLift Servers Anywhere fleets.

For most development and testing scenarios, deploying to a single location makes sense. As you prepare for launch and beyond, there are many reasons to deploy across multiple geographic locations. These include supporting a widespread group of players and improving overall game hosting resilience and reliability. Multiple locations can also boost player experience by speeding up game session placement and enabling more choices when optimizing placements for latency and cost.

For a list of locations that are supported by Amazon GameLift Servers and more information about locations for all fleet types, see [Amazon GameLift Servers service locations](#).

Multi-location fleets

A single managed fleet can deploy resources to multiple locations. You can manually set capacity for each individual location in a multi-location fleet.

Advantages of using a multi-location fleet:

- Simplified fleet deployment and management – You supply the game server software and fleet configuration, and Amazon GameLift Servers deploys it to fleet instances across multiple locations (build once, deploy anywhere). In a production fleet, you can view and manage all locations in a fleet instead of having to manage multiple fleets each located in a different region.
- Local Zone availability – If you want to use a Local Zone you must create a multi-location fleet with an AWS Region home location and Local Zones as remote locations. Local Zones are extensions of AWS Regions that can deliver even lower latency to areas and customers that need it. You can add a Local Zone to any multi-location fleet; you don't need to include the Local Zone's parent AWS Region.
- Compatibility with game session queues – You can build game session placement queues with one or more multi-location fleets. This approach gives the queue flexibility when prioritizing and choosing locations to host a new game session.
- Efficient resource utilization – With auto-scaling turned on, Amazon GameLift Servers can better optimize capacity scaling across all locations in a fleet.

Tips for using multi-location fleets:

- Check for quotas on the number of locations per AWS Region or fleet. See [Amazon GameLift Servers service quotas](#).
- Not all instance types are available in all locations. Depending on your chosen locations, you might have limited instance type options. The Amazon GameLift Servers console provides useful tools to help you find the right balance of locations and instance types.
- Consider using [UDP ping beacons](#) to collect player latency data for all your fleet locations. Amazon GameLift Servers can use this data to optimize game sessions for low latency and prevent players from joining sessions with unacceptably high latency. These special endpoints accept UDP messages instead of traditional ICMP pings, providing accurate latency measurements to help you select optimal fleet locations.

Operating systems

All instances in a managed fleet are deployed with an Amazon machine image (AMI) that provides a complete runtime environment for your game server software. For managed EC2 fleets, you specify the game server build's operating system when you upload the build to Amazon GameLift Servers. For managed container fleets, you specify the operating system in the container group definition. For more information about the latest AMI versions, see [Amazon GameLift Servers AMI versions](#).

AMI versions are regularly updated. When you create a new fleet, Amazon GameLift Servers assigns the latest available version of the AMI you selected for your game build. All instances that are deployed in that fleet use the same version. To keep your AMI version up to date with the latest security and software updates, you need to regularly replace your fleets. As a best practice, we recommend replacing your managed fleets every 30 days to maintain the runtime environment for your game servers. For guidance, see [Security best practices for Amazon GameLift Servers](#).

Instance types

A managed fleet's instance type determines the kind of hardware that is deployed for all fleet instances, and instance types are generally available in various sizes. All Amazon GameLift Servers managed fleet use Amazon EC2 instances, and support a wide range of instance types that offer different combinations of computing power, memory, storage, and networking capabilities. The availability of instance types varies depending on the locations you choose.

The Amazon GameLift Servers console provides useful tools to help you find the right instance types for your game build and your deployment locations. For managed container fleets, the console also offers guidance on your game's CPU power and memory requirements.

When choosing from available instance types for your game, consider:

- The compute architecture of your game server: x64 or Arm (AWS Graviton).

Note

Graviton Arm instances require a server build for a Linux AMI. Server SDK 5.1.1 or newer is required for C++ and C#. Server SDK 5.0 or newer is required for Go. These instances do not provide out-of-the-box support for Mono installation on Amazon Linux 2023 (AL2023) or Amazon Linux 2 (AL2).

- The computing, memory, and storage requirements of your game server build.

- The size of your instance type. In addition to meeting the requirements of your game server software executables, larger instance type sizes can run multiple game server processes and/or containers on each instance. Factors to consider include cost (is it cheaper to run a few large instances or many small instances). Also consider how game session capacity might be impacted by adding or removing instances during fleet scaling events or when shutting down unhealthy instances. If each instance runs many game server processes concurrently, adding or removing an instance can significantly affect game hosting capacity.

For more information about instance types, see [Amazon EC2 Instance Types](#).

On-Demand Instances versus Spot Instances

Amazon EC2 On-Demand Instances and Spot Instances offer the same hardware and performance, but they differ in availability and cost.

On-Demand Instances

You can acquire an On-Demand Instance when you need it and keep it for as long as you want. On-Demand Instances have a fixed cost, meaning you pay only for the amount of time that you use them. There are no long-term commitments.

Spot Instances

Spot Instances can offer a cost-efficient alternative to On-Demand Instances by utilizing unused AWS computing capacity. Spot Instance prices fluctuate based on the supply and demand for each instance type in each location. AWS can reclaim Spot Instances with a two-minute notification whenever it needs the capacity back, and game sessions actively running on a reclaimed instance are interrupted.

Amazon GameLift Servers offers several tools to help mitigate the likelihood of Spot interruptions to your game sessions. A spot viability algorithm tracks instance type historical data to anticipate when the risk of interruption reaches a critical point and avoids placing new game sessions on Spot Instances of that type. If an interruption does occur, your game servers can use the notification to gracefully end a game session for players.

Game hosting with Spot fleets must use a queue for game session placement. A queue is able to prioritize game session placements based on Spot fleet viability, cost, and other factors. See these topics for more information about how to take advantage of Spot for your game server hosting:

- [Reduce game hosting costs with Spot fleets](#)

- [Build a queue for Spot Instances](#)

Service quotas

You can view default service quotas for Amazon GameLift Servers and current quota status for your AWS account using the following tools:

- For general service quota information for Amazon GameLift Servers, see [Amazon GameLift Servers endpoints and quotas](#) in the *AWS General Reference*.
- For a list of available instance types per location for your account, open the [Service quotas](#) page of the Amazon GameLift Servers console. This page also displays your account's current usage for each instance type in each location.
- For a list of your account's current quotas for instance types per Region, run the AWS Command Line Interface (AWS CLI) command [describe-ec2-instance-limits](#). This command returns the number of active instances that you have in your default Region (or in another Region that you specify).

As you prepare to launch your game, fill out a launch questionnaire in the [Amazon GameLift Servers console](#). The Amazon GameLift Servers team uses the launch questionnaire to determine the correct quotas and limits for your game.

Customize an Amazon GameLift Servers container fleet

The topics in this section describe some of the optional features for Amazon GameLift Servers managed containers. You can choose to use any or all of these features.

Topics

- [Set resource limits](#)
- [Understand container fleet memory allocation](#)
- [Configuring NVMe Drive Access](#)
- [Designate essential containers](#)
- [Configure network connections](#)
- [Set up health checks for containers](#)
- [Set container dependencies](#)
- [Configure a container fleet](#)

Set resource limits

For each container group, you can determine how much memory and computing power the container group needs to run its software. Amazon GameLift Servers relies on this information to manage resources across the container group. It also uses this information to calculate how many game server container groups a fleet image can hold. You can also set limits for individual containers.

You can set a maximum limit on memory and computing power for a container group. By default, these resources are shared by all containers in the group. You can further customize resource management by setting limits for individual containers.

Set optional limits for individual containers

Setting container-specific resource limits allows you to exert greater control over how individual containers can use the group's resources. If you don't set container-specific limits, all containers in the group share the group resources. Sharing offers greater flexibility to use resources where they're needed. It also increases the potential for processes to compete with each other and result in container failure.

Set any of the following `ContainerDefinition` properties for any container.

- `MemoryHardLimitMebibytes` – Set a maximum memory limit for the container. If the container exceeds this limit, it results in a restart.
- `Vcpu limit` – Reserve a minimum amount of vCPU resources for the container's exclusive use. The container always has the reserved amount available to it. It can exceed this minimum at any time, if additional resources are available. (1024 CPU units is the equivalent of 1 vCPU.)

Set total resource limits for a container group

If you set limits for individual containers, you might need to modify how much memory and vCPU resources the container group needs. The goal is to allocate enough resources to optimize game server performance. Amazon GameLift Servers uses these limits to calculate how to pack game server container groups on a fleet instance. You'll also use them when choosing an instance type for a container fleet.

Calculate the total memory and vCPU needed for a container group. Consider the following:

- What are all the processes that run across all containers in the container group? Add up the resources required for these processes. Take note of any container-specific limits.
- How many concurrent game server processes do you plan to run in each container group? You determine this in your game server container image.

Based on your estimate of container group requirements, set the following `ContainerGroupDefinition` properties:

- `TotalMemoryLimitMebibytes` – Set a maximum memory limit for the container group. All containers in the group share allocated memory. If you set individual container limits, the total memory limit must be equal to or greater than the highest container-specific memory limit.
- `TotalVcpuLimit` – Set a maximum vCPU limit for the container group. All containers in the group share allocated CPU resources. If you set individual container limits, the total CPU limit must be equal to or greater than the sum of all container-specific CPU limits. As a best practice, consider setting this value to double the sum of the container CPU limits.

Example scenario

Let's say we're defining a game server container group with the following three containers:

- Container A is our game server container. We estimate the resource requirements for one game server at 512 MiB and 1024 CPU. We plan to have the container run 1 server process. Because this container runs our most critical software, we set no memory limit or vCPU reserve limit.
- Container B is a support container with resource requirements estimated at 1024 MiB and 1536 CPU. We set a memory limit of 2048 MiB, and a CPU reserve limit of 1024 CPU.
- Container C is another support container. We set a hard memory limit of 512 MiB and a CPU reserve limit of 512 CPU.

Using this information, we set the following total limits for the container group:

- Total memory limit: 7680 MiB. This value exceeds the highest memory limit (1024 MiB).
- Total CPU limit: 13312 CPU. This value exceeds the sum of the CPU limit (1024+512 CPU).

Understand container fleet memory allocation

When Amazon GameLift Servers deploys container groups on a fleet instance, not all of the instance's memory is available for your containers. Amazon GameLift Servers reserves a portion of the instance memory for the operating system, the Amazon ECS agent, and other supporting services. The amount of reserved memory varies based on the instance type's total memory. Understanding this overhead helps you configure your container group definitions to fully utilize the available resources.

Memory overhead formula

Amazon GameLift Servers calculates the memory available for your container groups using the following steps:

1. **Determine the memory buffer percentage.** Amazon GameLift Servers reserves a percentage of the instance's total memory based on the following tiers:

Instance memory (MiB)	Reserved percentage
Less than 5,000	8%
5,000 to 9,999	6%
10,000 to 89,999	5%
90,000 to 199,999	4%
200,000 or more	3%

2. **Calculate available memory.** Subtract the reserved memory from the total instance memory:

$$\text{AvailableMemory} = \text{InstanceMemory} - \text{round}(\text{InstanceMemory} \times \text{BufferPercentage})$$

3. **Subtract per-instance container group memory.** If your fleet uses a per-instance container group, subtract its `TotalMemoryLimitMebibytes` from the available memory. One per-instance container group runs on each fleet instance.

$$\text{AvailableMemory} = \text{AvailableMemory} - \text{PerInstanceCGD.TotalMemoryLimitMebibytes}$$

4. **Account for log router overhead.** If logging is enabled for the fleet, Amazon GameLift Servers reserves an additional 50 MiB per game server container group for the log router.

5. **Calculate maximum game server container groups.** The maximum number of game server container groups that fit on the instance by memory is:

$$\text{MaxGroupsByMemory} = \text{floor}(\text{AvailableMemory} / (\text{GameServerCGD.TotalMemoryLimitMebibytes} + \text{LogRouterMemory}))$$

Where `LogRouterMemory` is 50 MiB if logging is enabled, or 0 if logging is disabled.

Note

Memory is only one factor that determines how many game server container groups fit on an instance. Amazon GameLift Servers also considers vCPU capacity and available connection ports, and uses the minimum of all three calculations.

Example memory calculation

Consider a fleet using a `c5.xlarge` instance (8,192 MiB total memory) with logging enabled:

1. Instance memory is 8,192 MiB, which falls in the 5,000–9,999 tier (6% buffer)
2. Reserved memory = $\text{round}(8,192 \times 0.06) = 492$ MiB
3. Available memory = $8,192 - 492 = 7,700$ MiB
4. If using a per-instance container group with `TotalMemoryLimitMebibytes` of 512: Available memory = $7,700 - 512 = 7,188$ MiB
5. If each game server container group has `TotalMemoryLimitMebibytes` of 1,024:
 $\text{MaxGroupsByMemory} = \text{floor}(7,188 / (1,024 + 50)) = \text{floor}(7,188 / 1,074) = 6$

Available memory by instance type

The following table shows the total memory and available memory (after the Amazon GameLift Servers buffer) for commonly used instance types. Use these values as a starting point when configuring your container group definitions. The *Available memory* column shows the memory available for all container groups on the instance, before subtracting any per-instance container group or log router overhead.

Instance type	Total memory (MiB)	Buffer percentage	Available memory (MiB)
c5.large	4,096	8%	3,768
c5.xlarge	8,192	6%	7,700
c5.2xlarge	16,384	5%	15,565
c5.4xlarge	32,768	5%	31,130

Instance type	Total memory (MiB)	Buffer percentage	Available memory (MiB)
c5.9xlarge	73,728	5%	70,042
c5.12xlarge	98,304	4%	94,372
c5.18xlarge	147,456	4%	141,558
c5.24xlarge	196,608	4%	188,744
m5.large	8,192	6%	7,700
m5.xlarge	16,384	5%	15,565
m5.2xlarge	32,768	5%	31,130
m5.4xlarge	65,536	5%	62,259
m5.8xlarge	131,072	4%	125,829
m5.12xlarge	196,608	4%	188,744
r5.large	16,384	5%	15,565
r5.xlarge	32,768	5%	31,130
r5.2xlarge	65,536	5%	62,259
r5.4xlarge	131,072	4%	125,829
c6i.large	4,096	8%	3,768
c6i.xlarge	8,192	6%	7,700
c6i.2xlarge	16,384	5%	15,565
c6i.4xlarge	32,768	5%	31,130
c6i.8xlarge	65,536	5%	62,259
c7i.large	4,096	8%	3,768

Instance type	Total memory (MiB)	Buffer percentage	Available memory (MiB)
c7i.xlarge	8,192	6%	7,700
c7i.2xlarge	16,384	5%	15,565
c7i.4xlarge	32,768	5%	31,130
c7i.8xlarge	65,536	5%	62,259
m7i.large	8,192	6%	7,700
m7i.xlarge	16,384	5%	15,565
m7i.2xlarge	32,768	5%	31,130
m7i.4xlarge	65,536	5%	62,259
m7i.8xlarge	131,072	4%	125,829
m7i.12xlarge	196,608	4%	188,744
r7i.large	16,384	5%	15,565
r7i.xlarge	32,768	5%	31,130
r7i.2xlarge	65,536	5%	62,259
r7i.4xlarge	131,072	4%	125,829

For instance types not listed here, you can calculate the available memory using the formula described above. Check the [Amazon EC2 instance types documentation](#) for the total memory of your chosen instance type.

Configuring NVMe Drive Access

On d-type instances, the NVMe drive automatically mounts to the /data directory during host startup. To enable containers to access the SSD storage, set the following ContainerGroupDefinition property MountPoints:

- `InstancePath` – Set to `/data` to reference the auto-mounted NVMe drive on the host instance.
- `AccessLevel` – Choose the appropriate access level for your container's needs (e.g., `READ_ONLY` or `READ_WRITE`).
- `ContainerPath` – (Optional) Specify the path where the instance path will be mounted inside the container. If not specified, it defaults to the instance path.

For more information about mount points, see [ContainerMountPoint](#) in the Amazon GameLift Servers API Reference.

Designate essential containers

For a per-instance container group, designate each container as essential or non-essential. Per-instance container groups must have at least one essential support container. The essential container does the critical work of the container group. The essential container is always expected to be running. If it fails, the entire container group restarts.

Set the `ContainerDefinition` property `Essential` to either `true` or `false` for each container.

Configure network connections

You can customize network access to allow external traffic connect to any container in a container fleet. For example, you must establish network connections to the container that runs your game server processes, so that game clients can join and play your game. Game clients connect to game servers using ports and IP addresses.

In a container fleet, the connection between a client and server is not direct. Internally, a process in a container listens on a *container port*. Externally, incoming traffic connects to a fleet instance using a *connection port*. Amazon GameLift Servers maintains the mappings between internal container ports and external-facing connection ports, so that incoming traffic gets routed to the correct process on the instance. To retrieve the current port mappings for a specific container group, call the [DescribeContainerGroupPortMappings](#) operation. For more information about viewing port mappings, see [View container port mappings](#).

Amazon GameLift Servers provides an extra layer of control for your network connections. Each container fleet has an *inbound permissions* setting, which allows you to control access to each external-facing connection port. For example, you could remove permissions for all connection ports to shut off all access to the fleet's containers.

You can update a fleet's inbound permissions, connection ports, and container ports.

⚠ Warning

If you provide a custom `InstanceConnectionPortRange` or `InstanceInboundPermissions`, Amazon GameLift Servers will no longer manage either value for your fleet. You must set both fields to avoid undefined behavior.

Set container port ranges

Configure container port ranges as part of each container definition. This is a required parameter for a container group definition. You need to configure enough ports to accommodate all concurrently running processes that need external access. Some containers will not need any ports.

Your game server container, which runs your game servers, needs a port for every concurrently running game server process. The game server process listens on the assigned port and reports it to Amazon GameLift Servers.

Set connection port ranges

Configure your container fleet with a set of connection ports. Connection ports provide external access to the fleet instances that are running your containers. Amazon GameLift Servers assigns connection ports and maps them to container ports as needed.

By default, Amazon GameLift Servers calculates the number of ports required for all container groups and sets a port range to accommodate them. We highly recommend you use Amazon GameLift Servers calculated values, which are updated when you deploy updates to a container group definition. If you do need to customize connection port ranges, use the following guidance.

When you create a container fleet, define a connection port range (see [ContainerFleet:InstanceConnectionPortRange](#)). Make sure that the range has enough ports to map to every container port that's defined across all containers in both container groups in the fleet. To calculate the minimum connection ports needed, use the following formula:

```
[Total number of container ports defined for containers in the game  
server container group] * [Number of game server container groups per  
instance] + [Total number of container ports defined for containers in  
the per-instance container group]
```

As a best practice, double the minimum number of connection ports.

Note

The number of connection ports can potentially limit the number of game server container groups per instance. If a fleet has only enough connection ports for one game server container group per instance, Amazon GameLift Servers will deploy only one game server container group, even if the instances have enough compute power for multiple game server container groups.

Set inbound permissions

Inbound permissions control external access to a container fleet by specifying which connection ports to open for incoming traffic. You can use this setting to turn a fleet's network access on and off as needed.

By default, Amazon GameLift Servers calculates the number of ports required for all container groups and sets a port range to accommodate them. We highly recommend you use Amazon GameLift Servers calculated values, which are updated when you deploy updates to a container group definition. If you do need to customize connection port ranges, use the following guidance.

When you create a container fleet, define a set of inbound permissions (see [ContainerFleet:InstanceInboundPermissions](#)). Inbound permission ports should match with the fleet's connection port ranges.

Note

Since container ports are selected randomly from the `InstanceConnectionPortRange`, in order to guarantee that session connections can be made, all ports in `InstanceConnectionPortRange` should be covered by ports in `InstanceInboundPermissions`.

Example scenario

This example illustrates how to set all three network connection properties.

- Our fleet's game server container group has 1 container, which runs 1 game server process.

In the game server container group definition, we set the `PortConfiguration` parameter for this container as follows:

```
"PortConfiguration": {  
  "ContainerPortRanges": [ { "FromPort": 10, "ToPort": 20, "Protocol": "TCP"} ]  
}
```

- Our fleet also has a per-instance container group with 1 container. It has 1 process that needs network access. In the per-instance container definition, we set the `PortConfiguration` parameter for this container as follows:

```
"PortConfiguration": {  
  "ContainerPortRanges": [ { "FromPort": 25, "ToPort": 25, "Protocol": "TCP"} ] }
```

- Our fleet is configured with 20 game server container groups per fleet instance. Given this information, we can use the formula to calculate the number of connection ports we need:
 - Minimum: **21 ports** [1 game server container ports * 20 game server container groups per instance + 1 per-instance container port]
 - Best practice: **42 ports** [minimum ports * 2]

When creating the container fleet, we set the `InstanceConnectionPortRange` parameter as follows:

```
"InstanceConnectionPortRange": { "FromPort": 1010, "ToPort": 1071 }
```

- We want to allow access to all available connection ports. When creating the container fleet, we set the `InstanceInboundPermissions` parameter as follows:

```
"InstanceInboundPermissions": [  
  { "FromPort": 1010, "ToPort": 1071, "IpRange": "10.24.34.0/23", "Protocol":  
    "TCP"} ]
```

Set up health checks for containers

A container automatically restarts if it experiences a terminal failure and stops running. If a container is considered essential, it prompts the entire container group to restart.

All game server containers are automatically considered essential. Support containers can be designated essential, but they need to have a mechanism to report health. You can set health checks for non-essential support containers as well.

You can define additional custom criteria to measure container health and use a health check to test that criteria. To set up a container health check, you can define it in a Docker container image or in your container definition. If you set a health check in the container definition, it overrides any settings in the container image.

Set the following `SupportContainerDefinition` properties for a container health check:

- **Command** — Provide a command that checks some aspect of the container's health. You decide what criteria to use to measure health. The command must result in an exit value of 1 (unhealthy) or 0 (healthy).
- **StartPeriod** — Specify an initial delay before health check failures start counting. This delay gives the container time to bootstrap its processes.
- **Interval** — Decide how often to run the health check command. How quickly do you want to detect and resolve a container failure?
- **Timeout** — Decide how long to wait for success or failure before retrying the health check command. How long should the health check command take to complete?
- **Retries** — How many times should the health check command be retried before registering a failure?

Set container dependencies

Within each container group you can set dependencies between containers based on container status. A dependency impacts when the dependent container can start or shut down based the status of another container.

A key use case for dependencies is to create startup and shutdown sequences for the container group.

For example, you might want Container A to start first and complete successfully before Containers B and C start. To achieve this, first create a dependency for Container B on Container A, with the condition that Container A must complete successfully. Then create a dependency for Container C on Container A with the same condition. Startup sequences occur in reverse order for shutdown.

Configure a container fleet

When you create a container fleet, consider the following decision points. Most of these points are dependent on your container architecture and configuration.

Decide where you want to deploy your fleet

In general, you want to deploy your fleets geographically near your players to minimize latency. You can deploy your container fleet to any AWS Region that Amazon GameLift Servers supports. If you want to deploy the same game server to additional geographic locations, you can add remote locations to the fleet including AWS Regions and Local Zones. For a multi-location fleet, you can adjust capacity independently in each fleet location. For more information about supported fleet locations, see [Amazon GameLift Servers service locations](#).

Consider using [UDP ping beacons](#) to collect network latency data in various geographical locations to anticipate the latency between player devices and potential fleet locations. These special endpoints accept UDP messages instead of traditional ICMP pings, providing accurate latency measurements to help you select optimal fleet locations.

Choose an instance type and size for your fleet

Amazon GameLift Servers supports a wide range of Amazon EC2 instances types, all of which are available for use with a container fleet. Instance type availability and price varies by location. You can view a list of supported instance types, filtered by location, in the Amazon GameLift Servers console (under **Resources, Instance and service quotas**).

When choosing an instance type, first consider the instance family. Instance families offer various combinations of CPU, memory, storage, and networking capabilities. Get more information on [EC2 instance families](#). Within each family you have a range of instance sizes to choose from. Consider the following issues when selecting an instance size:

- What's the minimum instance size that can support your workload? Use this information to eliminate any instance types that are too small.
- What instance type sizes are a good fit for your container architecture? Ideally, you want to choose a size that can accommodate multiple copies of your game server container group with minimal wasted space.
- What scaling granularity makes sense for your game? Scale fleet capacity involves adding or removing instances, and each instance represents the ability to host a specific number of game sessions. Consider how much capacity you want to add or remove with each instance. If player demand varies by thousands from minute to minute, then it might make sense to use

very large instances that can host hundreds or thousands of game sessions. By contrast, you might prefer more fine-grained scaling control with smaller instance types.

- Are there cost savings available based on size? You might find that the cost of certain instance types vary by location due to availability.

Set other optional fleet settings

You can use the following optional features when configuring a container fleet:

- Set up your game servers to access other AWS resources. See [Connect your Amazon GameLift Servers hosted game server to other AWS resources](#).
- Protect game sessions with active players from terminating prematurely during a scale-down event.
- Limit the number of game sessions that one individual can create on the fleet within a limited span of time.

Reduce game hosting costs with Spot fleets

When hosting multiplayer game servers using Amazon GameLift Servers managed hosting, Spot instances can provide a cost-effective alternative to On-Demand instances. The Spot pricing model offers the same hardware and performance as On-Demand but at potentially significant cost savings (up to 70-90%). However, they come with a limitation: when AWS needs capacity back, it can reclaim these instances with a two-minute interruption notification.

Amazon GameLift Servers mitigates the risk of interruptions for game server hosting. Amazon GameLift Servers predicts the likelihood of interruptions on Spot instance types and avoids placing game sessions on any instances at risk. If a rare interruption does occur, the notification allows you to gracefully end a game session for players.

How Amazon GameLift Servers works with Spot fleets

When you set up Spot fleets for game hosting, Amazon GameLift Servers continually evaluates your Spot fleet instance types and locations for game hosting viability.

- The Spot viability algorithm analyzes recent availability patterns and historical interruption rates of Spot instance types by location.
- Based on this analysis, Amazon GameLift Servers identifies Spot instance types and locations where there's an unacceptable potential for game session interruption. It takes the following actions:

- It marks the combination of instance type and location as temporarily non-viable.
- It removes any non-viable Spot fleet location from consideration when placing new game sessions. As a result, game sessions are only placed in Spot fleet locations that have a high likelihood of uninterrupted game server hosting.
- It drains the Spot fleet location of existing instances, even if AWS doesn't reclaim them, so you're not paying for instances that can't be used for game hosting. If game session protection is turned on, instances are shut down only after active game sessions are completed.
- Amazon GameLift Servers continually re-evaluates your Spot fleet instance types and locations for game hosting viability. When a previously non-viable instance type becomes viable again based on updated historical data, you can scale up the Spot fleet again, and Amazon GameLift Servers will resume placing game sessions with it.

Design considerations

When designing your solution to use Spot fleets, consider the following issues:

- **Assess game session length** – The average length of your game sessions can impact how well Spot works for your game. With shorter game sessions, the faster turnaround keeps game sessions running on viable instance types based on the latest historical data. Longer game sessions continue to run on instance types without evaluating recent viability data, running a greater risk of interruption over time.
- **Evaluate instance type availability** – Not every fleet location offers every instance type as Spot. When choosing an instance type for your Spot fleet, use the Amazon GameLift Servers console fleet creation tool to help you find Spot instance types in the locations you need. With this tool, you can select your fleet locations and then view instance type availability across those locations.
- **Create multi-location Spot fleets** – You can create Spot fleets with multiple locations. A single multi-location Spot fleet deploys instances with the same instance type to multiple AWS Regions or Local Zones. The Spot viability algorithm evaluates viability based on both instance type and location. If a Spot fleet location is evaluated as nonviable, it doesn't impact other locations in the fleet, which can still be used to host game sessions.
- **Create queues with Spot fleet diversity** – If you're using Spot fleets for game hosting, you need to set up a game session placement queue. For each new game session request, the queue looks for available game hosting resources and selects the best possible option. With Spot fleets, you want a queue that can search across multiple fleets that vary in both locations and instance types, and you want to include at least one On-Demand fleet as backup capacity. A well-designed multi-fleet queue that offers diverse placement options is highly resilient against

interruptions, slowdowns, and outages. For additional guidance on designing a queue for Spot, see [Build a queue for Spot Instances](#).

- **Handle interruptions gracefully** – Set up your game server to minimize player impact in the event of a Spot interruption. When AWS reclaims a Spot instance, Amazon GameLift Servers passes the termination notification to all affected server processes using the server SDK callback function `onProcessTerminate()`. Your game needs to implement this callback to end the game session gracefully. For more information, see [Respond to a server process shutdown notification](#).

 Note

AWS makes every effort to provide the notification before it reclaims an instance, but it's possible that AWS reclaims the Spot Instance before the warning arrives. You should also prepare your game server to handle unexpected interruptions.

- **Configure auto-scaling for your backup fleets to maintain service during a Spot interruption.** Target-tracking auto-scaling maintains a capacity buffer and scales automatically with demand. With auto-scaling, backup fleets (Spot or On-Demand) will begin increasing capacity whenever they start receiving more game session requests.

To quickly replace lost capacity when a Spot fleet becomes non-viable, a custom scaling mechanism can use available queue and fleet metrics to initiate a rapid scale-out of backup fleets. Detect when Spot fleets become non-viable with metrics such as `FirstChoiceOutOfCapacity`, `FirstChoiceNotViable`, and `PercentAvailableGameSessions`. Estimate replacement capacity needs by analyzing recent `PlacementsStarted` metric data. After scaling backup fleets to handle immediate demand, normal auto-scaling can take over.

- **Integration with FlexMatch** – If your solution uses a FlexMatch matchmaker, there are no special requirements for Spot fleets. You can configure a matchmaker to use a queue with Spot fleets. Amazon GameLift Servers automatically prioritizes match placements across Spot and On-Demand fleets, including when placing new game sessions and when backfilling empty player slots in existing game sessions.

Optimize game server runtime configuration on managed Amazon GameLift Servers

You can set up an managed EC2 fleet's runtime configuration to run multiple game server processes per instance. This uses your hosting resources more efficiently.

How a fleet manages multiple processes

Amazon GameLift Servers uses a fleet's runtime configuration to determine the type and number of processes to run on each instance. A runtime configuration contains at least one server process configuration that represents one game server executable. You can define additional server process configurations to run other types of processes related to your game. Each server process configuration contains the following information:

- The file name and path of an executable in your game build.
- (Optional) Parameters to pass to the process on launch.
- The number of processes to run concurrently.

When an instance in the fleet activates, it launches the set of server processes defined in the runtime configuration. With multiple processes, Amazon GameLift Servers staggers the launch of each process. Server processes have a limited life span. As they end, Amazon GameLift Servers launches new processes to maintain the number and type of server processes defined in the runtime configuration.

You can change the runtime configuration at any time by adding, changing, or removing server process configurations. Each instance regularly checks for updates to the fleet's runtime configuration to implement the changes. Here's how Amazon GameLift Servers adopts runtime configuration changes:

1. The instance sends a request to Amazon GameLift Servers for the latest version of the runtime configuration.
2. The instance compares its active processes to the latest runtime configuration, and then does the following:
 - If the updated runtime configuration removes a server process type, then active server processes of this type continue to run until they end. The instance doesn't replace these server processes.

- If the updated runtime configuration decreases the number of concurrent processes for a server process type, then excess server processes of this type continue to run until they end. The instance doesn't replace these excess server processes.
- If the updated runtime configuration adds a new server process type or increases the concurrent processes for an existing type, then the instance starts new server processes, up to the Amazon GameLift Servers maximum. In this case, the instance launches new server processes as existing processes end.

Optimize a fleet for multiple processes

To use multiple processes on a fleet, do the following:

- [Create a build](#) that contains the game server executables that you want to deploy to a fleet, and then upload the build to Amazon GameLift Servers. All game servers in a build must run on the same platform and use the server SDK for Amazon GameLift Servers.
- Create a runtime configuration with one or more server process configurations and multiple concurrent processes.
- Integrate game clients with the AWS SDK version 2016-08-04 or later.

To optimize fleet performance, we recommend that you do the following:

- Handle server process shutdown scenarios so that Amazon GameLift Servers can recycle processes efficiently. For example:
 - Add a shutdown procedure to your game server code that calls the server API `ProcessEnding()`.
 - Implement the callback function `OnProcessTerminate()` in your game server code to handle termination requests from Amazon GameLift Servers.
- Make sure that Amazon GameLift Servers shuts down and relaunches unhealthy server processes. Report the health status back to Amazon GameLift Servers by implementing the `OnHealthCheck()` callback function in your game server code. Amazon GameLift Servers automatically shuts down server processes that are reported unhealthy for three consecutive reports. If you don't implement `OnHealthCheck()`, then Amazon GameLift Servers assumes that a server process is healthy, unless the process fails to respond to a communication.

Choose the number of processes per instance

When deciding on the number of concurrent processes to run on an instance, keep in mind the following:

- Amazon GameLift Servers limits each instance to a [maximum number of concurrent processes](#). The sum of all concurrent processes for a fleet's server process configurations can't exceed this quota.
- To maintain acceptable performance levels, the Amazon EC2 instance type might limit the number of processes that can run concurrently. Test different configurations for your game to find the right number of processes for your preferred instance type.
- Amazon GameLift Servers doesn't run more concurrent processes than the total number configured. This means that the transition from the previous runtime configuration to the new configuration might happen gradually.

Work with the Amazon GameLift Servers Agent

The Amazon GameLift Servers Agent oversees the running of game server processes on your Amazon GameLift Servers fleets. The Agent is deployed to each compute in a fleet, and it provides automated process management, hosting management, and logging for the compute. To use the Agent, you must have your game server build integrated with the server SDK for Amazon GameLift Servers version 5.x or later.

The Amazon GameLift Servers Agent is externally available for use with Amazon GameLift Servers fleets that are not managed EC2 fleets. (Managed EC2 fleets handle the Agent's tasks automatically.) You can opt to run Amazon GameLift Servers fleets, including Anywhere fleets, with or without the Agent. Without the Agent, you must provide an alternative solution to completing the required tasks.

When deployed to a compute, the Amazon GameLift Servers Agent should be launched before any game server processes are started. On launch, the Agent completes the following tasks:

- Registers the compute with an Amazon GameLift Servers Anywhere fleet using the [RegisterCompute](#) API.
- Calls the [GetComputeAuthToken](#) API to fetch an authorization token and stores it for use by server processes that are running on the compute.
- Sets the WebSocket URL environment variable for the compute, and establishes a WebSocket connection to the Amazon GameLift Servers service.

- Requests the latest version of the fleet's runtime configuration from the Amazon GameLift Servers service.
- Starts and stops server processes according to the runtime configuration instructions.

Source code and build instructions for the Amazon GameLift Servers Agent are available in the [Amazon GameLift Servers Agent](#) GitHub.

About the Agent

The Amazon GameLift Servers Agent is designed to handle the following tasks for your fleets:

Process management

- Starts new server processes as defined in runtime instructions. The Agent might use a custom runtime configuration that is deployed with the Agent. Alternatively, you can provide a `RuntimeConfiguration` as part of your fleet definition. This approach has an advantage in that you can modify the fleet's runtime configuration at any time. The Agent periodically requests updated runtime configurations from the Amazon GameLift Servers service.
- Monitors server process activations and terminates processes when they don't activate in time.
- Sends heartbeats to Amazon GameLift Servers. If the Agent fails to send heartbeats, the compute might be marked as stale.
- Reports to Amazon GameLift Servers when a server process ends. Amazon GameLift Servers uses this information to monitor game server availability for game session placement.
- Emits fleet events for server processes, including:
 - `SERVER_PROCESS_INVALID_PATH`: The game server process launch parameters were incorrectly configured.
 - `SERVER_PROCESS_TERMINATED_UNHEALTHY`: The game server process did not report a valid health check within 3 minutes of activating and was therefore terminated.
 - `SERVER_PROCESS_FORCE_TERMINATED`: The game server process did not exit cleanly after `OnProcessTerminate()` was sent within 30 seconds.
 - `SERVER_PROCESS_CRASHED`: A game server process crashed for some reason.

Compute management

- Receives messages from the Amazon GameLift Servers service to shut down the compute.
- Prompts the compute to be terminated by Amazon GameLift Servers.

Logging

- Uploads logs to an Amazon S3 bucket in your AWS account.

Abstract an Amazon GameLift Servers fleet designation with an alias

An Amazon GameLift Servers *alias* is used to abstract a hosting destination. Hosting destinations tell Amazon GameLift Servers where to look for available resources to host a new game session for players. Aliases are useful in the following scenarios:

- If your game doesn't use multi-fleet queues for game session placement, it requests new game sessions by specifying an Amazon GameLift Servers fleet ID. During the life of a game, you'll replace the fleet multiple times, to update a server build, update hosting hardware and operating system, or resolve performance issues. Use an alias to abstract the fleet ID so that you can seamlessly switch player traffic from an existing fleet to the new one.
- If you want to do something other than create a new game session when a game client requests one. For example, you might want to direct players who are using an out-of-date client to an upgrade website.

An alias must specify a routing strategy. There are two types. A *simple* routing strategy routes player traffic to a specified fleet ID, which you can update to redirect traffic. A *terminal* routing strategy passes a message back to the client instead of creating a new game session. You can change an alias's routing strategy at any time.

If you use a queue for game session placement, you don't need an alias to redirect traffic when replacing a fleet. With a queue, you can simply add the new fleet and remove the old fleet. This action is not visible to players, because new game session requests are automatically fulfilled using the new fleet. It doesn't impact existing game sessions. You can identify queues destinations by using either a fleet ID or alias.

Topics

- [Create an Amazon GameLift Servers alias](#)

Create an Amazon GameLift Servers alias

This topic describes how to create an Amazon GameLift Servers alias for use with game session placement.

To create an alias

Use either the Amazon GameLift Servers console or the AWS Command Line Interface (AWS CLI) to create an alias.

Console

In the [Amazon GameLift Servers console](#), use the navigation pane to open the **Aliases** page.

1. Choose **Create alias**.
2. Enter an alias **Name**. We recommend including meaningful characteristics in the alias name to help when viewing a list of aliases.
3. Enter an alias **Description** as needed.
4. Choose a **Routing strategy** for the alias.
 - a. If you choose a **Simple** routing strategy, select a fleet ID from the list to associate with this alias. The list includes all fleets in the currently selected AWS Region. You must create an alias in the same Region as the fleet.
 - b. If you choose a **Terminal** routing strategy, enter a string value that you want Amazon GameLift Servers to return to a game client in response to a game session request. A request with a terminal alias throws an exception with the message embedded.
5. (Optional) Add **Tags** to the alias resource. Each tag consists of a key and an optional value, both of which you define. Assign tags to AWS resources that you want to categorize in useful ways, such as by purpose, owner, or environment. Choose **Add new tag** for each tag that you want to add.
6. When you're ready to deploy the new fleet, choose **Create**.

AWS CLI

Use the [create-alias](#) command to create an alias. Amazon GameLift Servers creates the alias resource in your current default AWS Region (or you can add a `--region` tag to specify a different AWS Region).

At minimum, include an alias name and routing strategy. For a simple routing strategy, specify the ID of a fleet in the same Region as the alias. For a terminal routing strategy, provide a message string.

Managing an Amazon GameLift Servers hosting solution (launch and beyond)

This section provides guidance on how to manage your Amazon GameLift Servers resources for the long term. Successfully launching your multiplayer game is just the beginning. Managing a production hosting solution requires ongoing attention to performance, cost optimization, security updates, and player experience. This section provides guidance on the operational aspects of running Amazon GameLift Servers in production, from day-one launch activities through long-term maintenance and optimization.

Whether you're preparing for launch, managing a live game, or optimizing an established hosting solution, these topics will help you maintain reliable, cost-effective game hosting that scales with your player base.

Manage game hosting resources with Amazon GameLift Servers

Manage Amazon GameLift Servers resources, including builds, fleets, game session queues, and matchmakers. You can use the Amazon GameLift Servers console to scan resource lists, or view resource details including metrics. You can also retrieve resource information using the AWS SDK or AWS CLI for Amazon GameLift Servers

Amazon GameLift Servers console features

Use the Amazon GameLift Servers console to view and manage your game hosting resources and ongoing hosting activity in near real time. The console offers a graphical interface for most of the functionality of the service API for Amazon GameLift Servers. You can use the console to:

- **Use the dashboard for a high-level snapshot.** You can see the numbers and current status of all your Amazon GameLift Servers hosting resources and follow links to get details on individual resources.
- **Manage individual hosting resources.** You can create, view, and delete all Amazon GameLift Servers resources, and update their mutable properties. You can also view certain types of hosting activity, such as events and performance metrics.
- **Interact with game and player session activity.** You can track game session and player session activity by fleet, and use this information to troubleshoot game session issues. View details on

a game sessions, view player sessions for each game session, and look up player activity across multiple game sessions. You can also shut down individual game sessions as needed.

Topics

- [Hosting dashboard in the Amazon GameLift Servers console](#)
- [Game server builds](#)
- [Fleets in the Amazon GameLift Servers console](#)
- [Aliases in the Amazon GameLift Servers console](#)
- [Game session queues in the Amazon GameLift Servers console](#)

Hosting dashboard in the Amazon GameLift Servers console

Use the Amazon GameLift Servers console dashboard to get a high-level snapshot about the current status of the Amazon GameLift Servers hosting resources in your AWS account. The **Amazon GameLift Servers dashboard** provides a view of the following:

- The number of builds in **Ready**, **Initialized**, and **Failed** statuses. Choose **View builds** for details about builds in your current Region.
- The number of fleets in all statuses. Choose **View fleets** for details about fleets in your current Region.
- Your current resources.
- New feature and service announcements.

To open the Amazon GameLift Servers dashboard

- In the [Amazon GameLift Servers console](#), in the navigation pane, choose **Dashboard**.

From the dashboard, you can:

- Prepare your game for launch by choosing **Prepare for launch** and filling out the corresponding launch questionnaire.
- Request service quota increases in preparation for launches or in response to launches by choosing **View service quotas**.

- View blog posts and detailed information about new features by choosing the link in the **Features spotlight**.

GameLift > Dashboard

Dashboard

Build status overview

Viewing data for all builds in N. Virginia region.

View builds

Ready
 1

Initialized
 0

Failed
 0

Fleet status overview

Viewing data for all fleets in N. Virginia region.

View fleets

Active
 0

New
 0

Deleting
 0

Error
 0

In progress
 0

Terminated
 0

Resources (1)

View service quotas

Resource type	Count
Builds	1
Scripts	0
Fleets	0
Aliases	0
Queues	0
Matchmaking rule sets	0
Matchmaking configurations	0

Prepare for your game launch

[Learn more](#)

Fill out a launch questionnaire

Fill out our game launch questionnaire and email it to the GameLift launch team to ensure a smooth launch. The GameLift launch team will verify your GameLift setup and service limits, preparing you for launch.

[Prepare to launch](#)

Features spotlight

Updates on features available in N. Virginia region

March 22, 2022

Updates to Amazon GameLift FlexMatch for greater flexibility

October 28, 2021

New Asia Pacific (Osaka) region and Graviton2 support for Amazon GameLift

Game server builds

The build resource represents your game server software. You upload your build package to Amazon GameLift Servers for deployment to managed EC2 fleets.

View information about game server builds in the Amazon GameLift Servers console or using the or AWS SDK for Amazon GameLift Servers.

Console

On the **Builds** page of the [Amazon GameLift Servers console](#), you can view information about and manage all the game server builds that you've uploaded to Amazon GameLift Servers for deployment on managed EC2 fleets. In the navigation pane, choose **Hosting, Managed EC2, Builds**.

The **Builds** page shows the following information for each build. You can adjust the table content as needed using the **Preferences** tool (see the



icon in the upper right corner of the table). Custom preferences are saved to your AWS account user and are automatically applied whenever you view this page.

Note

The **Builds** page shows builds in your current AWS Region only.

- **Name** – The name associated with the uploaded build.
- **Status** – The status of the build. Displays one of three status messages:
 - **Initialized** – The upload hasn't started or is still in progress.
 - **Ready** – The build is ready for fleet creation.
 - **Failed** – The build timed out before Amazon GameLift Servers received the binaries.
- **Creation time** – The date and time that you uploaded the build to Amazon GameLift Servers.
- **Build ID** – The unique ID assigned to the build on upload.
- **Version** – The version label associated with the uploaded build.
- **Operating system** – The OS that the build runs on. The build OS determines which operating system Amazon GameLift Servers installs on a fleet's instances.
- **Size** – The size, in megabytes (MB), of the build file uploaded to Amazon GameLift Servers.
- **Fleets** – The number of fleets deployed with the build.

From this page you can do any of the following:

- View build details. Choose a build's name to open its build details page.
- Create a new fleet from a build. Select a build, and then choose **Create fleet**.

- Filter and sort the build list. Use the controls at the top of the table.
- Delete a build. Select a build, and then choose **Delete**.

Build details

On the **Builds** page, choose a build's name to open its details page. The **Overview** section of the details page displays the same build summary information as the **Builds** page. The **Fleets** section shows a list of fleets that are running the build, including the same summary information as the [Fleets page](#).

AWS SDK

Use the following AWS CLI commands to retrieve information about this resource:

- [ListBuild](#)
- [DescribeBuild](#)

Update a game server build

When you deploy your game server build for Amazon GameLift Servers managed EC2 hosting, you upload your game server software and create an Amazon GameLift Servers build resource. After you've created an Amazon GameLift Servers build, you can update the build's metadata, but you can't update the build files themselves. To deploy updates to your game server build, you must upload a new set of files and create a new Amazon GameLift Servers build using the AWS CLI [upload-build](#) command. Alternatively, you can use the [create-build](#) command to upload a new build from an Amazon S3 bucket that you control. Then deploy the new build by creating a new fleet for it.

You can update a build's metadata, including the name and description. For these tasks, use the Amazon GameLift Servers console or the [update-build](#) AWS CLI command.

Automate your game build updates

Follow these tips to help automate and streamline the process of updating game server builds for Amazon GameLift Servers managed fleets:

- **Use game session queues and swap out fleets as needed.** When sending game session requests to Amazon GameLift Servers, specify a game session queue instead of a specific fleet.

With queues, you can add fleets with a new build and remove old fleets as needed. For more information, see [Configure game session placement](#).

- **Use aliases to transfer players to a new game build.** When sending game session requests to Amazon GameLift Servers, specify a fleet alias instead of a fleet ID. For more information, see [Create an Amazon GameLift Servers alias](#).
- **Set up for iterative development.** During game development, explore options for setting up a hosted test environment that supports rapid iterative development. See [Set up for iterative development with Amazon GameLift Servers Anywhere](#).

Try out these resources from the [Amazon GameLift Servers Toolkit](#) on GitHub:

Fast build update tool (for development only)

This tool helps you modify game server builds that are already deployed on computes in a managed EC2 fleet, saving you time during fast development iteration. The tool has several options; you can replace the entire game build or change specific files, and you can manage how to restart game server processes after the updates. You can also use it to update all computes in a fleet or target individual computes.

Visit the Amazon GameLift Servers Toolkit repo in GitHub to get the [fast build update tool](#) in GitHub and learn more about how to use it.

Production deployment sample script

This script illustrates how you can automate the process of updating game server builds that are deployed on managed EC2 fleets in production. To use this script, your Amazon GameLift Servers hosting solution must use aliases to abstract fleet IDs. The sample script handles the following basic steps: upload an updated build, create a new build and deploy to a new fleet, redirect player traffic from an existing fleet to the new fleet, and delete the old fleet. Customize the sample script to meet your specific deployment requirements.

Visit the Amazon GameLift Servers Toolkit repo in GitHub to get the [production deployment sample script](#) in GitHub and learn more about how to use it.

Fleets in the Amazon GameLift Servers console

The fleet resource represents a set of compute machines that are deployed with and run your game servers. Depending on your hosting solution, you might have managed EC2 fleets, managed container fleets, Anywhere fleets, or a combination.

View information about fleets in the Amazon GameLift Servers console or using the or AWS SDK for Amazon GameLift Servers.

Console

You can view information on all the fleets created to host your games on Amazon GameLift Servers under your AWS account. In the console's left-side navigation, find the hosting option you want (Anywhere, Managed EC2, Managed containers), and choose **Fleets**.

For each hosting option, the **Fleets** page lists fleets that are located in your currently selected AWS Region. From the **Fleets** page, you can create a new fleet or view additional detail on a fleet. A fleet's [detail page](#) contains usage information, metrics, game session data, and player session data. You can also edit a fleet record or delete a fleet.

A **Fleets** page displays the following summary information by default. You can adjust the table content as needed using the **Preferences** tool (see the



icon in the upper right corner of the table). Custom preferences are saved to your AWS account user and are automatically applied whenever you view this page.

- **ID** – An identifier assigned to the fleet. This ID is unique within the AWS Region where the fleet is created.
- **Name** – A friendly name given to the fleet.
- **Status** – The status of the fleet: **New**, **Downloading**, **Building**, and **Active**.
- **Creation time** – The date and time the fleet was created.

Note

A fleet displays a warning icon for fleets that were created more than 90 days ago. As a best practice, we recommend replacing fleets every 30 days to maintain a secure and up-to-date runtime environment for your hosted game servers. For guidance, see [Security best practices for Amazon GameLift Servers](#).

- **Fleet type** – The availability of the instances used to host your games, which can potentially impact hosting costs. A managed fleet can use **On-Demand** (always available) or **Spot** (availability varies) instances.
- **Active instances** – The number of EC2 instances in use for the fleet.

- **Desired instances** – The number of EC2 instances to keep active.
- **Game sessions** – The number of active game sessions running in the fleet. The data is delayed by five minutes.
- **Player sessions** – The number of active player sessions in the fleet. The data is delayed by five minutes.

AWS SDK

Use the following AWS CLI commands to retrieve information about this resource:

- Managed EC2 fleets
 - [ListFleets](#)
 - [ListCompute](#)
 - [DescribeFleetAttributes](#)
 - [DescribeCompute](#)
 - [DescribeFleetCapacity](#)
 - [DescribeFleetUtilization](#)
- Managed container fleets
 - [ListContainerFleets](#)
 - [DescribeContainerFleet](#)
 - [ListContainerGroupDefinitions](#)
 - [DescribeContainerGroupDefinition](#)
- - [ListFleets](#)
 - [ListCompute](#)
 - [DescribeFleetAttributes](#)
 - [DescribeCompute](#)

Update an Amazon GameLift Servers fleet configuration

Use the Amazon GameLift Servers console or the AWS CLI to update your fleet settings, change remote locations, or delete a fleet. For managed fleets, you can't change a fleet's game server build or instance type. Instead, you must replace the fleet.

Fast Build Update Tool (for development only)

With managed EC2 fleets, to deploy a game server build update, you need to upload each new build to Amazon GameLift Servers and create a new fleet for it.

The Fast Build Update Tool lets you can bypass these steps during development, saving you time and allowing for faster development iteration. With this tool, you can quickly update your game build files across all computes in an existing fleet. The tool has several options; you can replace an entire game build or change 6 specific files, and you can manage how to restart game server processes after the updates. You can also use it to update individual computes in a fleet.

To get the Fast Build Update Tool and learn more about how to use it, visit the Amazon GameLift Servers Toolkit repo for [The Fast Build Update Tool](#) in Github.

You can update mutable fleet attributes, port settings, and runtime configurations using the Amazon GameLift Servers console or the AWS CLI. To change scaling limits, see [Auto-scale fleet capacity with Amazon GameLift Servers](#).

Console

1. In the [Amazon GameLift Servers console](#), in the navigation pane, choose **Fleets**.
2. Choose the fleet you want to update. A fleet must be in ACTIVE status before you can edit it.
3. On the Fleet detail page, in any of the following sections, choose **Edit**.
 - **Fleet settings**
 - Change the fleet attributes such as **Name** and **Description**.
 - Add or remove **Metric groups**, that Amazon CloudWatch uses to track aggregated Amazon GameLift Servers metrics for multiple fleets.
 - Update **Resource creation limit** settings.
 - Turn game session protection on or off.
 - **Runtime configuration** – You can change any of the following settings of your runtime configurations and add or remove runtime configurations.
 - Change the **Launch path** of your game server.
 - Add, remove, or change optional **Launch parameters**.

- Change the number of **Concurrent processes** that your game servers run.
 - **Game session activation** – Change how you want server processes to run and host game sessions by updating **Max concurrent game session activations** and **New activation timeout**.
 - **EC2 port settings** – Update the IP addresses and port ranges that allow inbound access to the fleet.
4. Choose **Confirm** to save changes.

AWS CLI

Use the following AWS CLI commands to update a fleet:

- [update-fleet-attributes](#)
- [update-fleet-port-settings](#)
- [update-runtime-configuration](#)

Update fleet locations

You can add or remove a fleet's remote locations using the Amazon GameLift Servers console or the AWS CLI. You can't change a fleet's home Region.

Console

1. In the [Amazon GameLift Servers console](#), in the navigation pane, choose **Fleets**.
2. Choose the fleet you want to update. A fleet must be in ACTIVE status before you can edit it.
3. On the Fleet detail page, choose the **Locations** tab to view the fleet's locations.
4. To add new remote locations, choose **Add** and select the locations you want to deploy instances to. This list doesn't include instances where the fleet's instance type isn't available.
5. With new locations selected, choose **Add**. Amazon GameLift Servers adds the new locations to the list, with status set to NEW. Amazon GameLift Servers then begins provisioning an instance in each added location and preparing it to host game sessions.
6. To remove existing remote locations from the fleet, use the check boxes to select one or more listed locations.

7. With one or more fleets selected, choose **Remove**. The removed locations remain in the list, with status set to DELETING. Amazon GameLift Servers then begins the process of terminating activity in the removed location. If there are active instances that are hosting game sessions, Amazon GameLift Servers uses the game server termination process to gracefully end game sessions, terminate game servers, and shut down instances.

AWS CLI

Use the following AWS CLI commands to update fleet locations:

- [create-fleet-locations](#)
- [delete-fleet-locations](#)

Delete a fleet

You can delete a fleet when you no longer need it. Deleting a fleet permanently removes all data associated with game sessions and player sessions, and collected metric data. As an alternative, you can retain the fleet, disable auto scaling, and manually scale the fleet to 0 instances.

Note

If the fleet has a VPC peering connection, first request authorization by calling [CreateVpcPeeringAuthorization](#). Amazon GameLift Servers deletes the VPC peering connection during fleet deletion.

You can use either the Amazon GameLift Servers console or the AWS CLI tool to delete a fleet.

Console

1. In the [Amazon GameLift Servers console](#), in the navigation pane, choose **Fleets**.
2. Choose the fleet you want to delete. You can only delete fleets in ACTIVE or ERROR status.
3. Choose **Delete**.
4. In the **Delete fleet** dialog box, confirm the deletion by entering **delete**.
5. Choose **Delete**.

AWS CLI

Use the following AWS CLI command to delete a fleet:

- [delete-fleet](#)

Fleet details in the Amazon GameLift Servers console

Access a **Fleet** detail page from the dashboard or the **Fleets** page by choosing the fleet name.

On the fleet details page you can take the following actions:

- Update a fleet's attributes, port settings, and runtime configuration.
- Add or remove fleet locations.
- Change fleet capacity settings.
- Set or change target-tracking auto scaling.
- Delete a fleet.

Details

Fleet settings

- **Fleet ID** – An identifier assigned to the fleet. This ID is unique within the AWS Region where the fleet is created.
- **Fleet name** – A friendly name given to the fleet.
- **ARN** – A unique identifier assigned to this fleet. A fleet's ARN identifies it as an Amazon GameLift Servers resource and specifies the region and AWS account.
- **Description** – A short identifiable description of the fleet.
- **Status** – Current status of the fleet, which may be **New**, **Downloading**, **Building**, and **Active**.
- **Creation time** – The date and time when the fleet was created.

Note

A fleet displays a warning icon for fleets that were created more than 90 days ago. As a best practice, we recommend replacing fleets every 30 days to maintain a secure and up-

to-date runtime environment for your hosted game servers. For guidance, see [Security best practices for Amazon GameLift Servers](#).

- **Compute type** – The type of compute used to host your games. A fleet can be a **Managed EC2**, **Managed container**, or **Anywhere** fleet.
- **Operating system** – The Amazon Machine Image (AMI) that's deployed to every instance in the fleet. The AMI version is the latest version available at the time the fleet was created.
- **Certificate type** – The status of the fleet's ability to use a TLS certificate for authenticating a game server and encrypting all client/server communication.
- **Fleet type** – The availability of the instances used to host your games, which can potentially impact hosting costs. A fleet can use **On-Demand** (always available) or **Spot** (availability varies) instances.
- **EC2 instance type** – Amazon EC2 [instance type](#) selected for the fleet when it was created.
- **Instance role ARN** – An AWS IAM role that manages access to your other AWS resources, if one was provided during fleet creation.
- **Instance role credentials provider** – An AWS IAM role that manages access to your other AWS resources, if one was provided during fleet creation.
- **Metric group name** – The group used to aggregate metrics for multiple fleets.
- **Game scaling protection policy** – Indicates whether the fleet's game session protection is enabled, which prevents active game sessions from ending prematurely during a scale-down event.
- **Maximum game sessions per player** – The maximum number of sessions a player can create during the **Policy period**.
- **Policy period** – The length of time used to limit a player's number of sessions created.

Build details

The **Build details** section displays the build hosted on the fleet. Select the build name to see the full build detail page.

Runtime configuration

The **Runtime configuration** section displays the server processes to launch on each instance. It includes the path for the game server executable and optional launch parameters.

Game session activation

The **Game session activation** section displays the number of server processes that launch at the same time and how long to wait for the process to activate before terminating it.

EC2 port settings

The **Ports** section displays the fleet's connection permissions, including IP address and port setting ranges.

Metrics

The **Metrics** tab displays a graphical representation of fleet metrics over time. For more information about using metrics in Amazon GameLift Servers, see [Monitor Amazon GameLift Servers with Amazon CloudWatch](#).

Events

The **Events** tab provides a log of all events that have occurred on the fleet, including the event code, message, and time stamp. See [Event](#) descriptions in the Amazon GameLift Servers API Reference.

Scaling

The **Scaling** tab contains information about fleet capacity, including the current status and capacity changes over time. It also provides tools to update capacity limits and manage auto scaling.

Scaling capacity

View current fleet capacity settings for each fleet location. For more information about changing limits and capacity, see [Scaling game hosting capacity with Amazon GameLift Servers](#).

- **AWS Location** – Name of a location where fleet instances are deployed.
- **Status** – Hosting status of the fleet location. Location status must be ACTIVE to be able to host games.
- **Min size** – The smallest number of instances that must be deployed in the location.
- **Desired instances** – The target number of active instances to maintain the location. When active instances and desired instances aren't the same, a scaling event is started to start or shut down instances as needed until active instances equals desired instances.

- **Max size** – The most instances that can be deployed in the location.
- **Available** – The service limit on instances minus the number of instances in use. This value tells you the maximum number of instances that you can add to the location.

Auto scaling policies

This section covers information about auto scaling policies that are applied to the fleet. You can set up or update a target-based policy. The fleet's rule-based policies, which must be defined using the AWS SDK or CLI, are displayed here. For more information about scaling, see [Auto-scale fleet capacity with Amazon GameLift Servers](#).

Scaling history

View graphs of capacity changes over time.

Locations

The **Locations** tab lists all locations where fleet instances are deployed. Locations include the fleet's home Region and any remote locations that have been added. You can add or remove locations directly in this tab.

- **Location** – Name of a location where fleet instances are deployed.
- **Status** – Hosting status of the fleet location. Location status tracks the process of activating the first instances in the location. Location status must be ACTIVE to be able to host games.
- **Active instances** – The number of instances with server processes running on the fleet location.
- **Active servers** – The number of game server processes able to host game sessions in the fleet location.
- **Game sessions** – The number of game sessions active on instances in the fleet location.
- **Player sessions** – The number of player sessions, which represent individual players, that are participating in game sessions that are active in the fleet location.

Game sessions

The **Game sessions** tab lists past and present game sessions hosted on the fleet, including some detail information. Choose a game session ID to access additional game session information, including player sessions. For more information about player sessions, see [Game and player sessions in the Amazon GameLift Servers console](#).

Game and player sessions in the Amazon GameLift Servers console

You can use the Amazon GameLift Servers console to work with games sessions and player sessions. For more information about game sessions and player sessions, see [Amazon GameLift Servers and the player experience](#). The Amazon GameLift Servers console provides information and tools to help you investigate issues with your game sessions.

What you can do:

- Explore game session and player session activity that is hosted on a specific fleet.
- Look up game session activity for a specific player across multiple fleets.
- Shut down a specific game session.

View game session detail

Game session and player session data is organized by the fleet that hosts the game session.

To access game session and player session information

1. In the [Amazon GameLift Servers console](#), open the left navigation pane. Select a hosting solution type and open the **Fleets** page. For example:
 - **Hosting, Anywhere, Fleets**
 - **Hosting, Managed EC2, Fleets**
 - **Hosting, Managed containers, Fleets**
2. Each **Fleets** page displays the list of fleets for your currently selected AWS Region. Choose the fleet that you want to view game session data for.
3. In the fleet's detail page, open the **Game sessions** tab. This tab lists all game sessions that were hosted on the fleet, along with summary information. You can adjust the table content as needed using the **Preferences** tool (see the  icon in the upper right corner of the table). Custom preferences are saved to your AWS account user and are automatically applied whenever you view this page.
4. Choose a game session from the list to view additional information.
5. If the game session includes player session data, choose **View player sessions** to open the player sessions lookup tool with the game session ID automatically filled in.

The **Game sessions** detail include the following information:

- **Status** – Game session status.
 - **Activating** – The instance is initiating a game session.
 - **Active** – A game session is running and available to receive players, depending on the session's [player creation policy](#).
 - **Terminated** – the game session has ended.
- **ARN** – The Amazon Resource Name of the game session.
- **Name** – Name generated for the game session.
- **Location** – The location that Amazon GameLift Servers hosted the game session in.
- **Creation time** – Date and time that Amazon GameLift Servers created the stream session.
- **Ending time** – Date and time that the game session ended.
- **DNS name** – The host name of the game session.
- **IP address** – IP address specified for the game session.
- **Port** – Port number used to connect to the game session.
- **Creator ID** – A unique identifier of the player that initiated the game session.
- **Player session creation policy** – Indicates if the game session is accepting new players.
- **Game scaling protection policy** – The type of game session protection to set on all new instances that Amazon GameLift Servers starts in the fleet.

Game data

Game property data, formatted as a string, to send to your game session on start.

Game properties

Game property data, formatted as key/value pairs, to send to your game session on start.

Matchmaking data

If the game session was created with FlexMatch, matchmaking data describes information about the matchmaking configuration and rule set. This includes each match's player attributes and team assignments. Data is in JSON format. For more information about FlexMatch matchmaking, see [Build a matchmaker](#).

Look up player session data

If your game hosting solution uses player sessions and provides unique player IDs, you can explore player-specific activity for past or present game sessions across multiple fleets. Open the Player session lookup tool using either of the following methods:

- In the Amazon GameLift Servers console, open the left navigation pane and choose **Player session lookup** and select the filter type to use.
- When viewing a fleet's game session details, choose **View player sessions**. The lookup tool opens with the game session with the game session ID filter preselected and the game session value filled in.

When using the lookup tool, you can provide any the following information:

- A player session ID to get information on a specific player session.
- A game session ID to get information on all player sessions for the requested game session. The results represent all players who reserved a slot or connected to the game session. You can optionally filter results by player session status.
- A player ID to get information on all player sessions for the requested player. The results represent all game sessions that the player participated in.

Note

The lookup tool searches for all player session activity across the currently selected AWS Region. If you have multiple fleets in the Region, the results include player session activity across all fleets. For multi-location fleets, the results also include player session activity across the fleets' remote locations.

The following player session data is collected for each game session:

- **Player session ID** – The identifier assigned to the player session.
- **Player ID** – A unique identifier for the player. Choose this ID to get additional player information.
- **Game session ID** – The identifier assigned to the game session.
- **Fleet ID** – The identifier assigned to the fleet that hosted the game session.

- **Status** – The status of the player session. The following are possible statuses:
 - **Reserved** – Player session has been reserved, but the players isn't connected.
 - **Active** – Player session is connected to the game server.
 - **Completed** – Player session has ended; player is no longer connected.
 - **Timed Out** – Player failed to connect.
- **Creation time** – The time the player connected to the game session.
- **Ending time** – The time the player disconnected from the game session.
- **Connection data** – IP address, DNS name and port that the player used to connect to the game session.
- **Player data** – Information about the player that was provided during player session creation.

Shut down a game session

Use the Amazon GameLift Servers console to shut down a specific game session. This feature gives you a simple and fast method for locating a game session and sending a signal to terminate it. Another termination method requires that you find the fleet instance where the game session is running, remotely access the instance, and manually shut down the game session.

You can shut down a game session for any reason. The most common reason is to resolve a game session that's failing to shut down naturally. As a result, the hosting resource for the game session can't be freed to host a new game session, and the fleet's hosting capacity is degraded.

Note

This feature relies on certain configuration settings for your hosting solution. It has the following limitations:

- The game session must be hosted on a fleet that's running a game server build with server SDK for Amazon GameLift Servers v5 or greater. If your game servers are deployed with an older version, you need to use remote access to delete the game session.
- If the game session is hosted on an Anywhere fleet, the fleet must be using the Amazon GameLift Servers Agent to manage game server processes.

To terminate a game session

1. In the [Amazon GameLift Servers console](#), open the left navigation pane. Select a hosting solution type and open the **Fleets** page. For example:
 - **Hosting, Anywhere, Fleets**
 - **Hosting, Managed EC2, Fleets**
 - **Hosting, Managed containers, Fleets**
2. Each **Fleets** page displays the list of fleets for your currently selected AWS Region. Choose the fleet that's hosting the game session that you want to terminate.
3. In the fleet's detail page, open the **Game sessions** tab. In the list of game sessions, select the one that you want to terminate, and choose the **Terminate** button.
4. In the **Terminate game session?** window, verify that you're shutting down the right game session and choose a termination method.
 - Normal game session shutdown – This option sends a signal to the server process that's hosting the game session to shut down. If your game server build was properly integrated for Amazon GameLift Servers, the server process initiates its game session shutdown sequence, notifies Amazon GameLift Servers that it's ending, and stops. Depending on your game design, the shutdown sequence might include steps to gracefully complete the game session, such as saving data and notifying active players. This method might require a small delay to complete the game session shutdown sequence.
 - Immediate game session shutdown – This option sends a signal to a process manager to shut down the server process that's hosting the game session. This option bypasses the normal game session shutdown. It's able to terminate the game session even when the server process is unable to respond.
5. Confirm game session termination. You can track the shutdown progress on the **Game sessions** console page. The game session status will change to "Terminating", and then to "Terminated" when shutdown is complete.

Related topics

- You can also shutdown game sessions using the AWS SDK and the AWS CLI. For more details and examples, see the Amazon GameLift Servers API Reference topic [TerminateGameSession](#).

- For more information on game server integration and how a server process responds to signals from the Amazon GameLift Servers service, see [Add Amazon GameLift Servers to your game server with the server SDK](#).

Update an Amazon GameLift Servers managed container fleet

You can update most of the properties of a managed container fleet, including the container group definitions. Depending on the settings being updated, a fleet update might initiate a new fleet deployment. In a fleet deployment, all instances in the fleet are removed and replaced with instances with the new configuration. Settings that require a deployment include:

- Container group definitions, including updates to container images
- Connection port ranges and inbound permissions
- Log configuration

You can track the status of fleet deployments in the [Amazon GameLift Servers console](#) or the AWS Command Line Interface (AWS CLI) to create a container fleet.

Note

You can't update a fleet's runtime environment. At fleet creation, the fleet's Amazon Machine Image (AMI) is set to the latest available version of the Linux AMI. All container images that are deployed to this fleet must be compatible with this version. To change the fleet's AMI or upgrade to a newer version, you must create a new fleet. As a best practice, we recommend replacing your fleets every 30 days to maintain a secure and up-to-date runtime environment for your hosted game servers. For more guidance, see [Security best practices for Amazon GameLift Servers](#).

Console

In the [Amazon GameLift Servers console](#), select the AWS Region where you want to create the fleet. The container group definitions must be in the same region where you want to create the fleet.

Open the console's left navigation bar and choose **Managed containers: Fleets**. On the managed container fleets page, select a fleet from the list and choose **Edit**.

1. Update container fleet settings as needed. When you're finished, choose **Create**.
2. If your updates require a fleet deployment, you're asked to specify deployment options as follows:
 - Game session protection. You can choose to protect fleet instances that have active game sessions (safe deployment). With this setting, the fleet instances aren't replaced until after the game sessions end. Alternatively, you can choose to replace fleet instances regardless of game session activity (unsafe deployment). Unsafe deployments are useful during development and testing phases in order to reduce deployment time.
 - Minimum healthy percentage. You can manage how quickly the fleet's instances are replaced. Use this setting to maintain a minimal amount of healthy tasks the during deployment. A low value prioritizes deployment speed, while a high value ensures that game server availability remains high throughout the deployment.
 - Deployment failure strategy. Decide what actions to take if a deployment fails. A deployment failure means that some of the updated containers have failed status checks and are considered impaired. You can set deployments to automatically roll back all fleet instances to the previously deployed state. Alternatively you can choose to maintain some of the impaired fleet instances for use in debugging.

If your request is successful, the console displays the **Deployments** tab for the managed container fleet. Use this tab to track the status of each deployment. If you start a new deployment for the fleet, this action automatically cancels any deployment that is currently in process for the fleet.

AWS CLI

To create a container fleet with the AWS CLI, open a command line window and use the `update-container-fleet` command. For more information about this command, see [update-container-fleet](#) in the *AWS CLI Command Reference*.

The following example updates an existing container fleet with the following characteristics:

- It updates the game server container group definition to use version 2.
- It specifies safe deployment options.

```
{  
  "DeploymentConfiguration": {
```

```
"ImpairmentStrategy": "ROLLBACK",
"MinimumHealthyPercentage": 75,
"ProtectionStrategy": "WITH_PROTECTION"
},
"FleetId": "containerfleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
"GameServerContainerGroupDefinitionName": "arn:aws:gamelift:us-
west-2:111122223333:containergroupdefinition/MyAdventureGameContainerGroup:2"
}
```

Update a container group definition for an Amazon GameLift Servers container fleet

You can update most of the properties of a container group definition, including the individual container definitions. Container group definitions have a version number. When you update a container group definition, Amazon GameLift Servers saves the update and increments the definition's version number. When configuring a container fleet, you can specify which version of a container group definition to deploy.

After updating a container group definition, you can deploy the new version to a new or existing container fleet.

Update a game server container group definition

This topic describes how to update game server container group definition using the Amazon GameLift Servers console or AWS CLI tools. For more detailed information on optional features, see [Customize an Amazon GameLift Servers container fleet](#).

To update a container group definition:

Console

In the [Amazon GameLift Servers console](#), select the AWS Region where you want to create the container group.

Open the console's left navigation bar and choose **Managed containers: Group definitions**. On the Container groups definition page, choose a container group definition and version to update.

After you've saved your updates, you can use the new version to create a new container fleets or you can deploy the updates to an existing container fleet.

Step 1: Define container group definition details

- You can update the total memory and vCPU limit settings.

Step 2: Add container definitions

You can make the following container definition updates:

- Update existing container definitions.
 - Add new support container definitions.
 - Remove support container definitions.
1. You can update the **ECR image URI**. Make sure to update the **Server SDK version** setting to match the new image.
 2. You can update the **Internal container port range** as needed. Changes you make to these settings might impact a container fleet's connection port settings when these changes are deployed to a fleet. For more details, see [Configure network connections](#).

Step 3: Configure dependencies

- You can change dependencies as needed. For more information, see [Set container dependencies](#).

Step 3: Review and create

- Review your container group definition updates. Use **Edit** to make additional changes in any section. When you're finished, choose **Create** to generate a new version of the container group definition.

If your request is successful, the console displays the detail page for the new container group definition resource. Initially the status is **COPYING**, as Amazon GameLift Servers starts taking snapshots of all the container images for the group. When this phase is complete, the container group definition status changes to **READY**. A container group definition must be in **READY** status before you can create a container fleet with it.

AWS CLI

When you use the AWS CLI to create or update a container group definition, maintain your container definition configurations in a separate JSON file. You can reference the file in your CLI command. See [Create a container definition JSON file](#) for schema examples.

When updating a definition, you only need to specify the values you want to update. Amazon GameLift Servers retains any values that you don't include in your update request. If you're changing a container definition. However, when changing a container definition, provide a complete set.

To update a container group definition

To update a new container group definition, use the `update-container-group-definition` CLI command. For more information about this command, see [update-container-group-definition](#) in the *AWS CLI Command Reference*.

Example: game server container group

You can specify a container group definition version when retrieving, updating, or deleting a container group definition, or when creating or updating a container fleet. Each container group definition has a version property. In addition, the definition's ARN value specifies the version number.

This example illustrates a request for a change to a game server container group definition. It assumes that you've created a JSON file with the container definitions for this group. This example uses the ARN value for definition name, and specifies that the update is to version 1.

```
aws gamelift update-container-group-definition \
  --name arn:aws:gamelift:us-west-2:111122223333:containergroupdefinition/
  MyAdventureGameContainerGroup:1 \
  --operating-system AMAZON_LINUX_2023 \
  --container-group-type GAME_SERVER \
  --total-memory-limit-mebibytes 4096 \
  --total-vcpu-limit 1 \
  --container-definitions file://SimpleServer.json
```

Clone a container group definition

You can use the Amazon GameLift Servers console to clone an existing container group definition.

To clone a container group

1. In the [Amazon GameLift Servers console](#), go to the left navigation pane and choose **Container groups**.
2. On the **Container groups** list page, select the existing container group that you want to clone. After you select a container group, the **Clone** button is active.
3. Choose **Clone**. This action opens the container group creation wizard with pre-filled settings.
4. Enter a new name for the cloned container group. Container group in the same region must have unique names.
5. Step through the container group and container definition pages, review, and **Create** the new container group.

Delete a container group definition for an Amazon GameLift Servers container fleet

You have several options for deleting a container group definition. When you delete a container group definition, this action also deletes all the container definitions in the container group.

Container group definitions can have multiple versions. Container group versions have the same name, but have a different version number. Container group definition ARNs specify both the name and version.

You can specify a container group definition version when retrieving, updating, or deleting a container group definition, or when creating or updating a container fleet. Each container group definition has a version property. In addition, the definition's ARN value specifies the version number.

There are several ways to delete container group definitions:

- You can delete all versions of a specific definition.
- You can delete one particular version of a specific definition.
- You can keep some number of newest versions and delete the older versions of a specific definition. For example, you might delete all versions older than version 5.

You can delete a container group definition version only if it's not being used in a container fleet.

To delete a container group definition

Console

In the [Amazon GameLift Servers console](#), select the AWS Region where you want to create the container group.

Open the console's left navigation bar and choose **Managed containers: Group definitions**. On the Container group definitions page, select the definition you want to modify and choose **Delete**.

You're prompted to select the type of deletion you want to make and specify additional settings depending on the delete type.

AWS CLI

- To delete a container group definition, use the `delete-container-group-definition` CLI command and provide values for the type of delete you want to do. For more information about this command, see [delete-container-group-definition](#) in the *AWS CLI Command Reference*.

This example illustrates a request to delete all versions of game server container group definition older than version 5.

Example

```
aws gamelift delete-container-group-definition \  
  --name MyAdventureGameContainerGroup \  
  --version-count-to-retain 5 \  
  --delete-type AllVersionsOlderThanVersion
```

Aliases in the Amazon GameLift Servers console

The alias resource represents a pointer to a fleet or terminal destination. They are used to direct player traffic to specific hosting destinations. Use an alias to abstract a destination so that you can more easily redirect traffic when you update your hosting fleets.

View information about aliases in the Amazon GameLift Servers console or using the or AWS SDK for Amazon GameLift Servers.

Console

The **Alias** page displays information about Amazon GameLift Servers aliases that direct traffic specific hosting destinations. To view aliases, choose **Hosting, Aliases** in the navigation pane.

You can do the following on the aliases page:

- Create a new alias. Choose **Create alias**.
- Filter and sort the aliases table. Use the controls at the top of the table. You can adjust the table content as needed using the **Preferences** tool (see the  icon in the upper right corner of the table). Custom preferences are saved to your AWS account user and are automatically applied whenever you view this page.
- View alias details. Choose an alias name to open the alias detail page.
- Delete an alias. Choose an alias and then choose **Delete**.

Alias details

The alias details page displays information about the alias.

From this page you can:

- Edit an alias. Choose **Edit**.
- View the fleets you associated with the alias.
- Delete an alias. Choose **Delete**.

Alias detail information includes:

- **ID** – The unique number used to identify the alias.
- **Description** – The description of the alias.
- **ARN** – The Amazon Resource Name of the alias.
- **Creation** – The date and time the alias was created.
- **Last updated** – The date and time that the alias was last updated.
- **Routing type** – The routing type for the alias, which can be one of these:
 - **Simple** – Routes player traffic to a specified fleet ID. You can update the fleet ID for an alias at any time.
 - **Terminal** – Passes a message back to the client. For example, you can direct players who are using an out-of-date client to a location where they can get an upgrade.
- **Tags** – Key and value pairs used to identify the alias.

AWS SDK

Use the following AWS CLI commands to retrieve information about this resource:

- [ListAliases](#)
- [DescribeAlias](#)

Edit an alias

You can edit an alias using the Amazon GameLift Servers console or with the AWS CLI command [update-alias](#).

This topic describes how to edit an Amazon GameLift Servers alias for use with game session placement. You can make the following edits:

To edit an alias

Use either the Amazon GameLift Servers console or the AWS Command Line Interface (AWS CLI) to edit an alias.

Console

In the [Amazon GameLift Servers console](#), use the navigation pane to open the **Aliases** page.

1. Select the alias you want to edit and choose **Edit**. If you don't see the alias you want to edit, check your currently selected AWS Region.
2. On the **Edit alias** page, you can make the following edits:
 - Change the alias name.
 - Change the alias description.
 - Change the routing strategy from simple to terminal, or from terminal to simple.
 - For an alias with a simple routing strategy, change the fleet ID the alias is associated with.
 - For an alias with a terminal routing strategy, change the message text.
3. Choose **Save changes**. When updating the fleet ID for an alias with a simple routing strategy, it may take up to 2 minutes for the transition to complete. During this time, new game session placements might take place on the old fleet.

AWS CLI

Use the [update-alias](#) command to make changes to an alias resource. You can update an alias resource in your current default AWS Region, or you can add a `--region` tag to specify a different AWS Region.

You can change the following properties:

- Alias name.
- Alias description.
- Routing strategy type. Be sure to provide a fleet ID or a message string for the new routing strategy.
- Fleet ID for an existing simple routing strategy. The fleet ID must be in the same region as the alias.
- Message string for an existing terminal routing strategy.

Game session queues in the Amazon GameLift Servers console

The queue resource represents your configuration of the game session placement component. It determines how Amazon GameLift Servers searches for and chooses the best possible compute resource to host a new game session, based on your configuration choices.

View information about game session queues in the Amazon GameLift Servers console or using the or AWS SDK for Amazon GameLift Servers.

Console

You can view information on all queues, which are used to process requests for new game sessions. The queues page shows game session queues in the currently selected AWS Region. From the **Queues** page, you can create a new queue, delete existing queues, or open a details page for a selected queue. Each queue details page contains the queue's configuration and metrics data. For more information about queues, see [Configure game session placement](#).

The queues page displays the following summary information for each queue. You can adjust the table content as needed using the **Preferences** tool (see the



icon in the upper right corner of the table). Custom preferences are saved to your AWS account user and are automatically applied whenever you view this page.

- **Queue name** – The name assigned to the queue. Requests for new game sessions specify a queue by this name.
- **Queue timeout** – Maximum length of time, in seconds, that a game session placement request remains in the queue before timing out.
- **Destinations in queue** – Number of fleets listed in the queue configuration. Amazon GameLift Servers places new game sessions on any fleet in the queue.

View queue details

You can access detailed information on any queue, including the queue configuration and metrics. To open a queue details page, go to the **Queues** page and choose a queue name.

The queue detail page displays a summary table and tabs containing additional information. On this page you can do the following:

- Update the queue's configuration, list of destinations and player latency policies. Choose **Edit**.
- Delete a queue. After you delete a queue, all requests for new game sessions that reference that queue name will fail. Choose **Delete**.

Note

To restore a deleted queue, create a new queue with the deleted queue's name.

Details

The **Overview** section displays the queue's Amazon Resource Name (**ARN**) and the **Timeout**. You can use the ARN when referencing the queue in other actions or areas of Amazon GameLift Servers. The timeout is the maximum length of time, in seconds, that a game session placement request remains in the queue before timing out.

Event notification

The **Event notification** section lists the **SNS topic** Amazon GameLift Servers publishes event notifications to and the **Event data** that is added to all events created by this queue.

Tags

The **Tags** table displays the keys and values used to tag the resource. For more information about tagging, see [Tagging AWS resources](#).

Metrics

The **Metrics** tab shows a graphical representation of queue metrics over time.

Queue metrics include a range of information describing placement activity across the queue, including successful placements organized by Region. You can use Region data to understand where you are hosting your games. Regional placement metrics can help to detect issues with the overall queue design.

Queue metrics are also available in Amazon CloudWatch. For descriptions of available metrics, see [Amazon GameLift Servers metrics for queues](#).

Destinations

The **Destinations** tab shows all fleets or aliases listed for the queue.

When Amazon GameLift Servers searches the destinations for available resources to host a new game session, it searches the default order listed here. As long as there is capacity on the first destination listed, Amazon GameLift Servers places new game sessions there. You can have individual game session placement requests override the default order by providing player latency data. This data tells Amazon GameLift Servers to search for an available destination with the lowest average player latency. For more information about designing your queues, see [Customize a game session queue](#).

Session placement

Configuration settings that customize the placement process for your game.

Player latency policies

The **Player latency policies** section shows all policies that the queue uses. The tables lists the policies in the order they're enforced.

Locations

The **Locations** section shows the locations that this queue can put a game session in.

Priority

The **Priority** section shows the order that the queue evaluates a game sessions details.

Location order

The **Location order** section shows the default order that the queue uses when placing game sessions. The queue uses this order if you haven't defined other types of priority.

AWS SDK

Use the following AWS CLI commands to retrieve information about this resource:

- [DescribeGameSessionQueues](#)

Prepare for launch with Amazon GameLift Servers hosting

Use the following checklists to validate each deployment phase of your game. Items marked **[Critical]** are critical for your production launch.

Download and complete the Amazon GameLift Servers launch questionnaire, which is available in the [Amazon GameLift Servers console](#). We want every game developer using Amazon GameLift Servers to have a smooth launch day, and the requested information helps us help you prepare for upcoming load testing, soft launch or public launch. Plan to submit the completed questionnaire at least three (3) months before conducting your first load test.

Topics

- [Get your game ready](#)
- [Prepare for testing](#)
- [Prepare for launch](#)
- [Plan for post-launch updates](#)

Get your game ready

- **[Critical]** Verify that you've completed all the [development roadmap steps](#) for your hosting solution, and that you have all the required components in place, including and integrated game server, a backend service for game clients, hosting fleets, and a game session placement method (such as a queue).
- **[Critical]** [Create AWS Identity and Access Management \(IAM\) roles](#) that allow your game server to access other AWS resources while running.
- **[Critical]** Design and implement failover to other hosting resources as needed.

- [Plan the rollout of fleets to your target locations](#), considering your game's queue and fleet structure.
- [Automate your deployment](#) using infrastructure as code (IaC) with CloudFormation and the AWS Cloud Development Kit (AWS CDK).
- [Collect logs and analytics](#) using Amazon CloudWatch and Amazon Simple Storage Service (Amazon S3).

Prepare for testing

- **[Critical]** [Request increases for Amazon GameLift Servers service quotas](#) and other AWS service quotas so that your live environment can scale up to production needs.
- **[Critical]** Verify that the open ports on live fleets match the range of ports that your servers could use.
- **[Critical]** Close RDP port 3389 and SSH port 22.
- Develop a plan for the DevOps management of your game. If you're using Amazon CloudWatch Logs or Amazon CloudWatch custom metrics, define alarms for severe or critical problems on the server fleet. Simulate failures and test the runbooks.
- Verify that the compute resources you're using can support the number of server processes that you want to run concurrently on each compute.
- [Tune your scaling policy](#) to be more conservative at first and provide more idle capacity than you think you need. You can optimize for cost later. Consider the use of target-based scaling policy with 20 percent idle capacity.
- For FlexMatch, use [latency rules](#) to match players who are geographically near each other. Test how this behaves under load with synthetic latency data from your load test client.
- Load test your player authentication and game session infrastructure to see if it scales effectively to meet demand.
- Verify that a server left running for several days can still accept connections.
- Raise your Support plan level to Business or Enterprise so that AWS can respond to you during problems or outages.

Prepare for launch

- **[Critical]** [Set the fleet protection policy](#) to full protection on all live fleets so that scaling down doesn't stop active game sessions.

- **[Critical]** [Set fleet maximum sizes](#) high enough to accommodate peak anticipated demand, at minimum. We recommend that you double your maximum size for unanticipated demand.
- Encourage your entire development team to participate in the launch event and monitor your game launch in a launch room.
- Monitor player latency and player experience.

Plan for post-launch updates

- [Tune scaling policy](#) to minimize idle capacity based on player usage.
- [Modify FlexMatch rules](#) or [add hosting locations](#) based on player latency data and revised requirements.
- Optimize the runtime configuration to run as many games sessions as possible on each computing resource. Maximizing performance efficiency in this way can directly affect your fleet costs, because you might be able to run more server processes with the same compute resources.
- [Use your analytics data](#) to drive continued development, improve player experience and game longevity, and optimize monetization.

Manage Amazon GameLift Servers hosting resources using CloudFormation

You can use CloudFormation to manage your Amazon GameLift Servers resources. In CloudFormation, you create a template that models each resource and then use the template to create your resources. To update resources, you make the changes to your template and use CloudFormation to implement the updates. You can organize your resources into logical groups, called stacks and stack sets.

Using CloudFormation to maintain your Amazon GameLift Servers hosting resources offers a more efficient way to manage sets of AWS resources. You can use version control to track template changes over time and coordinate updates made by multiple team members. You can also reuse templates. For example, when deploying a game across multiple Regions, you might use the same template to create identical resources in each Region. You can also use these templates to deploy the same sets of resources in another partition.

For more information about CloudFormation, see the [AWS CloudFormation User Guide](#). To view template information for Amazon GameLift Servers resources, see the [Amazon GameLift Servers resource type reference](#).

Best practices

For detailed guidance on using CloudFormation, see the [CloudFormation best practices](#) in the *AWS CloudFormation User Guide*. In addition, these best practices have special relevance with Amazon GameLift Servers.

- **Consistently manage your resources through CloudFormation.** If you change your resources outside of CloudFormation your resources will get out of sync with your resource templates.
- **Use CloudFormation stacks and stack sets to efficiently manage multiple resources.**
 - Use stacks to manage groups of connected resources. For example, a stack that contains a build, a fleet that references the build, and an alias that references the fleet. If you update your template to replace a build, CloudFormation replaces the fleets connected to the build. CloudFormation then updates the existing aliases to point to the new fleets. For more information, see [Working with stacks](#) in the *AWS CloudFormation User Guide*.
 - Use CloudFormation stack sets if you're deploying identical stacks across multiple regions or AWS accounts. For more information, see [Working with stack sets](#) in the *AWS CloudFormation User Guide*.
- **If you are using Spot Instances, include an On-Demand Fleet as a back-up.** We recommend setting up your templates with two fleets in each region, one fleet with Spot Instances, and one fleet with On-Demand Instances.
- **Group your location-specific resources and global resources into separate stacks when you are managing resources in multiple locations.**
- **Place your global resources close to the services that use it.** Resources like queues and matchmaking configurations tend to receive a high volume of requests from specific sources. By placing your resources close to the source of those requests, you minimize the request travel time and can improve overall performance.
- **Place your matchmaking configuration in the same Region as the game session queue that it uses.**
- **Create a separate alias for each fleet in the stack.**

Using CloudFormation stacks

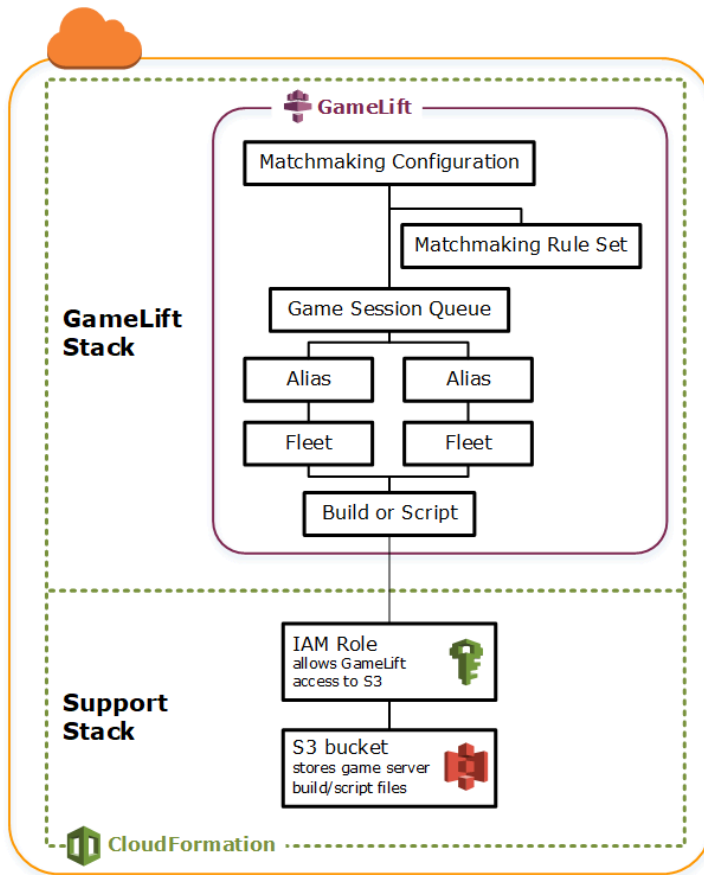
We recommend the following structures to use when setting up CloudFormation stacks for Amazon GameLift Servers resources. Your optimal stack structure varies depending on if you are deploying your game in one location or multiple locations.

Stacks for a single location

To manage Amazon GameLift Servers resources in a single location, we recommend a two-stack structure:

- **Support stack** – This stack contains resources that your Amazon GameLift Servers resources depend on. At a minimum, this stack should include the S3 bucket where you store your custom game server or Realtime script files. The stack should also include an IAM role that gives Amazon GameLift Servers permission to retrieve your files from the S3 bucket when creating an Amazon GameLift Servers build or script resource. This stack might also contain other AWS resources that are used with your game, such as DynamoDB tables, Amazon Redshift clusters, and Lambda functions.
- **Amazon GameLift Servers stack** – This stack contains all of your Amazon GameLift Servers resources, including the build or script, a set of fleets, aliases, and game session queue. CloudFormation creates a build or script resource with files stored in the S3 bucket location and deploys the build or script to one or more fleet resources. Each fleet should have a corresponding alias. The game session queue references some or all of the fleet aliases. If you are using FlexMatch for matchmaking, this stack also contains a matchmaking configuration and rule set.

The diagram below illustrates a two-stack structure for deploying resources in a single AWS Region.



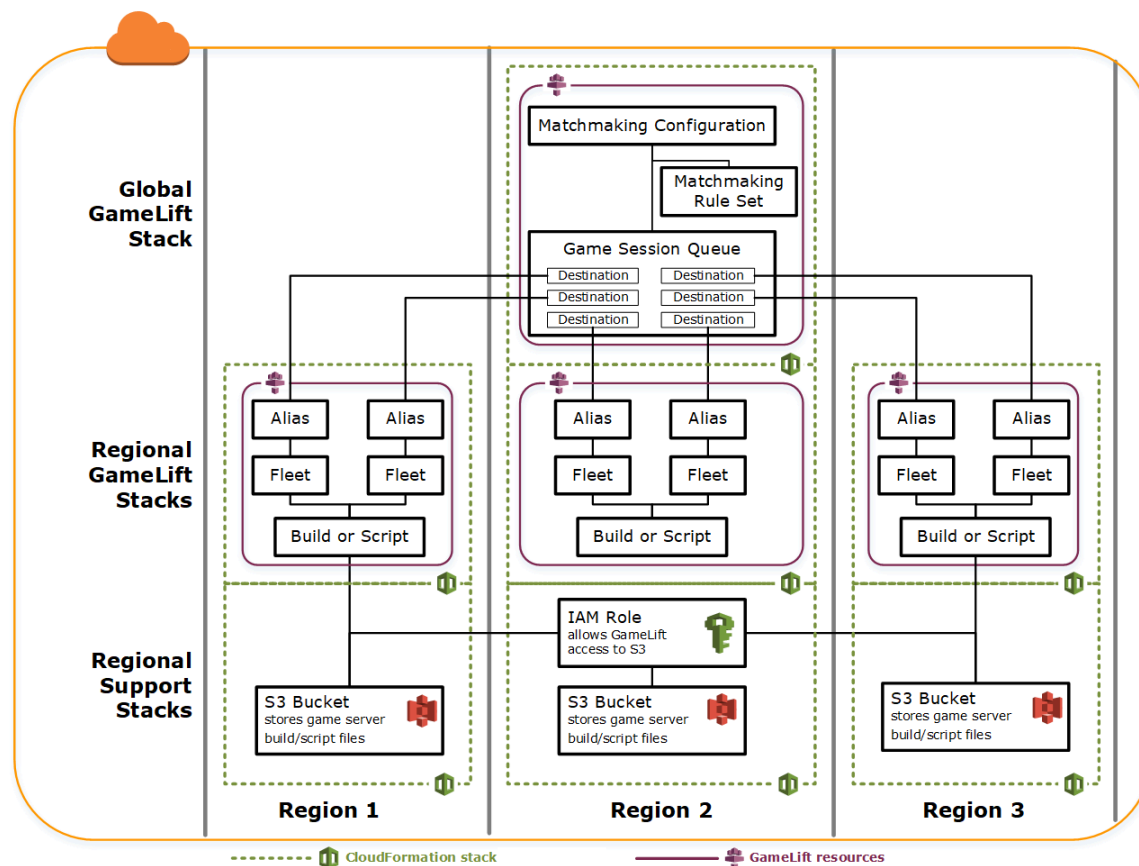
Stacks for multiple regions

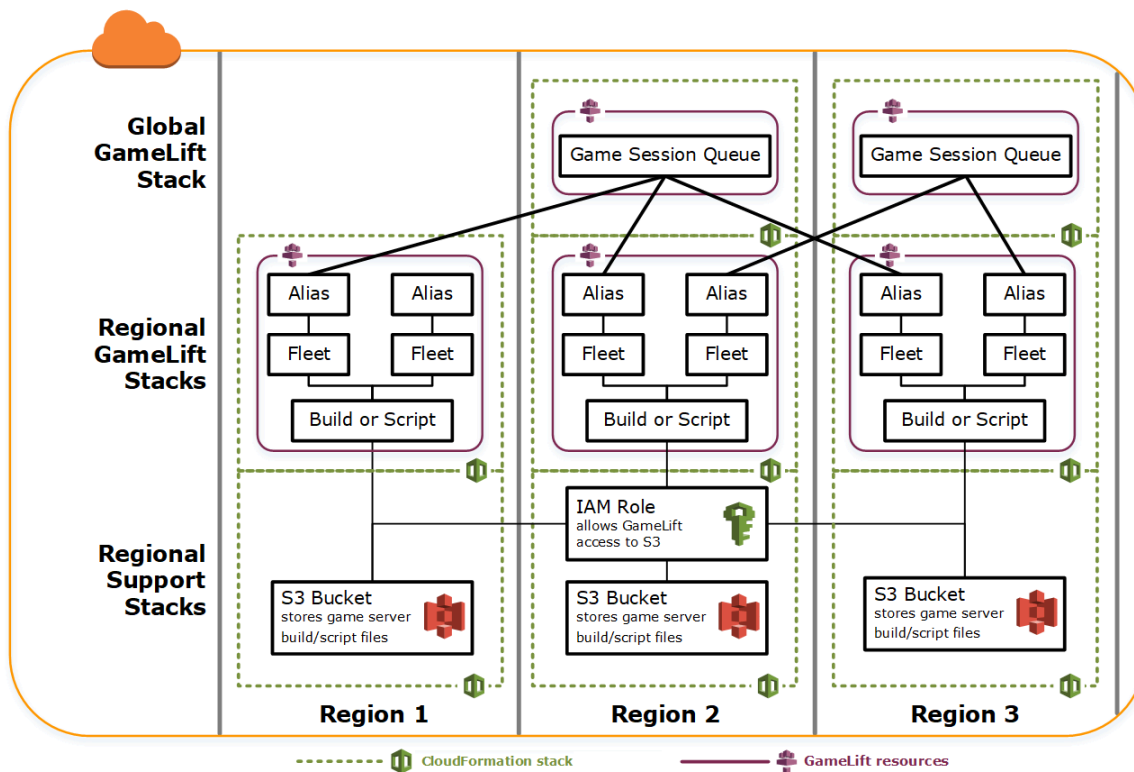
When deploying your game in more than one Region, keep in mind how resources can interact across Regions. Some resources, such as Amazon GameLift Servers fleets, can only reference other resources in the same Region. Other resources, such as an Amazon GameLift Servers queue, are Region agnostic. To manage Amazon GameLift Servers resources in multiple Regions, we recommend the following structure.

- Regional support stacks** – These stacks contain resources that your Amazon GameLift Servers resources depend on. This stack must include the S3 bucket where you store your custom game server or Realtime script files. It might also contain other AWS resources for your game, such as DynamoDB tables, Amazon Redshift clusters, and Lambda functions. Many of these resources are Region specific, so you must create them in every Region. Amazon GameLift Servers also needs an IAM role that allows access to these support resources. Because an IAM role is Region agnostic, you only need one role resource, placed in any Region and referenced in all of the other support stacks.

- **Regional Amazon GameLift Servers stacks** – This stack contains the Amazon GameLift Servers resources that must exist in each region where your game is being deployed, including the build or script, a set of fleets, and aliases. CloudFormation creates a build or script resource with files in an S3 bucket location, and deploys the build or script to one or more fleet resources. Each fleet should have a corresponding alias. The game session queue references some or all of the fleet aliases. You can maintain one template to describe this type of stack and use it to create identical sets of resources in every Region.
- **Global Amazon GameLift Servers stack** – This stack contains your game session queue and matchmaking resources. These resources can be located in any Region and are usually placed in the same Region. The queue can reference fleets or aliases that are located in any Region. To place additional queues in different Regions, create additional global stacks.

The diagrams below illustrates a multistack structure for deploying resources in several AWS Regions. The first diagram shows a structure for a single game session queue. The second diagram shows a structure with multiple queues.





Updating builds

Amazon GameLift Servers builds are immutable, as is the relationship between a build and a fleet. As a result, when you update your hosting resources to use a new set of game build files, the following need to happen:

- Create a new build using the new set of files (replacement).
- Create a new set of fleets to deploy the new game build (replacement).
- Redirect aliases to point to the new fleets (update with no interruption).

For more information, see [Update behaviors of stack resources](#) in the *AWS CloudFormation User Guide*.

Deploy build updates automatically

When updating a stack containing related build, fleet and alias resources, the default CloudFormation behavior is to automatically perform these steps in sequence. You trigger this update by first uploading the new build files to a new S3 location. Then you modify your CloudFormation build template to point to the new S3 location. When you update your stack with the new S3 location, this triggers the following CloudFormation sequence:

1. Retrieves the new files from S3, validates the files, and creates a new Amazon GameLift Servers build.
2. Updates the build reference in the fleet template, which triggers new fleet creation.
3. After the new fleets are active, updates the fleet reference in the alias, which triggers the alias to update to target the new fleets.
4. Deletes the old fleet.
5. Deletes the old build.

If your game session queue uses fleet aliases, player traffic is automatically switched to the new fleets as soon as the aliases are updated. The old fleets are gradually drained of players as game sessions end. Auto scaling handles the task of adding and removing instances from each set of fleets as player traffic fluctuates. Alternatively, you can specify an initial desired instance count to quickly ramp up for the switch and enable auto scaling later.

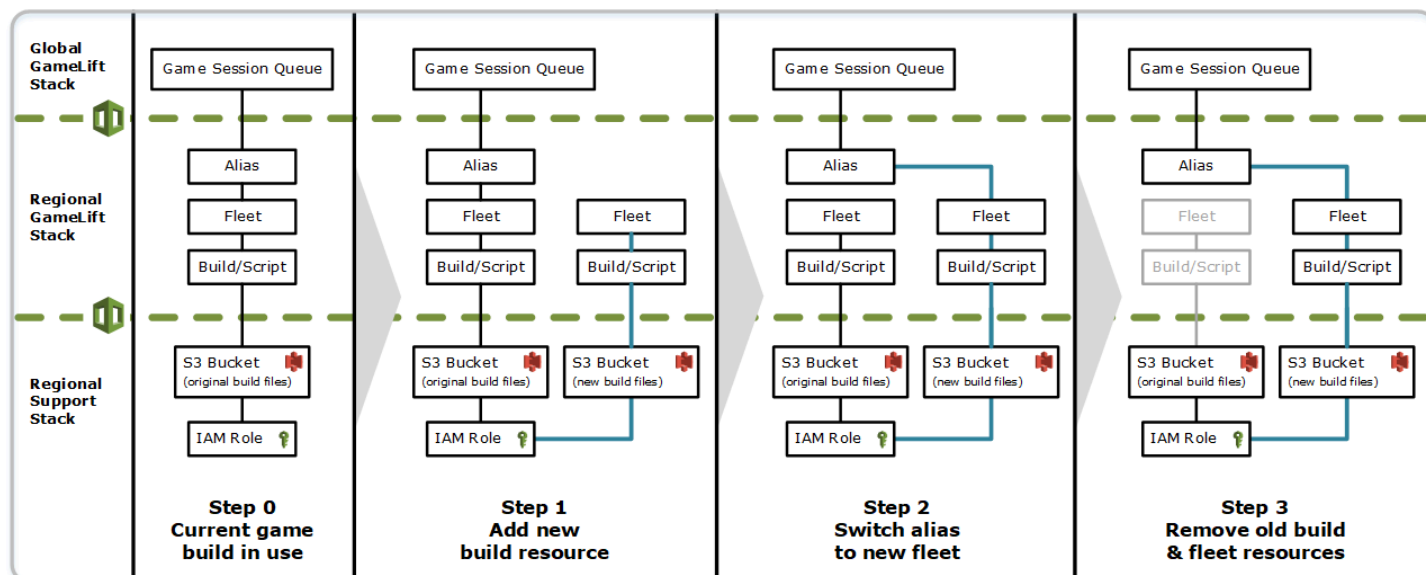
You can also have CloudFormation retain resources instead of deleting them. For more information, see [RetainResources](#) in the *AWS CloudFormation API Reference*.

Deploy build updates manually

If you want to have more control over when new fleets go live for players, you have some options. You can choose to manage aliases manually using the Amazon GameLift Servers console or the CLI. Alternatively, instead of updating your build template to replace the build and fleets, you can add a second set of build and fleet definitions to your template. When you update the template, CloudFormation creates a second build resource and corresponding fleets. Since the existing resources are not replaced, they are not deleted, and the aliases remain pointing at original fleets.

The main advantage with this approach is that it gives you the flexibility. You can create separate resources for the new version of your build, test the new resources, and control when the new fleets go live to players. A potential drawback is that it requires twice as many resources in each Region for a brief period of time.

The following diagram illustrates this process.



How rollbacks work

When executing a resource update, if any step is not completed successfully, CloudFormation automatically initiates a rollback. This process reverses each step in sequence, deleting the newly created resources.

If you need to manually trigger a rollback, change the build template's S3 location key back to the original location and update your stack. A new Amazon GameLift Servers build and fleet are created, and the alias switches over to the new fleet after the fleet is active. If you are managing aliases separately, you need to switch them to point to the new fleets.

For more information about how to handle a rollback that fails or gets stuck, see [Continue rolling back an update](#) in the *AWS CloudFormation User Guide*.

DDoS protection with Amazon GameLift Servers

Amazon GameLift Servers player gateway is a relay-based network that routes UDP traffic between game clients and game servers hosted on Amazon GameLift Servers. Player gateway provides proactive DDoS protection by validating traffic before it reaches game servers, rate limiting player traffic, hiding game server IP addresses from the public, and providing updated endpoints when relay endpoints become unhealthy. This feature helps protect your game servers from volumetric attacks with negligible added latency.

Player gateway requires game servers to run on Linux-based Amazon GameLift Servers Managed EC2 fleets or Container Fleets. You must enable player gateway during fleet creation and update your game client and game backend to use this feature.

Key benefits

Player gateway provides the following security and operational benefits:

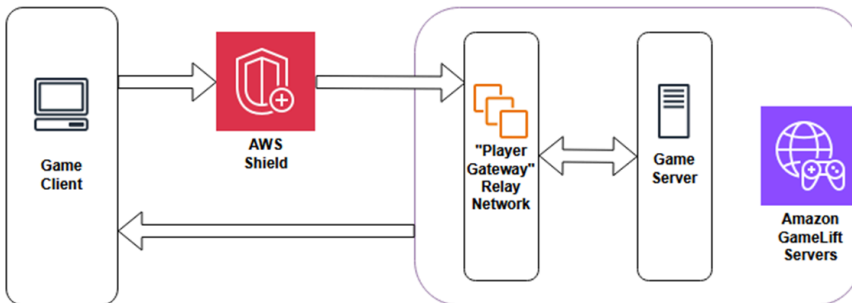
- **Hide game server IP addresses** – Game clients connect through relay endpoints instead of directly to game servers, hiding your game server addresses from the public.
- **Validate traffic** – All traffic through player gateway requires a player gateway token, allowing only traffic from authenticated players to reach your game servers.
- **Dynamic endpoint replacement** – When relay endpoints become unhealthy, Amazon GameLift Servers quickly replaces the endpoints and provides updated healthy endpoints on the next call to refresh player connection details.
- **Distribute player traffic** – Relay endpoints vary across players, reducing the impact of a single unhealthy relay endpoint to other players within the same game session.

How player gateway works

Player gateway uses a relay network to route UDP traffic between game clients and game servers. This protects against DDoS attacks by validating traffic before it reaches game servers, rate limiting player traffic, hiding game server IP addresses from the public, and providing updated endpoints when relay endpoints become unhealthy.

Traffic flow

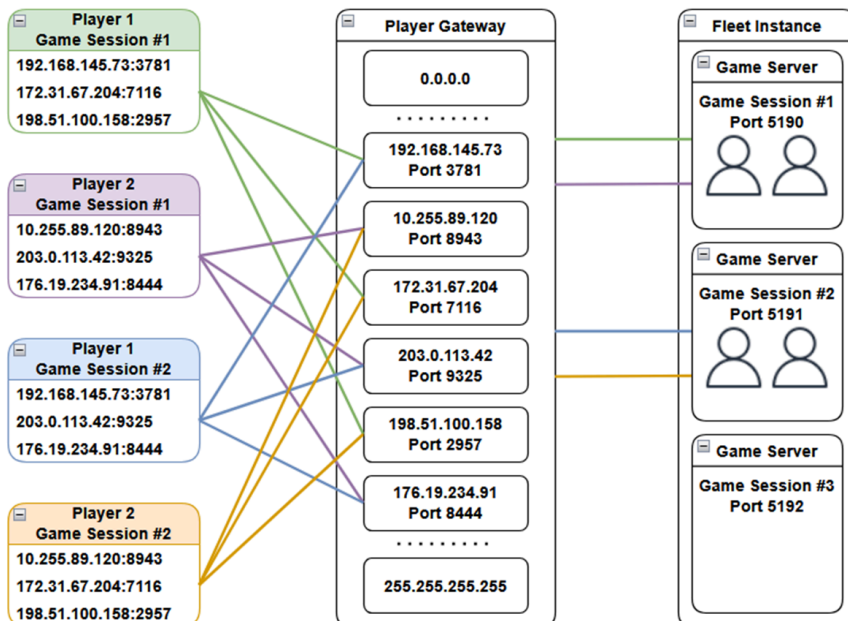
When a player connects to a game session, your game backend retrieves relay endpoints and a player gateway token from the `GetPlayerConnectionDetails` API and sends them to the game client. The game client prepends the player gateway token to UDP packets and sends the packets to a relay endpoint. The relay network validates the token and routes legitimate traffic to the game server. Before delivery, the relay network strips the player gateway token so that game servers receive the raw game client payload and should not require code changes. Communication from the game server back to the game client returns through the relay network without modification.



Core concepts

Relay endpoints

Relay endpoints are IP address and port combinations that game clients use to send traffic through player gateway. Each player receives multiple endpoints that vary across players to distribute traffic and reduce the impact of attacks on other players in the same game session.



Player gateway tokens

Player gateway tokens are encrypted byte arrays that authorize a player to send traffic to a game session through player gateway. The `GetPlayerConnectionDetails` API returns tokens as base64-encoded strings. Game clients must prepend the player gateway token to every UDP packet. The relay network validates tokens and rejects packets with invalid or missing tokens.

Important

Player gateway tokens must not be encrypted and must appear at the beginning of each UDP packet sent by the game client. If your game encrypts payloads, prepend the unmodified player gateway token after encrypting the game data.

Player gateway tokens remain valid for at least 3 minutes. We recommend refreshing tokens every 60 seconds to ensure players receive updated endpoints when relay endpoints become unhealthy.

GetPlayerConnectionDetails API

Your game backend calls the `GetPlayerConnectionDetails` API to retrieve connection details for players in a game session. The API returns either relay endpoints and player gateway tokens, or falls back to the game server's IP address and port for direct connection. Your game client should be designed to handle both connection types. To receive updated endpoints when relay endpoints become unhealthy, call this API periodically throughout the game session (recommended every 60 seconds).

For more information, see [GetPlayerConnectionDetails API](#).

Monitoring player gateway

Player gateway publishes metrics to Amazon CloudWatch to help you monitor network traffic patterns, identify potential DDoS attacks, and track relay performance. Metrics include packets and bytes in/out, throttled traffic, and player sessions. For the complete list of player gateway metrics, see [DDoS protection \(player gateway\) metrics](#).

IPv4 and IPv6 compatibility

Game clients communicate using IPv4. Player gateway uses IPv6 to communicate with game servers. Amazon GameLift Servers automatically handles the translation between IPv4 and IPv6 based on your fleet configuration.

For more information on configuring player gateway on your fleet, see [CreateFleet](#) or [CreateContainerFleet](#). For more information about IPv4 and IPv6 support, see [IPv4 and IPv6 compatibility](#).

Enable player gateway on fleets

Enable player gateway when creating a fleet to route game traffic through relay endpoints that validate and rate limit incoming UDP traffic to your game servers.

Prerequisites

Before enabling player gateway, verify the following requirements:

- **Supported locations** – For a list of AWS locations that support player gateway, see [Amazon GameLift Servers service locations](#).
- **Fleet type** – Player gateway supports Linux-based Managed EC2 fleets and Managed Container fleets.
- **Server SDK** – Player gateway requires Amazon GameLift Servers Server SDK 5.0 or later.

Player gateway modes

Configure player gateway compatibility for your fleet using the `PlayerGatewayMode` property:

DISABLED (default)

Fleet is not compatible with player gateway.

ENABLED

Fleet is compatible with player gateway in locations where it is supported. Fleet locations that do not support the feature can still be added to the fleet but will not utilize player gateway, directly utilizing the game server IP address.

REQUIRED

Fleet is compatible with player gateway. Fleet locations that do not support the feature cannot be added to the fleet.

IPv4 and IPv6 compatibility

Game clients must communicate with player gateway via IPv4. The player gateway relay network communicates with game servers using IPv6. Managed Container fleets automatically translate the IPv6 traffic to IPv4. For Managed EC2 fleets, configure how your fleet handles IPv6 traffic using the `GameServerIpProtocolSupported` property in `PlayerGatewayConfiguration`:

IPv4 (default)

Your game server only accepts incoming IPv4 traffic. Lightweight IP translation software will be installed and executed on your instances to receive and transform the incoming IPv6 traffic to IPv4.

DUAL_STACK

Your game server is compatible with incoming IPv6 traffic. IP translation software will not be installed or executed on your instances. Game servers that support IPv6 natively may have slightly higher performance.

Integrate player gateway into a game

After creating a player gateway-enabled fleet, you must update your game client and game backend to integrate with player gateway. Your game backend retrieves relay endpoints and player gateway tokens, then provides them to game clients. Game clients send all UDP traffic to these relay endpoints.

The integration follows this call path:

1. Your game backend calls the `GetPlayerConnectionDetails` API to retrieve relay endpoints and player gateway tokens for each player in a game session.
2. Your game backend sends the relay endpoints and player gateway tokens to the game client.
3. Your game client prepends the player gateway token to all outgoing UDP packets and sends the packets to the relay endpoints.
4. The relay network validates the player gateway token and routes the traffic to your game server.
5. Your game server sends traffic back to your game client through the same relay paths.

Backend integration

Your game backend must call the `GetPlayerConnectionDetails` API to retrieve relay endpoints and player gateway tokens for players. The backend then provides this information to game clients.

GetPlayerConnectionDetails API

The `GetPlayerConnectionDetails` API returns connection details for players in a game session:

Relay endpoints and tokens

When player gateway is enabled and supported in the location, returns:

- **Relay endpoints** – Multiple relay endpoints (IP address and port) that vary across players
- **Player gateway token** – Token that the client must prepend to all UDP packets (valid for at least 3 minutes)
- **Expiration** – Player gateway token expiration timestamp

Direct connection

When player gateway is not enabled or not supported in the location, returns the game server's IP address and port

Your game client should be designed to handle both connection types.

Example API call:

```
// C++ example using AWS SDK
Aws::GameLift::GameLiftClient client;
Aws::GameLift::Model::GetPlayerConnectionDetailsRequest request;
request.SetGameSessionId(gameSessionId);
request.SetPlayerIds(playerIds); // Vector of player IDs

auto outcome = client.GetPlayerConnectionDetails(request);
if (outcome.IsSuccess()) {
    auto result = outcome.GetResult();
    auto connectionDetails = result.GetPlayerConnectionDetails();

    // Process each player's connection details
    for (const auto& detail : connectionDetails) {
        std::string playerId = detail.GetPlayerId();
```

```
// Get relay endpoints (IP address and port)
auto endpoints = detail.GetEndpoints();
for (const auto& endpoint : endpoints) {
    std::string ipAddress = endpoint.GetIpAddress();
    int port = endpoint.GetPort();
}

// Get player gateway token
auto token = detail.GetPlayerGatewayToken();

// Get expiration time
auto expiration = detail.GetExpiration();

// Send endpoints and token to game client
}
}
```

See [GetPlayerConnectionDetails](#) in the *Amazon GameLift Servers API Reference*.

Refresh relay endpoints

Call `GetPlayerConnectionDetails` periodically to provide players with updated endpoints when relay endpoints become unhealthy. We recommend calling the API every 60 seconds for all players in a game session. To optimize API usage, batch multiple player IDs in a single API call.

Important

Regular refresh calls are the primary mechanism for delivering updated endpoints to players. While player gateway tokens remain valid for at least 3 minutes, refreshing every 60 seconds ensures players receive updated endpoints regularly.

Client integration

Your game client must prepend player gateway tokens to all outgoing UDP packets and send the packets to the provided relay endpoints. The Amazon GameLift Servers client SDK for C++, C#, Unreal Engine, and Unity provides utilities to simplify this integration.

Client requirements

To route traffic through player gateway, your game client must:

- **Prepend player gateway tokens** – Prepend the player gateway token to all outgoing UDP packets. Player gateway tokens must not be encrypted and must appear at the beginning of each packet. Packets without a valid player gateway token will be dropped.
- **Send to relay endpoints** – Send all UDP packets to the provided relay endpoints.
- **Maintain connection activity** – Ensure that either the game client sends a packet to the game server, or the game server sends a packet to the player, at least once every 30 seconds. This activity maintains the connection through the relay network.
- **Handle endpoint updates** – Accept updated relay endpoints and player gateway tokens from your backend (recommended every 60 seconds) and transition to new endpoints without dropping the connection.

Client SDK

The Amazon GameLift Servers client SDK for C++, C#, Unreal Engine, and Unity provides utilities to simplify player gateway integration:

- **Token management** – Prepends player gateway tokens to all outgoing UDP packets.
- **Endpoint selection** – Routes traffic to relay endpoints using a configurable algorithm.
- **Endpoint refresh** – Schedules periodic callbacks to retrieve updated relay endpoints and player gateway tokens from your backend.

Endpoint selection algorithms

The client SDK provides two built-in algorithms for selecting which relay endpoint to use:

Fallback algorithm

Uses one endpoint until it becomes unhealthy, then switches to another endpoint. Best for menus, lobbies, and turn-based games where brief interruptions are acceptable. During endpoint failover, packets may be lost for the configured timeout period (default: 2 seconds).

Predictive rotation algorithm

Continuously rotates through all available endpoints and predicts failures before they occur. Best for real-time gameplay such as first-person shooters and racing games where consistent packet delivery is critical. Requires the game server to send messages at a consistent frequency.

You can also implement custom algorithms by extending the SDK's base algorithm class.

Client SDK resources

For complete integration instructions and sample code, see the following resources:

C++

A client SDK for C++ is available at [Amazon GameLift Servers client SDK for C++](#).

C#

A client SDK for C# is available at [Amazon GameLift Servers client SDK for C#](#).

Unreal Engine

A plugin for Unreal Engine is available at [Amazon GameLift Servers client SDK for Unreal Engine](#). Access requires membership in the Epic Games organization on GitHub. See [Unreal Engine on GitHub](#) for details.

Unity

A client SDK for Unity is available at [Amazon GameLift Servers client SDK for Unity](#).

Test your integration

Before deploying to a fleet, test your player gateway integration locally using the player gateway testing tool. This tool simulates the player gateway UDP proxy infrastructure and helps you verify that your game client correctly prepends player gateway tokens, routes traffic through multiple endpoints, and handles network degradation.

For setup and usage instructions, see the [player gateway testing app](#) on GitHub.

Best practices

Follow these best practices when integrating player gateway:

- **Refresh endpoints every 60 seconds** – Call `GetPlayerConnectionDetails` every 60 seconds to ensure players receive updated endpoints regularly.
- **Batch API calls** – When calling `GetPlayerConnectionDetails` for multiple players in the same game session, batch the calls together to reduce API overhead.

- **Maintain connection activity** – Ensure that traffic flows between the game client and game server at least once every 30 seconds. This can be either client-to-server or server-to-client traffic.
- **Handle endpoint updates gracefully** – When your game client receives updated endpoints from the backend, transition to the new endpoints without dropping the player's connection.
- **Choose the right algorithm** – Select the endpoint selection algorithm that best matches your game's requirements. Use the fallback algorithm for turn-based games and the predictive rotation algorithm for real-time games.

Testing and troubleshooting Amazon GameLift Servers

This section provides guidance for diagnosing and resolving issues with your Amazon GameLift hosting solution. Whether you're experiencing problems during development, deployment, or production operations, these resources will help you identify root causes and implement effective solutions.

- **Development and testing environments** - Set up iterative development workflows that mirror production conditions, enabling you to catch and resolve integration issues early in your development cycle.
- **Fleet deployment and management** - Diagnose common fleet creation failures, server process activation issues, scaling problems, and deployment errors that can prevent your game servers from running properly.
- **Runtime diagnostics and monitoring** - Access live fleet instances for real-time troubleshooting, performance analysis, and configuration validation to resolve issues affecting player experience.

Use these resources alongside Amazon GameLift's monitoring and logging capabilities to maintain optimal performance and availability for your multiplayer games.

Set up for iterative development with Amazon GameLift Servers Anywhere

Amazon GameLift Servers provides tools and solutions to help you set up a hosted test environment for use during game development. With these tools, you can create an environment that mirrors the real-world player experience of managed hosting with Amazon GameLift Servers and supports a rapid, iterative development process.

With a separate test environment, you remove the overhead of an Amazon GameLift Servers managed fleet during testing. You no longer have to upload each new game server build iteration, create a new fleet for it, and then wait 15+ minutes to it to activate. Instead, you can create a new build, quickly update the test fleet with the new build, start it, and commence testing.

Using an Amazon GameLift Servers Anywhere fleet, you can set up a test environment using a local device, such as your development workstation. You can also set up a test environment using a cloud-based hosting resource.

Set up an Anywhere test environment to develop and test a range of scenarios, including these:

- Test your game server integration with the Amazon GameLift Servers server SDK. You can test even without a working game client by using AWS CLI calls to start new game sessions and track game session events.
- Test interactions between your game client, backend service, and the Amazon GameLift Servers service as you develop components for your game. Fine-tune the player experience for joining a game.
- Experiment with your FlexMatch matchmaker design. Try out rule set variations and other matchmaking feature implementations. Set up and test matchmaking backfill.
- Try out other Amazon GameLift Servers hosting features, such as runtime configuration settings (with the Amazon GameLift Servers Agent) for game server life cycle management.
- Quickly build, test, and repeat to validate all aspects of your game's player experience, including multiplayer interactions, in a live, hosted environment.

Later, as you prepare your game for launch, you'll want to add Amazon GameLift Servers managed fleets to fine-tune your hosting configurations and test additional scenarios, including the following:

- Experiment with and test game session queue designs, including use of multi-location fleets, Spot and On-Demand fleets, and multiple instance types.
- Try out game session placement options with managed fleets, including the use of optional latency policies and fleet prioritization settings.
- Configure capacity scaling to meet player demand, using automatic or manual scaling options.
- Set up AWS CloudFormation with Amazon GameLift Servers managed fleets to manage your hosting resources long term.

Fast Build Update Tool (for development only)

With managed EC2 fleets, to deploy a game server build update, you need to upload each new build to Amazon GameLift Servers and create a new fleet for it.

The Fast Build Update Tool lets you can bypass these steps during development, saving you time and allowing for faster development iteration. With this tool, you can quickly update your game build files across all computes in an existing fleet. The tool has several options; you can replace an entire game build or change 6 specific files, and you can manage how

to restart game server processes after the updates. You can also use it to update individual computes in a fleet.

To get the Fast Build Update Tool and learn more about how to use it, visit the Amazon GameLift Servers Toolkit repo for [The Fast Build Update Tool](#) in Github.

Topics

- [Build a cloud-based test environment](#)
- [Set up local testing with Amazon GameLift Servers Anywhere](#)
- [Test your integration using Amazon GameLift Servers Local](#)

Build a cloud-based test environment

Note

This topic covers iterative testing for games that are integrated with the server SDK for Amazon GameLift Servers version 5.x. If your game uses server SDK version 4.x or earlier, see [Test your integration using Amazon GameLift Servers Local](#).

Use an Amazon GameLift Servers Anywhere fleet to iteratively build and test your game components in a cloud-based hosted environment. Create an Anywhere fleet with hosting resources and a connection to the Amazon GameLift Servers service, run your game servers on them, and test game functionality as needed.

Deploy an Anywhere fleet with the Amazon GameLift Servers Agent

If your game server build is integrated with Amazon GameLift Servers SDK 5.x or later, you can deploy it to a cloud-based Anywhere fleet with the Amazon GameLift Servers Agent. The Agent is a background process that manages game server life cycles and other tasks on each compute in a fleet. These tasks include registering the compute with an Anywhere fleet, acquiring an authentication token, and starting/stopping game server processes based on a set of instructions. The Agent is controlled by a fleet's runtime configuration, which you can update at any time during the life of the fleet. (The Agent is automatically deployed to managed EC2 fleets.) For more information and to download the Agent, see the [Amazon GameLift Servers GitHub repository](#).

Set up iterative testing with Amazon EC2

Use the guided workflow in this [Amazon GameLift Servers toolkit solution](#) to set up a cloud-based hosting environment that mirrors the managed hosting experience with Amazon GameLift Servers.

The GitHub repository provides a set of scripts that automate most of the processes for setting up a test environment with Amazon GameLift Servers Anywhere and the Amazon GameLift Servers Agent. It also provides guidance for updating the environment whenever you have a new game server build to test. You can run a single script that deploys a test environment with a sample game server build, or you can walk through each step to set it up with your own game server build.

In this workflow, you'll work entirely in the AWS Management Console, using AWS CloudShell to run scripts and complete command-line tasks.

Note

For the tasks in this tutorial, you need an AWS account user with permissions for the following services: Amazon GameLift Servers, AWS CloudShell, Amazon S3, AWS Systems Manager, Amazon EC2, and AWS Identity and Access Management. Users with admin-level access to the AWS account already have the required permissions.

The workflow covers the following tasks:

- **Package a game server build for Amazon GameLift Servers.** The workflow provides a script to build a sample C++ game server, which has already been integrated with the server SDK for Amazon GameLift Servers version 5.x and is ready for hosting. Alternatively, you can work with your own game project if you've completed integration.
- **Set up an Amazon Simple Storage Service bucket to store game server builds and dependencies.** As you produce new versions of your game builds, you can store them in S3 and use the scripts to update the Anywhere fleet for game testing.
- **Get and build the Amazon GameLift Servers Agent.** The Agent manages game server processes on a hosting resource based on your configuration. It uses the same logic and behaves identically to Amazon GameLift Servers managed EC2 hosting.
- **Set up an Anywhere fleet for your hosting resources.** With an Anywhere fleet you can use the Amazon GameLift Servers service for hosting resources that aren't managed by Amazon GameLift Servers. In this step, you'll also configure the runtime configuration, which instructs Amazon GameLift Servers Agent when and how to start game server processes.

- **Set up an Amazon EC2 instance.** This is your test environment for iterative testing. It is much faster to use a standard EC2 instance instead of a fully managed Amazon GameLift Servers instance (which is optimized for production-level usage). With a standard EC2 instance, you can quickly and continually update the game server as needed.
- **Deploy your game server build and Amazon GameLift Servers Agent to the Amazon EC2 instance.** The workflow provides a script that gets the latest version of your game build and all dependencies and installs it on your EC2 instance. In this workflow, dependencies include the Amazon GameLift Servers Agent and the CloudWatch Agent.
- **Start the Amazon GameLift Servers Agent.** Once installed, the Agent automatically starts and begins executing instructions. These include:
 - Register the EC2 instance as a compute in the Amazon GameLift Servers Anywhere fleet.
 - Establish a WebSocket connection with the Amazon GameLift Servers service and get the latest runtime configuration.
 - Start up game server processes based on the instructions in the runtime configuration. In this workflow, the Agent is instructed to start a single process of the game server executable.
- **Test your game scenarios.** With the test environment set up and your latest game server build installed, you can commence testing. The workflow walks through several steps for testing including starting a game session. Access CloudWatch game server logs to track progress as the game session starts up and prepares to accept players.

As you develop your game components, including a game client and client-side backend service, you can include these in your test scenarios. Use a game client to request a game session, retrieve connection info from the Amazon GameLift Servers service, and then connect directly to the game session.

- **Deploy a new game server build and repeat tests.** As you develop your game, you can generate new game server builds, then quickly deploy them to the EC2 test environment for testing. Upload them to the Amazon S3 bucket and then use the workflow scripts to update the test environment.

Transition your game to Amazon GameLift Servers managed fleets

After you've completed development testing and you're ready to prepare for launch, this is a good time to switch over to Amazon GameLift Servers managed fleets. Use managed fleets to fine-tune and test your game hosting resources. Implement your game session placement solution (queues and matchmakers), select optimum hosting hardware (including Spot fleets) and locations, and

choose a strategy for scaling capacity. You might also want to start using AWS CloudFormation to more efficiently manage the life cycles of all your game hosting resources, including fleets, queues, and matchmakers.

It requires minimal effort to transition from a cloud-based Anywhere test fleet to an Amazon GameLift Servers managed fleet. You don't need to change any game code, and you can reuse the same queues and matchmakers. Do the following tasks:

- **Create an Amazon GameLift Servers build resource.** With an Anywhere test fleet, you have to manually deploy your game server build and dependencies to each fleet compute. With a managed fleet, upload your game build package to Amazon GameLift Servers, which automatically deploys it to all fleet computes. See [Create a game server build for Amazon GameLift Servers](#) for details on packaging your game build files and creating a build resource with files in an Amazon S3 bucket.
- **Create a managed fleet.** Create a fleet using the console or AWS CLI, specifying an EC2 managed fleet. This type of fleet requires additional configuration settings, including specifying the build resource and instance types. You can use the same runtime configuration to manage game server life cycle on each fleet compute. See [Create an Amazon GameLift Servers managed EC2 fleet](#) for details on creating a managed fleet.
- **Redirect fleet aliases (optional).** If you set up aliases to use with your Anywhere fleets, you can reuse the same aliases for your managed fleets. See [Create an Amazon GameLift Servers alias](#) for details on creating or updating an alias.

Set up local testing with Amazon GameLift Servers Anywhere

Note

This topic covers local testing for games that are integrated with the server SDK for Amazon GameLift Servers version 5.x. If your game uses server SDK version 4.x or earlier, see [Test your integration using Amazon GameLift Servers Local](#).

Use an Amazon GameLift Servers Anywhere fleet and your own hardware to iteratively build and test your game components in a simulated hosted environment. Set up an Anywhere fleet and register a local device to establish a connection to the Amazon GameLift Servers service. Install your game server build onto the device, start a game server process, and test game functionality as needed. You can update your game server build as often as needed to test each new build iteration.

With an Anywhere fleet, you can test using the AWS CLI or with test scripts. If you've integrated a game client with Amazon GameLift Servers, you can run the client on the same local device or on a different device.

Testing locally with an Anywhere fleet is particularly useful for testing your game server integration with Amazon GameLift Servers. You have full visibility into all hosting activity on the local machine, as well as events and logging data.

Note

Are you using the Amazon GameLift Servers plugin for Unreal Engine or Unity? These tools include guided workflows for setting up local testing with an Anywhere fleet. Follow the documentation for [Plugin for Unity: Set up local testing with Amazon GameLift Servers Anywhere](#) or [Plugin for Unreal: Host your game locally with Amazon GameLift Servers Anywhere](#).

Topics

- [Set up a local Anywhere fleet](#)
- [Update and install your game server](#)
- [Test game session activity](#)
- [Iterate on your game server](#)
- [Transition your game to Amazon GameLift Servers managed fleets](#)

Set up a local Anywhere fleet

Follow these steps to create an Anywhere fleet for your local workstation. For detailed instructions using either the AWS CLI or the AWS Management Console for Amazon GameLift Servers, see [Create an Amazon GameLift Servers Anywhere fleet](#).

To create the Anywhere fleet

1. **Create a custom location for your local workstation. (AWS CLI or console).** A custom location is simply a label for the compute resource you plan to include in your Anywhere fleet. Custom location names must start with `custom-`. For example: `custom-my_laptop`. See [Create a custom location](#).

2. **Create an Anywhere fleet (AWS CLI or console).** In this step, create the fleet resource with the custom location for your local workstation. See [Create an Anywhere fleet](#).

Make a note of the new fleet's ID or ARN value. You'll need this value for the next step.

3. **Register your local workstation as a fleet compute (AWS CLI only).** An Anywhere fleet must have at least one compute resource to host your game servers. See [Add a compute to the fleet](#). To add a compute to the fleet, you need the following information:

- A compute name. Each compute in a fleet must have a unique name.
- The Anywhere fleet identifier. You can use either the `FleetID` or `FleetArn`.
- The compute's connection info. Specify either an `IpAddress` or `DnsName`. This is how Amazon GameLift Servers and game clients will connect to game servers.
- A custom location in the Anywhere fleet.

Make a note of the `GameLiftServiceSdkEndpoint` return value. You'll need this value when you update your game server to run on an Anywhere fleet.

Update and install your game server

This task assumes that you've already integrated a game server build with Amazon GameLift Servers server SDK 5.x. The integration process involves adding code to your game server so that it can interact with the Amazon GameLift Servers service to start and manage game sessions.

For an Anywhere fleet, you need to manually configure certain game server settings. On an Amazon GameLift Servers managed fleet, these settings are configured automatically.

To prepare your game server for an Anywhere fleet

1. **Get an authentication token.** Your game server must include an authentication token with every communication with the Amazon GameLift Servers service. Amazon GameLift Servers auth tokens are short-lived and must be regularly refreshed.

As a best practice, create a script to complete the following tasks:

- Call the AWS CLI action `get-compute-auth-token`.
- Store the returned token value where game server processes can retrieve it, such as in an environment variable on the local compute.

Install the script with your game server on the compute. Set the script to run before starting the first game server process. While game server processes are active, run the script regularly to maintain a valid auth token. All game server processes on the compute can use the same auth token.

2. **Update your Amazon GameLift Servers game server code.** When you integrated your game server code with the server SDK for Amazon GameLift Servers, you added a call to the action `InitSdk()`. When the game server runs on an Anywhere fleet, this call requires additional server parameters. For more information, see [Initialize the server process](#) and the [Server SDK 5.x for Amazon GameLift Servers](#) for your development language. The server parameters are:

- `webSocketUrl` – Set this parameter to the `GameLiftServiceSdkEndpoint` value that is returned when you register a compute with the fleet.
- `hostId` – Set this parameter to the compute name that you specify when you register a compute with the Anywhere fleet.
- `fleetId` – Set this parameter to the ID of the Anywhere fleet.
- `authToken` – Set this parameter to the token that is returned in response to a request to retrieve an authentication token for a compute.
- `processId` – Set this parameter to identify a game server process that's running on the local compute. Each concurrent game server process must have a unique process ID.

The server parameter values that each game server process uses needs to be specific to the Anywhere fleet compute where the process is running. For details on how to get the appropriate values for a compute, see [Add a compute to the fleet](#). As a best practice, set `webSocketUrl`, `hostId`, `fleetId`, and `authToken` as environment variables on the local compute. All server processes that run on the compute will use these values.

3. Install the game server build on the local compute. Include all dependencies needed to run the game server.
4. Start one or more game server processes running on the local compute. When the game server process calls the server SDK action `ProcessReady()`, the process is ready to host a game session.

Test game session activity

Test your game server integration by working with game sessions. If you don't have a game client integrated with Amazon GameLift Servers functionality, you can use the AWS CLI to start game sessions. Try the following scenarios:

- **Create a game session.** Call [create-game-session](#) command (or the [CreateGameSession](#) API operation). Specify your Anywhere fleet's ID and custom location. This call returns a unique identifier for the new game session.
- **Check game session status.** Call [describe-game-sessions](#) command (or the [DescribeGameSessions](#) API action). Specify the game session ID. This call returns detailed game session information, including the game session status. Game sessions in Active status are ready for players to connect. To get a list of all game sessions for the fleet, call [list-game-sessions](#) command (or the [ListGameSessions](#) API action).
- **Connect to the game session.** If your game client has the ability to join a game session, use the connection information included in the game session information.

Iterate on your game server

You can use the same Anywhere fleet and compute to test other versions of your game server build.

1. **Clean up your existing GameSession.** If the game server process crashes or won't call `ProcessEnding()`, Amazon GameLift Servers cleans up the GameSession after the game server stops sending health checks.
2. **Generate a new game server build.** Make changes to your game server and package an revised build.
3. **Update the game server build on your local compute.** Your previous Anywhere fleet is still active and your laptop is still registered as a compute resource in the fleet.
4. **Get an updated authorization token.** Call the [get-compute-auth-token](#) CLI command and store the token on the local compute.
5. **Start one or more game server processes running on the local compute.** When the game server process calls `ProcessReady()`, it's ready to be used for testing.

Transition your game to Amazon GameLift Servers managed fleets

After you've completed development testing and you're ready to prepare for launch, this is a good time to switch over to Amazon GameLift Servers managed fleets. Use managed fleets to fine-tune and test your game hosting resources. Implement your game session placement solution (queues and matchmakers), select optimum hosting hardware (including Spot fleets) and locations, and choose a strategy for scaling capacity. You might also want to start using AWS CloudFormation to more efficiently manage the life cycles of all your game hosting resources, including fleets, queues, and matchmakers.

You need to make a few minor modifications to transition from a local Anywhere test fleet to an Amazon GameLift Servers managed fleet. You can reuse the same queues and matchmakers. Do the following tasks:

- **Change the game server code call to `InitSdk()`.** Remove the server parameters. For a managed fleet, Amazon GameLift Servers automatically tracks this information.
- **Create an Amazon GameLift Servers build resource.** With an Anywhere test fleet, you have to manually deploy your game server build and dependencies to each fleet compute. With a managed fleet, you create and upload your game build package to Amazon GameLift Servers, which automatically deploys it to all fleet computes. See [Create a game server build for Amazon GameLift Servers](#) for details on packaging your game build files and creating a build resource with files in an Amazon S3 bucket. Don't include scripts that register a compute and get an authentication token, as Amazon GameLift Servers automatically handles these tasks with managed fleets.
- **Create a managed fleet.** Create a fleet using the console or AWS CLI, specifying an EC2 managed fleet. This type of fleet requires additional configuration settings, including specifying the build resource and instance types. You also need to set up a runtime configuration to manage game server life cycle on each fleet compute. See [Create an Amazon GameLift Servers managed EC2 fleet](#) for details on creating a managed fleet.
- **Redirect fleet aliases (optional).** If you set up aliases to use with your Anywhere fleets, you can reuse the same aliases for your managed fleets. See [Create an Amazon GameLift Servers alias](#) for details on creating or updating an alias.

Test your integration using Amazon GameLift Servers Local

Note

This topic covers testing for games that are integrated with the server SDK for Amazon GameLift Servers version 3.x and 4.x only. Your server SDK package includes a compatible version of Amazon GameLift Servers Local. If you're using server SDK version 5.x, see [Local testing with GameLift Server SDK 5.x](#) for local testing with an Amazon GameLift Servers Anywhere fleet. For guidance on migrating from SDK 4.x to 5.x, see [Migrate to GameLift Server SDK 5.x](#).

Use Amazon GameLift Servers Local to run a limited version of the managed Amazon GameLift Servers service on a local device and test your game integration against it. This tool is useful when doing iterative development on your game integration. The alternative—uploading each new build to Amazon GameLift Servers and configuring a fleet to host your game—can take several or more each time.

With Amazon GameLift Servers Local, you can verify the following:

- Your game server is correctly integrated with the Server SDK and is properly communicating with the Amazon GameLift Servers service to start new game sessions, accept new players, and report health and status.
- Your game client is correctly integrated with the AWS SDK for Amazon GameLift Servers and is able to retrieve information on existing game sessions, start new game sessions, join players to games and connect to the game session.

Amazon GameLift Servers Local is a command-line tool that starts a self-contained version of the managed Amazon GameLift Servers service. Amazon GameLift Servers Local also provides a running event log of server process initialization, health checks, and API calls and responses. Amazon GameLift Servers Local recognizes a subset of the AWS SDK actions for Amazon GameLift Servers. You can make calls from the AWS CLI or from your game client. All API actions perform locally just as they do in the Amazon GameLift Servers web service.

Each server process should only host a single game session. The game session is the executable you use to connect to Amazon GameLift Servers Local. When the game session is completed, you should call `GameLiftServerSDK::ProcessEnding` and then exit the process. When testing

locally with Amazon GameLift Servers Local, you can start multiple server processes. Each process will connect to Amazon GameLift Servers Local. You can then create one game session for each server process. When your game session ends, your game server process should exit. You must then manually start another server process.

Amazon GameLift Servers local supports the following APIs:

- CreateGameSession
- CreatePlayerSession
- CreatePlayerSessions
- DescribeGameSessions
- DescribePlayerSessions

Set up Amazon GameLift Servers local

Amazon GameLift Servers Local is provided as an executable `.jar` file bundled with the [Server SDK](#). It can be run on Windows or Linux and used with any Amazon GameLift Servers-supported language. For SDK version 4.0.2, you can download it from the [official GitHub releases](#). The `GameLiftLocal.jar` file is included in the `GameLift-CSharp-ServerSDK-4.0.2.zip` artifact.

Before running Local, you must also have the following installed.

- A build of the server SDK for Amazon GameLift Servers version 3.1.5 to 4.x.
- Java 8

Test a game server

If you want to test your game server only, you can use the AWS CLI to simulate game client calls to the Amazon GameLift Servers Local service. This verifies that your game server is performing as expected with the following:

- The game server launches properly and initializes the server SDK for Amazon GameLift Servers.
- As part of the launch process, the game server notifies Amazon GameLift Servers that the server is ready to host game sessions.
- The game server sends health status to Amazon GameLift Servers every minute while running.
- The game server responds to requests to start a new game session.

1. Start Amazon GameLift Servers Local.

Open a command prompt window, navigate to the directory containing the file *GameLiftLocal.jar* and run it. By default, Local listens for requests from game clients on port 8080. To specify a different port number, use the `-p` parameter, as shown in the following example:

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

Once Local starts, you see logs indicating that two local servers were started, one listening for your game server and one listening for your game client or the AWS CLI. Logs continue to report activity on the two local servers, including communication to and from your game components.

2. Start your game server.

Start your Amazon GameLift Servers-integrated game server locally. You don't need to change the endpoint for the game server.

In the Local command prompt window, log messages indicate that your game server has connected to the Amazon GameLift Servers Local service. This means that your game server successfully initialized the server SDK for Amazon GameLift Servers (with `InitSDK()`). It has called `ProcessReady()` with the log paths shown and, if successful, is ready to host a game session. While the game server is running, Amazon GameLift Servers logs each health status report from the game server. The following log messaging example shows a successfully integrated game server:

```
16:50:53,217 INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - SDK
connected: /127.0.0.1:64247
16:50:53,217 INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - SDK pid is 17040,
sdkVersion is 3.1.5 and sdkLanguage is CSharp
16:50:53,217 INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - NOTE: Only SDK
versions 3.1.5 and above are supported in GameLiftLocal!
16:50:53,451 INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - onProcessReady
received from: /127.0.0.1:64247 and ackRequest requested? true
16:50:53,543 INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - onProcessReady
data: logPathsToUpload: "C:\\game\\logs"
logPathsToUpload: "C:\\game\\error"
port: 1935
```

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

Potential error and warning messages include the following:

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

3. Start the AWS CLI.

Once your game server successfully calls `ProcessReady( )`, you can start making client calls. Open another command prompt window and start the AWS CLI tool. The AWS CLI by default uses the Amazon GameLift Servers web service endpoint. You must override this with the Local endpoint in every request using the `--endpoint-url` parameter, as shown in the following example request.

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

In the AWS CLI command prompt window, `AWS gamelift` commands result in responses as documented in the [AWS CLI Command Reference](#).

4. Create a game session.

With the AWS CLI, submit a [`CreateGameSession\(\)`](#) request. The request should follow the expected syntax. For Local, the `FleetId` parameter can be set to any valid string (`^fleet-\S+`).

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

In the Local command prompt window, log messages indicate that Amazon GameLift Servers Local has sent your game server an `onStartGameSession` callback. If a game session is successfully created, your game server responds by invoking `ActivateGameSession`.


```

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

```

In the AWS CLI window, Amazon GameLift Servers responds with a game session object including a game session ID. Notice that the new game session's status is `Activating`. The status changes to `Active` once your game server invokes `ActivateGameSession`. If you want to see the changed status, use the AWS CLI to call `DescribeGameSessions()`.

```

{
  "GameSession": {
    "Status": "ACTIVATING",
    "MaximumPlayerSessionCount": 2,
    "FleetId": "fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d",
    "GameSessionId": "arn:aws:gamelift:local::gamesession/
fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d/gsess-abcdef12-3456-7890-abcd-
ef1234567890",
    "IpAddress": "127.0.0.1",
    "Port": 1935
  }
}

```

Test a game server and client

To check your full game integration, including connecting players to games, you can run both your game server and client locally. This allows you to test programmatic calls from your game client to the Amazon GameLift Servers Local. You can verify the following actions:

- The game client is successfully making AWS SDK requests to the Amazon GameLift Servers Local service, including to create game sessions, retrieve information on existing game sessions, and create player sessions.
- The game server is correctly validating players when they try to join a game session. For validated players, the game server may retrieve player data (if implemented).
- The game server reports a dropped connection when a player leaves the game.
- The game server reports ending a game session.

1. Start Amazon GameLift Servers Local.

Open a command prompt window, navigate to the directory containing the file *GameLiftLocal.jar* and run it. By default, Local listens for requests from game clients on port 8080. To specify a different port number, use the `-p` parameter, as shown in the following example.

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

Once Local starts, you see logs showing that two local servers were started, one listening for your game server and one listening for your game client or the AWS CLI.

2. Start your game server.

Start your Amazon GameLift Servers-integrated game server locally. See [Test a game server](#) for more detail on message logs.

3. Configure your game client for Local and start it.

To use your game client with the Amazon GameLift Servers Local service, you must make the following changes to your game client's setup, as described in [Set up the Amazon GameLift Servers API](#):

- Change the `ClientConfiguration` object to point to your Local endpoint, such as `http://localhost:9080`.

- Set a target fleet ID value. For Local, you do not need a real fleet ID; set the target fleet to any valid string (`^fleet-\S+`), such as `fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d`.
- Set AWS credentials. For Local, you do not need real AWS credentials; you can set the access key and secret key to any string.

In the Local command prompt window, once you start the game client, log messages should indicate that it has initialized the `GameLiftClient` and is successfully communicated with the Amazon GameLift Servers service.

4. Test game client calls to the Amazon GameLift Servers service.

Verify that your game client is successfully making any or all of the following API calls:

- [`CreateGameSession\(\)`](#)
- [`DescribeGameSessions\(\)`](#)
- [`CreatePlayerSession\(\)`](#)
- [`CreatePlayerSessions\(\)`](#)
- [`DescribePlayerSessions\(\)`](#)

In the Local command prompt window, only calls to `CreateGameSession()` result in log messages. Log messages show when Amazon GameLift Servers Local prompts your game server to start a game session (`onStartGameSession` callback) and gets a successful `ActivateGameSession` when your game server invokes it. In the AWS CLI window, all API calls result in responses or error messages as documented.

5. Verify that your game server is validating new player connections.

After creating a game session and a player session, establish a direct connection to the game session.

In the Local command prompt window, log messages should show that the game server has sent an `AcceptPlayerSession()` request to validate the new player connection. If you use the AWS CLI to call `DescribePlayerSessions()`, the player session status should change from `Reserved` to `Active`.

6. Verify that your game server is reporting game and player status to the Amazon GameLift Servers service.

For Amazon GameLift Servers to manage player demand and correctly report metrics, your game server must report various statuses back to Amazon GameLift Servers. Verify that Local is logging events related to following actions. You may also want to use the AWS CLI to track status changes.

- **Player disconnects from a game session** – Amazon GameLift Servers Local log messages should show that your game server calls `RemovePlayerSession()`. An AWS CLI call to `DescribePlayerSessions()` should reflect a status change from `Active` to `Completed`. You might also call `DescribeGameSessions()` to check that the game session's current player count decreases by one.
- **Game session ends** – Amazon GameLift Servers Local log messages should show that your game server calls `TerminateGameSession()`.

 Note

Previous guidance was to call `TerminateGameSession()` when ending a game session. This method is deprecated with Amazon GameLift Servers Server SDK v4.0.1. See [End a game session](#).

- **Server process is terminated** – Amazon GameLift Servers Local log messages should show that your game server calls `ProcessEnding()`. An AWS CLI call to `DescribeGameSessions()` should reflect a status change from `Active` to `Terminated` (or `Terminating`).

Variations with local

When using Amazon GameLift Servers Local, keep in mind the following:

- Unlike the Amazon GameLift Servers web service, Local does not track a server's health status and initiate the `onProcessTerminate` callback. Local simply stops logging health reports for the game server.
- For calls to the AWS SDK, fleet IDs are not validated, and can be any string value that meets the parameter requirements (`^fleet-\S+`).
- Game session IDs created with Local have a different structure. They include the string `local`, as shown here:

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

Debug Amazon GameLift Servers fleet issues

This topic provides guidance on how to resolve issues with your Amazon GameLift Servers managed EC2 fleets.

Fleet creation issues

When you create a managed EC2 fleet, the Amazon GameLift Servers service initiates a workflow that creates the fleet, deploys EC2 instances with your game server build installed, and starts game server processes on each instance. For a detailed description, see . A fleet cannot host game sessions and players until it reaches **Active** status.

You can debug issues that prevent fleets from becoming active by identifying the fleet creation phase where the issue occurred and reviewing fleet creation events and logs. If the logs do not offer useful information, it's possible that the problem is due to an internal service error. In this situation, try to create the fleet again. If the problem persists, try re-uploading the game build to resolve possible file corruption). You can also contact Amazon GameLift Servers support or post a question on the forum.

Downloading and validating the build

During this phase, Amazon GameLift Servers gets your uploaded game server build, extracts the files, and runs any install scripts. If fleet creation fails during these phases, look at fleet events and logs to pinpoint the issue. Possible causes include:

- Amazon GameLift Servers can't get the compressed build file (event `FLEET_BINARY_DOWNLOAD_FAILED`). Verify that the build's storage location can be accessed, that you're creating a fleet in the same AWS Region as the build, and that Amazon GameLift Servers has the correct permissions to access it.
- Amazon GameLift Servers can't extract the build files (event `FLEET_CREATION_EXTRACTING_BUILD`).
- An install script in the build files failed to complete successfully (event `FLEET_CREATION_FAILED_INSTALLER`).

Building fleet resources

Issues during this phase usually involve the allocation and deployment of fleet resources. Possible causes include:

- The requested instance type isn't available.
- The requested fleet type (Spot or On-Demand) isn't available.

Activating game server processes

During this phase, Amazon GameLift Servers is attempting a number of tasks and testing key elements, including the game server's viability, runtime configuration settings, and the game server's ability to connect with the Amazon GameLift Servers service using the Server SDK.

Note

In this phase, you can remotely access a fleet instance to further investigate issues. See [Connect to fleet instances](#).

Possible issues include:

- Server processes don't start running. This suggests an issue with the fleet's runtime configuration settings (events `FLEET_VALIDATION_LAUNCH_PATH_NOT_FOUND` or `FLEET_VALIDATION_EXECUTABLE_RUNTIME_FAILURE`. Verify that you've correctly set the launch path and optional launch parameters.
- Server processes start running, but the fleet fails to activate. If server processes start and run successfully, but the fleet does not move to **Active** status, a likely cause is that the server process is failing to communicate with the Amazon GameLift Servers service. Verify that your game server is making these correct server SDK calls (see [Initialize the server process](#)):
 - Server process fails to initialize (event `SERVER_PROCESS_SDK_INITIALIZATION_TIMEOUT`). The server process is not successfully calling `InitSdk()`.
 - Server process fails to notify Amazon GameLift Servers when it's ready to host a game session (event `SERVER_PROCESS_PROCESS_READY_TIMEOUT`). The server process initialized but didn't call `ProcessReady()` in time.
- A VPC peering connection request failed. For fleets that are created with a VPC peering connection (see [To set up VPC peering with a new fleet](#)), VPC peering is done during this **Activating** phases. If a VPC peering fails for any reason, the new fleet will fail to move to

Active status. You can track the success or failure of the peering request by calling [describe-vpc-peering-connections](#). Be sure to check that a valid VPC peering authorization exists ([describe-vpc-peering-authorizations](#), since authorizations are only valid for 24 hours.

Server process issues

Server processes start but fail quickly or report poor health.

Other than issues with your game build, this outcome can happen when trying to run too many server processes simultaneously on the instance. The optimum number of concurrent processes depends on both the instance type and your game server's resource requirements. Try reducing the number of concurrent processes, which is set in the fleet's runtime configuration, to see if performance improves. You can change a fleet's runtime configuration using either the Amazon GameLift Servers console (edit the fleet's capacity allocation settings) or by calling the AWS CLI command [update-runtime-configuration](#).

Fleet deletion issues

Fleet can't be terminated due to max instance count.

The error message indicates that the fleet being deleted still has active instances, which is not allowed. You must first scale a fleet down to zero active instances. This is done by manually setting the fleet's desired instance count to "0" and then waiting for the scale-down to take effect. Be sure to turn off auto scaling, which will counteract manual settings.

VPC actions are not authorized.

This issue only applies to fleets that you have specifically created VPC peering connections for (see [Set up VPC peering for Amazon GameLift Servers](#)). This scenario occurs because the process of deleting a fleet also includes deleting the fleet's VPC and any VPC peering connections. You must first get an authorization by calling the service API for Amazon GameLift Servers [CreateVpcPeeringAuthorization\(\)](#) or use the AWS CLI command `create-vpc-peering-authorization`. Once you have the authorization, you can delete the fleet.

Remotely access fleet resources

You can connect to any instance in your active Amazon GameLift Servers managed EC2 or managed container fleets. For container fleets, you can also connect to individual containers

running on an instance to troubleshoot game sessions, inspect logs, and debug runtime issues. Common reasons to remotely access an instance include:

- Troubleshoot issues with your game server integration.
- Fine-tune your runtime configuration and other fleet-specific settings.
- Monitor real-time game server activity, such as log tracking.
- Run benchmarking tools using actual player traffic.
- Investigate specific issues with a game session or server process.

When connecting to an instance, keep the following in mind:

- You can connect to any instance in an active fleet. Generally, you can't connect to non-active fleets, such as fleets that are in the process of activating or are in an error state. (These fleets might have limited availability for a short period of time.) For help with fleet activation issues, see [Debug Amazon GameLift Servers fleet issues](#).
- Connecting to an active instance doesn't affect the instance's hosting activity. The instance continues to start and stop server processes based on the runtime configuration. It activates and runs game sessions. The instance might shut down in response to a scale down event or other event.
- Any changes you make to files or settings on the instance might impact the instance's active game sessions and connected players.

Connect to fleet instances

You can remotely connect to any active Amazon GameLift Servers managed EC2 or managed container fleet instance to troubleshoot game server issues, inspect logs, and debug runtime behavior. Connect using the Amazon GameLift Servers console or the AWS CLI.

Remote access through the console

You can connect to fleet instances directly from the Amazon GameLift Servers console using Amazon EC2 Systems Manager (SSM). This method provides secure access without requiring additional setup or credential management. For container fleets, after connecting to the instance, you can access individual containers running on it. For more information, see [Connect to containers](#).

1. In the Amazon GameLift Servers console, choose **Managed EC2** or **Managed containers** from the navigation pane, and then **Fleets**.
2. Choose the fleet ID that contains the instance you want to access.
3. On the fleet details page, choose the **Instances** tab to view all compute instances for the fleet.
4. Select the instance you want to connect to, then choose **Connect**. This displays the Connect to instance dialog which informs you of the details of the connection, and allows you to view the script that will be used to connect to your instance. Confirm by choosing **Connect** again.
5. In the connection dialog, choose **Run** to create a new SSM session. The system authenticates your session through AWS Key Management Service (AWS KMS) and opens a terminal in your browser.

Note

Console-based remote access is available for fleets running server SDK version 5.x. For fleets running earlier SDK versions, use the AWS CLI method described in the following section.

Remote access with the AWS CLI

The following instructions describe how to remotely connect to an instance using the AWS command line interface (CLI). You can also make programmatic calls using the AWS SDK, as documented in the [service API reference for Amazon GameLift Servers](#).

Gather instance data

To connect to an Amazon GameLift Servers managed EC2 fleet instance, you need the following information:

- The ID of the instance you want to connect to. You can use either the instance ID or ARN.
- the server SDK for Amazon GameLift Servers version being used on the instance. The server SDK is integrated with the game build that is running on the instance.

The following instructions describe how complete these tasks using the AWS CLI. You must know the fleet ID for the instance you want to connect to.

1. **Get the compute name.** Get a list of all active computes in the fleet. Call [list-compute](#) with a fleet ID or ARN. For a single-location fleet, specify the fleet identifier only. For a multi-location fleet, specify the fleet identifier and a location. With managed EC2 fleets, `list-compute` returns a list of fleet instances, and the property `ComputeName` is the instance ID. Find the compute you want to access.

Request

```
aws gamelift list-compute \  
  --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \  
  --location sa-east-1
```

Response

```
{  
  "ComputeList": [  
    {  
      "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",  
      "FleetArn": "arn:aws:gamelift:us-west-2::fleet/  
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",  
      "ComputeName": "i-0abc12d3e45fa6b78",  
      "IpAddress": "00.00.000.00",  
      "DnsName":  
"b08444ki909kvqu6zpw3is24x5pyz4b6m05i3jbxvpk9craztu0lqrbbrbnnbkks.uwp57060n1k6dn1nw49b78hg1  
west-2.amazongamelift.com",  
      "ComputeStatus": "Active",  
      "Location": "sa-east-1",  
      "CreationTime": "2023-07-09T22:51:45.931000-07:00",  
      "OperatingSystem": "AMAZON_LINUX_2023",  
      "Type": "c4.large"  
    }  
  ]  
}
```

2. **Find the server SDK version.** For this information you need to look up the build that is deployed to the fleet. Server SDK version is a build property.
 - a. Call [describe-fleet-attributes](#) with a fleet ID or ARN to get the fleet's build ID and ARN.
 - b. Call [describe-build](#) with the build ID or ARN to get the build's server SDK version.

For example:

Request

```
aws gamelift describe-fleet-attributes \  
--fleet-ids fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa
```

Response

```
{  
  "FleetAttributes": [  
    {  
      "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",  
      "ComputeType": "EC2",  
      "BuildId": "build-3333cccc-44dd-55ee-66ff-00001111aa22",  
      . . .  
    }  
  ]  
}
```

Request

```
aws gamelift describe-build \  
--build-id build-3333cccc-44dd-55ee-66ff-00001111aa22
```

Response

```
"Build": {  
  "BuildId": "build-1111aaaa-22bb-33cc-44dd-5555eeee66ff",  
  "Name": "My_Game_Server_Build_One",  
  "OperatingSystem": "AMAZON_LINUX_2023",  
  "ServerSdkVersion": "5.1.1",  
  . . .  
}
```

Connect to an instance (server SDK 5)

If the instance you want to connect to is running a game build with server SDK version 5.x, connect to the instance using Amazon EC2 Systems Manager (SSM). You can access remote instances that

are running either Windows or Linux. For container fleets, after connecting to the instance, you can access individual containers running on it. For more information, see [Connect to containers](#).

Before you start:

Complete the SSM setup steps and install the SSM plugin on your local machine. For more information, see [Setting up SSM](#) and [Install the Session Manager plugin for the AWS CLI](#) in the *Amazon EC2 Systems Manager User Guide*.

1. **Request access credentials for the instance.** Call [get-compute-access](#) with the fleet ID and the compute name for the instance you want to connect to. Amazon GameLift Servers returns a set of temporary credentials for accessing the instance. For example:

Request

```
aws gamelift get-compute-access \  
--compute-name i-11111111a222b333c \  
--fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \  
--region us-west-2
```

Response

```
{  
  "ComputeName": "i-11111111a222b333c",  
  "Credentials": {  
    "AccessKeyId": "ASIAIOSFODNN7EXAMPLE",  
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",  
    "SessionToken": "AQoDYXdzEJr...<remainder of session token>"  
  },  
  "FleetArn": "arn:aws:gamelift:us-west-2::fleet/  
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",  
  "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa"  
}
```

2. **Export the access credentials (optional).** You can export the credentials to environment variables and use them to configure the AWS CLI for the default user. For more details, see [Environment variables to configure the AWS CLI](#) in the AWS Command Line Interface User Guide.

```
export AWS_ACCESS_KEY_ID=ASIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr...<remainder of session token>
```

3. **Connect to the fleet instance.** Start an SSM session with the instance you want to connect to. Include the AWS Region or location of the instance. For more information, including how to set up SSM and the SSM plugin, see [Starting a session \(AWS CLI\)](#) in the *Amazon EC2 Systems Manager User Guide*.

The start-session request will automatically use the credentials that you acquired in Step 1.

```
aws ssm start-session \
--target i-11111111a222b333c \
--region us-west-2 \
```

Note

If you get an access denied error, you might have an `AWS_PROFILE` environment variable set to an AWS profile, which causes AWS CLI to use the wrong credentials for remote access. To resolve, temporarily unset your `AWS_PROFILE` environment variable. Alternatively, you can create a custom AWS profile for your remote access credentials and add the `--profile` command line parameter to your `start-session` request.

Connect to an instance (server SDK 4.x or earlier)

If the instance you want to connect to is running a game build with server SDK version 4 or earlier, use the following instructions. You can connect to instances that are running either Windows or Linux. Connect to a Windows instance using a remote desktop protocol (RDP) client. Connect to a Linux instance using an SSH client.

1. **Request access credentials for the instance.** When you have an instance ID, use the command [get-instance-access](#) to request access credentials. If successful, Amazon GameLift Servers returns the instance's operating system, IP address, and a set of credentials (user name and secret key). The credentials format depends on the instance operating system. Use the following instructions to retrieve credentials for either RDP or SSH.

- **For Windows instances** – To connect to a Windows instance, RDP requires a user name and password. The `get-instance-access` request returns these values as simple strings, so you can use the returned values as is. Example credentials:

```
"Credentials": {
  "Secret": "aA1bBB2cCCd3EEE",
  "UserName": "gl-user-remote"
}
```

- **For Linux instances** – To connect to a Linux instance, SSH requires a user name and private key. Amazon GameLift Servers issues RSA private keys and returns them as a single string, with the newline character (`\n`) indicating line breaks. To make the private key usable, take these steps: (1) convert the string to a `.pem` file, and (2) set permissions for the new file. Example credentials returned:

```
"Credentials": {
  "Secret": "-----BEGIN RSA PRIVATE KEY-----
nEXAMPLEKEYKCAQEAY7WZhaDsra1W3mRlQtvhwyORRX8gnxgDAfRt/gx42kWXsT4rXE/b5CpSgie/
\nvBoU7jLxx92pNHoFnByP+Dc21eyyz6CvjTmWA0JwfWiW5/akH7i05dSrvC7dQkW2duV5QuUdE0QW
\nZ/aNxMniGQE6XAgfwlnXVBwrerrQo+ZWQeqiUwwMkuEbLeJFLhMCvYURpUMSC1oehm449ilx9X1F
\nG50TCFe0zf18dqqCP6GzbPaIjiU19xX/az0R9V+tpU0zEL+wmXnZt3/nHPQ5xvD20JH67km6SuPW
\noPzev/D8V+x4+bHthfSjR9Y7DvQFjfbVwHXigBdtZcU2/wei8D/HYwIDAQABaoIBAGZ1kaEvnrrqu
\n/uler7vgIn5m71N5LKw4hJLAIW6tUT/fzvtcHK0SkbQCQXuriHmQ2MQyJX/0kn2NfjLV/
ufGxbL1\nmb5qwmGUNepJaZD6QSSs3kICLwUUYUiGfc0uiSbmJoap/
GTLU0W5Mfcv36PaBUNy5p53V6G7hXb2\nbahyWyJNfjLe4M86yd2YK3V2CmK+X/
B0sShnJ36+hjrXPPWmV3N9zEmCdJjA+K15DYmhm/
tJWSD9\n81oGk9TopEp7CkIfatEATyyZiVqoRq6k64iuM9JkA30zdXzMqexXVJ1TLZVEH0E7bh1Y9d801ozR
\noQs/FiZNAx2iijCWyv0lpjE73+kCgYEA9mZtyhkHkFDpwrSM1APaL8oNAbbjwEy7Z5Mqfq1
+1Ip1\nYkriL0DbLXlvRAH+yHPRit2hH0jtUNZh4Axv+cpq09qbUI3+43eEy24B7G/Uh
+GTfbjsXs0xQx/x\np9otyVwc7hsQ5TA5PZb
+mvkJ50BEKzet9XcKw0NBYELGhnEPe7cCgYEA06Vgov6YHleHui9kHuws
\nayav0elc5zkxjF9nfHFJRry21R1trw2Vdpn+9g481URrpzWV0Eihvm+XTtmaZ1Sp//1kq75XDwnU
\nWA8gkn603QE3fq2yN98BURsAKdJfJ5RL1HvGQvTe10HLYYXpJnEkHv+Unl2ajLivWUt5pbBrKbUC
\nngYBjb0+0Zk0sCcpZ29sbzjYjpIddErySIyRX5gV2uNQwAjLdp9PfN295yQ+BxMBXiIycWVQiw0bH
\nnoMo7yykABY70zd5wQewBQ4AdSlWSX4nGDtsiFxiWiI5sKuAAe0CbTosy1s8w8fxoJ5Tz1sdoxNeGs
\nArq6Wv/G16zQuAE9zK9vwwKBgF+09VI/1wJBirsDGz9whVwFFPrTkJNvJZzYt69qezx1sjgFKshy
\nWBhd4xHZtmCqpBP1AymEjr/T0lboxyARMXmNIOWIANXMGb4KGSyl1mzSVAoQ+fqR+cJ3d0dyP11j
\nnjjb0Ed/NY8fr1NDxAVHE8BSkdsx2f6ELEyBKJSR9snRAoGAMrTwYneXzvTskF/S5Fyu0i0egLda
\nNWUH38v/nDCgEpIXD5Hn3qAEcju1IjmbwlvT+nY2jVhv7UGd8MjwUTNGItdb6nsYqM2asrnF3qS
\nVRkAKKKYeGjkuUfVTrW0YFjXkfcRr/V+QFL50ndHAKJXjW7a4ejJLncTzmZSpYzwApc=\n-----END
RSA PRIVATE KEY-----",
```

```
"UserName": "gl-user-remote"
}
```

When using the AWS CLI, you can automatically generate a `.pem` file by including the `--query` and `--output` parameters to your `get-instance-access` request.

To set permissions on the `.pem` file, run the following command:

```
$ chmod 400 MyPrivateKey.pem
```

2. **Open a port for the remote connection.** You can access instances in Amazon GameLift Servers fleets through any port authorized in the fleet configuration. You can view a fleet's port settings using the command [describe-fleet-port-settings](#).

As a best practice, we recommend opening ports for remote access only when you need them and closing them when you're finished. You can't update port settings after creating a fleet but before it's active. If you get stuck, re-create the fleet with the port settings open.

Use the command [update-fleet-port-settings](#) to add a port setting for the remote connection (such as 22 for SSH or 3389 for RDP). For the IP range value, specify the IP addresses for the devices you plan to use to connect (converted to CIDR format). Example:

```
$ AWS gamelift update-fleet-port-settings
  --fleet-id "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa"
  --inbound-permission-authorizations
    "FromPort=22,ToPort=22,IpRange=54.186.139.221/32,Protocol=TCP"
```

The following example opens up port 3389 on a Windows fleet

```
$ AWS gamelift update-fleet-port-settings
  --fleet-id "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa"
  --inbound-permission-authorizations
    "FromPort=3389,ToPort=3389,IpRange=54.186.139.221/32,Protocol=TCP"
```

3. **Open a remote connection client.** Use Remote Desktop for Windows or SSH for Linux instances. Connect to the instance using the IP address, port setting, and access credentials.

SSH example:

```
ssh -i MyPrivateKey.pem gl-user-remote@192.0.2.0
```

View files on remote instances

Once connected to a fleet instance, you have full user and administrative access. This means you also have the ability to cause errors or failures with game hosting. If the instance is hosting games with active players, you might run the risk of crashing game sessions and dropping players, or disrupting game shutdown processes which could cause errors in saved game data and logs.

For container fleets, after accessing the container (see [Connect to containers](#)), the following file locations apply within the container filesystem. For EC2 fleets, these file locations apply directly on the instance.

Look for these resources on a hosting instance:

- **Game build files.** These files are the game build that you uploaded to Amazon GameLift Servers. They include one or more game server executables, assets, and dependencies. Game build files are in a root directory called `game`:
 - On Windows: `c:\game`
 - On Linux: `/local/game`
- **Game log files.** Find the log files that your game server generates in the game root directory at whatever directory path you designated.
- **Amazon GameLift Servers hosting resources.** The root directory `Whitewater` contains files used by the Amazon GameLift Servers service to manage game hosting activity. Don't modify these files for any reason.
- **Runtime configuration.** Don't access runtime configuration for individual instances. To make changes to a runtime configuration property, update the fleet's runtime configuration (see the AWS SDK operation [UpdateRuntimeConfiguration](#) or the AWS CLI [update-runtime-configuration](#)).
- **Fleet data.** A JSON file contains information about the fleet that the instance belongs to, for use by server processes running on the instance. The JSON file is in the following location:
 - On Windows: `C:\GameMetadata\gamelift-metadata.json`
 - On Linux: `/local/gamemetadata/gamelift-metadata.json`
- **TLS certificates.** If the instance is on a fleet that has TLS certificate generation enabled, look for certificate files, including the certificate, certificate chain, private key, and root certificate in the following location:
 - On Windows: `c:\\GameMetadata\Certificates`

- On Linux: `/local/gamemetadata/certificates/`

Connect to containers

For Amazon GameLift Servers container fleets, you can access game server containers running on a fleet instance. Use container access to troubleshoot game sessions, inspect logs, and debug runtime issues.

Connect to a container

 Before you start:

Connect to the fleet instance. For instructions, see [Connect to fleet instances](#).

Run the following command to list running containers on the instance:

```
sudo docker ps
```

The output lists all containers running on the instance, including game server containers and internal Amazon GameLift Servers containers. Look for containers with your game server image to identify the game server containers.

Example output:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
b9676e9489f5	game-server-container	"/bin/sh -c ./ \$GAME_..."	2 days ago	Up 2 days
1d1c8443efe2	support-container	"/bin/sh -c ./ \$SUPPO..."	2 days ago	Up 2 days

To connect to a game server container, use the container short ID from the CONTAINER ID column. This gives you full read and write access to the container filesystem.

```
sudo docker exec -it container-short-id sh
```

Connect to a container through the console

You can connect to game server containers from the Amazon GameLift Servers console using Amazon EC2 Systems Manager (SSM). This method provides secure access without requiring

additional setup or credential management. You can connect to a container from either the **Computes** tab or the **Game sessions** tab on the fleet details page.

1. In the Amazon GameLift Servers console, choose **Managed containers** from the navigation pane, and then **Fleets**.
2. Choose the fleet ID that contains the container or game session you want to access.
3. On the fleet details page, choose one of the following tabs:
 - **Computes** – Lists the containers running on the fleet. Select the container you want to connect to.
 - **Game sessions** – Lists the game sessions for the fleet. Select the game session to connect to the container hosting it.
4. Choose **Connect**. Copy the command displayed to connect to the container, then choose **Connect** again.
5. In the connection dialog, choose **Run** to create a new SSM session. The system authenticates your session through AWS Key Management Service (AWS KMS) and opens a terminal in your browser.
6. Once you have connected to the instance, paste the docker command from step 4 and execute it on the instance to access the container.

Connect to container hosting a game session

To connect to the game server container hosting a specific game session, follow these steps.

1. **Get the compute name.** Call [describe-game-sessions](#) to get the ComputeName for the game session.

Request

```
aws gamelift describe-game-sessions \
  --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \
  --game-session-id arn:aws:gamelift:us-west-2::gamesession/
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa/gs-1111aaaa-2222-3333-4444-5555bbbb66cc
```

Response

```
{
```

```
"GameSessions": [  
  {  
    "GameSessionId": "arn:aws:gamelift:us-west-2::gamesession/  
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa/  
gs-1111aaaa-2222-3333-4444-5555bbbb66cc",  
    "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",  
    "ComputeName": "62c5ff7f7a9a445d84877074c80aeafc",  
    "Status": "ACTIVE",  
    . . .  
  }  
]  
}
```

Note the ComputeName value from the response (for example, 62c5ff7f7a9a445d84877074c80aeafc).

2. **Get compute access and container attributes.** Call [get-compute-access](#) with the fleet ID and compute name.

The response includes the following fields:

- **ContainerIdentifiers** – The ContainerName and ContainerRuntimeId for each container.
- **GameServerContainerGroupDefinitionArn** – The ARN of the container group definition.
- **Credentials** – Temporary credentials to connect to the instance.

Request

```
aws gamelift get-compute-access \  
  --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \  
  --compute-name 62c5ff7f7a9a445d84877074c80aeafc
```

Response

```
{  
  "ComputeName": "62c5ff7f7a9a445d84877074c80aeafc",  
  "ContainerIdentifiers": [  
    {  
      "ContainerName": "game-server",
```

```

    "ContainerRuntimeId":
    "02accb92cd9bef3373300e7151d5c2b3dcca3b06eff1bb4e345085fc008d4678"
  },
  "Credentials": {
    "AccessKeyId": "ASIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY",
    "SessionToken": "AQoDYXdzEJr...<remainder of session token>"
  },
  "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
  "GameServerContainerGroupDefinitionArn": "arn:aws:gamelift:us-
west-2::containergroupdefinition/MyGameServerGroup"
}

```

3. **Connect to the instance.** Use the credentials from step 2 to connect to the fleet instance. For detailed instructions, see [Connect to fleet instances](#).
4. **Find the game server container name.** Call [describe-container-group-definition](#) using the GameServerContainerGroupDefinitionArn from step 2 to identify the game server container name.

Request

```

aws gamelift describe-container-group-definition \
  --name arn:aws:gamelift:us-west-2::containergroupdefinition/MyGameServerGroup

```

Response

```

{
  "ContainerGroupDefinition": {
    "ContainerGroupDefinitionArn": "arn:aws:gamelift:us-
west-2:123456789012:containergroupdefinition/MyGameServerGroup:3",
    "Name": "MyGameServerGroup",
    "ContainerGroupType": "GAME_SERVER",
    "GameServerContainerDefinition": {
      "ContainerName": "game-server",
      . . .
    },
    . . .
  }
}

```

Note the `GameServerContainerDefinition.ContainerName` value (for example, `game-server`).

5. **Identify the game server container runtime ID.** Using the game server container name from the previous step, find the matching entry in the `ContainerIdentifiers` from the `get-compute-access` response in step 2. Note the `ContainerRuntimeId` value.
6. **Connect to the container.** Use the `ContainerRuntimeId` as the container ID and run the following command:

```
sudo docker exec -it  
02accb92cd9bef3373300e7151d5c2b3dcca3b06eff1bb4e345085fc008d4678 sh
```

View container port mappings

Port mappings show how container ports map to connection ports on your fleet instances. Each container port that accepts inbound traffic is assigned a connection port on the instance. You can check port mappings to discover which connection ports map to your container ports or to troubleshoot connection issues. You can view port mappings in the Amazon GameLift Servers console, or use the AWS CLI or AWS SDK.

View port mappings in the console

In the Amazon GameLift Servers console, choose **Managed containers** from the navigation pane, and then **Fleets**. Choose a fleet to open the fleet details page. You can view port mappings in the following locations:

- **Instance details page** – Shows port mappings for the per-instance container group on the selected instance.
- **Compute details page** – Shows port mappings for the game server container group on the selected compute.

Both pages include a search field to filter port mappings by container name.

View port mappings with the AWS CLI

Use the `describe-container-group-port-mappings` command to retrieve port mappings for a container group on a fleet instance. Specify the container group type and either a `--compute-`

name (for game server groups) or an `--instance-id` (for per-instance groups). Optionally, include the `--container-name` parameter to filter results to a specific container.

Example: Game server container group

The following command retrieves port mappings for the game server container group on a specific compute.

```
aws gamelift describe-container-group-port-mappings \
  --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \
  --container-group-type GAME_SERVER \
  --compute-name 62c5ff7f7a9a445d84877074c80aeafc
```

If successful, Amazon GameLift Servers returns a response like the following:

```
{
  "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
  "Location": "us-west-2",
  "ContainerGroupDefinitionArn": "arn:aws:gamelift:us-west-2:123456789012:containergroupdefinition/MyGameServerGroup",
  "ContainerGroupType": "GAME_SERVER",
  "ComputeName": "62c5ff7f7a9a445d84877074c80aeafc",
  "InstanceId": "i-1234567890abcdef0",
  "ContainerGroupPortMappings": [
    {
      "ContainerName": "MyGameServer",
      "ContainerRuntimeId": "a1b2c3d4e5f6",
      "ContainerPortMappings": [
        {
          "ContainerPort": 7777,
          "ConnectionPort": 1025,
          "Protocol": "UDP"
        },
        {
          "ContainerPort": 8080,
          "ConnectionPort": 1026,
          "Protocol": "TCP"
        }
      ]
    }
  ]
}
```

Example: Per-instance container group

The following command retrieves port mappings for the per-instance container group on a specific instance.

```
aws gamelift describe-container-group-port-mappings \  
  --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \  
  --container-group-type PER_INSTANCE \  
  --instance-id i-1234567890abcdef0
```

If successful, Amazon GameLift Servers returns a response like the following:

```
{  
  "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",  
  "Location": "us-west-2",  
  "ContainerGroupDefinitionArn": "arn:aws:gamelift:us-  
west-2:123456789012:containergroupdefinition/MyPerInstanceGroup",  
  "ContainerGroupType": "PER_INSTANCE",  
  "InstanceId": "i-1234567890abcdef0",  
  "ContainerGroupPortMappings": [  
    {  
      "ContainerName": "MySupportContainer",  
      "ContainerRuntimeId": "f6e5d4c3b2a1",  
      "ContainerPortMappings": [  
        {  
          "ContainerPort": 8443,  
          "ConnectionPort": 2025,  
          "Protocol": "TCP"  
        }  
      ]  
    }  
  ]  
}
```

Scaling game hosting capacity with Amazon GameLift Servers

Hosting capacity, measured in instances, represents the number of game sessions that Amazon GameLift Servers can host concurrently and the number of concurrent players that those game sessions can accommodate. One of the most challenging tasks with game hosting is scaling capacity to meet player demand without wasting money on resources that you don't need.

Capacity is adjusted at the fleet location level. All fleets have at least one location: the fleet's home AWS Region. When viewing or scaling capacity, the information is listed by location, including the fleet's home Region and any additional remote locations.

You can manually set the number of instances to maintain, or you can set up auto scaling to dynamically adjust capacity as player demand changes. We recommend that you start by turning on the target-based auto scaling option. The goal of target-based auto scaling is to maintain enough hosting resources to accommodate current players plus a little extra to handle unexpected spikes in player demand. For most games, target-based auto scaling offers a highly effective scaling solution.

You can do most fleet scaling activities using the Amazon GameLift Servers console. You can also use an AWS SDK or the AWS Command Line Interface (AWS CLI) with the [service API for Amazon GameLift Servers](#).

Topics

- [To manage fleet capacity in the console](#)
- [Set Amazon GameLift Servers capacity limits](#)
- [Manually set capacity for an Amazon GameLift Servers fleet](#)
- [Auto-scale fleet capacity with Amazon GameLift Servers](#)
- [Manage Scaling an Amazon GameLift Servers Fleet To/From Zero](#)
- [Scale Amazon GameLift Servers container fleets](#)

To manage fleet capacity in the console

1. Open the [Amazon GameLift Servers console](#).

2. In the navigation pane, choose **Hosting, Fleets**.
3. On the **Fleets** page, choose the name of an active fleet to open the fleet's detail page.
4. Choose the **Scaling** tab. On this tab, you can:
 - View historical scaling metrics for the entire fleet.
 - View and update capacity settings for each fleet location, including scaling limits and current capacity settings.
 - Update target-based auto scaling, view rule-based auto scaling policies applied to the entire fleet, and suspend auto scaling activity for each location.

Set Amazon GameLift Servers capacity limits

When scaling hosting capacity for an Amazon GameLift Servers fleet location, either manually or by auto scaling, consider the location's scaling limits. All fleet locations have a minimum and maximum limit that define the allowed range for the location's capacity. By default, limits on fleet locations have a minimum of 0 instances and a maximum of 1 instance. Before you can scale a fleet location, adjust the limits.

If you're using auto scaling, the maximum limit allows Amazon GameLift Servers to scale up a fleet location to meet player demand but prevents runaway hosting costs, such as during a DDOS attack. Set up an [Amazon CloudWatch alarm](#) to notify you when capacity approaches the maximum limit, so you can evaluate the situation and manually adjust as needed. (You can also [create a billing alarm](#) to monitor AWS costs.) The minimum limit is useful to maintain hosting availability, even when player demand is low.

You can set capacity limits for a fleet's locations in the [Amazon GameLift Servers console](#) or by using the AWS Command Line Interface (AWS CLI).

To set capacity limits

Console

1. Open the [Amazon GameLift Servers console](#).
2. In the navigation pane, choose **Hosting, Fleets**.
3. On the **Fleets** page, choose the name of an active fleet to open the fleet's detail page.
4. On the **Scaling** tab, under **Scaling capacity**, select a fleet location, and then choose **Edit**.

5. In the **Edit scaling capacity** dialog box, set instance counts for **Min size**, **Desired instances**, and **Max size**.
6. Choose **Confirm**.

AWS CLI

1. **Check current capacity settings.** In a command line window, use the [describe-fleet-location-capacity](#) command with the fleet ID and location that you want to change capacity for. This command returns a [FleetCapacity](#) object that includes the location's current capacity settings. Determine whether the new instance limits can accommodate the current desired instances setting.

```
aws gamelift describe-fleet-location-capacity \  
  --fleet-id <fleet identifier> \  
  --location <location name>
```

2. **Update limit settings.** In a command line window, use the [update-fleet-capacity](#) command with the following parameters. You can adjust both instance limits and desired instance count with the same command.

```
--fleet-id <fleet identifier>  
--location <location name>  
--max-size <maximum capacity for scaling>  
--min-size <minimum capacity for scaling>  
--desired-instances <fleet capacity goal>
```

Example:

```
aws gamelift update-fleet-capacity \  
  --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \  
  --location us-west-2 \  
  --max-size 10 \  
  --min-size 1 \  
  --desired-instances 10
```

If your request is successful, Amazon GameLift Servers returns the fleet ID. If the new max-size or min-size value conflicts with the current desired-instances setting, Amazon GameLift Servers returns an error.

Manually set capacity for an Amazon GameLift Servers fleet

When you create a new fleet, Amazon GameLift Servers automatically sets the desired instances to one instance in each fleet location. Then, Amazon GameLift Servers deploys one new instance in each location. To change fleet capacity, you can add a target-based auto scaling policy, or you can manually set the number of instances that you want for a location.

Setting a fleet's capacity manually can be useful when you don't need auto scaling or when you need to hold capacity at a specified level. Manually setting capacity works only if you aren't using a target-based auto scaling policy. If you have a target-based auto scaling policy, it immediately resets the desired capacity based on its own scaling rules.

You can manually set capacity in the Amazon GameLift Servers console or by using the AWS Command Line Interface (AWS CLI). The fleet's status must be active.

Suspend auto scaling

You can suspend all auto scaling activity for each fleet location. With auto scaling suspended, the desired number of instances in the fleet location remains the same unless manually changed. When you suspend auto scaling for a location, it affects the fleet's current policies and any policies that you may define in the future.

To manually set fleet capacity

Console

1. Open the [Amazon GameLift Servers console](#).
2. In the navigation pane, choose **Hosting, Fleets**.
3. On the **Fleets** page, choose the name of an active fleet to open the fleet's detail page.
4. On the **Scaling** tab, under **Suspended auto scaling locations**, select each location that you want to suspend auto scaling for, and then choose **Suspend**.
5. Under **Scaling capacity**, select a location that you want to set manually, and then choose **Edit**.
6. In the **Edit scaling capacity** dialog box, set your preferred value for **Desired instances**, and then choose **Confirm**. This tells Amazon GameLift Servers the number of instances to maintain in an active state, ready to host game sessions.

Amazon GameLift Servers responds to the changes by deploying additional instances or shutting down unneeded ones. As Amazon GameLift Servers completes this process, the number of active instances in the location changes to match the updated desired instances value. This process may take a little time.

AWS CLI

1. **Check current capacity settings.** In a command line window, use the [describe-fleet-location-capacity](#) command with the fleet ID and location that you want to change capacity for. This command returns a [FleetCapacity](#) object that includes the location's current capacity settings. Determine whether the instance limits can accommodate the new desired instances setting.

```
aws gamelift describe-fleet-location-capacity \  
  --fleet-id <fleet identifier> \  
  --location <location name>
```

2. **Update desired capacity.** Use the [update-fleet-capacity](#) command with the fleet ID, location, and a new value for desired instances. If this value falls outside the current limit range, you can adjust limit values in the same command.

```
--fleet-id <fleet identifier>  
--location <location name>  
--desired-instances <fleet capacity as an integer>  
--max-size <maximum capacity> [Optional]  
--min-size <minimum capacity> [Optional]
```

Example:

```
aws gamelift update-fleet-capacity \  
  --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \  
  --location us-west-2 \  
  --desired-instances 5 \  
  --max-size 10 \  
  --min-size 1
```

If your request is successful, Amazon GameLift Servers returns the fleet ID. If the new desired instances setting is outside the minimum and maximum limits, Amazon GameLift Servers returns an error.

Auto-scale fleet capacity with Amazon GameLift Servers

Use auto scaling in Amazon GameLift Servers to dynamically scale your fleet capacity in response to game server activity. As players arrive and start game sessions, auto scaling can add more instances; as player demand wanes, auto scaling can terminate unneeded instances. Auto scaling is an effective way to minimize your hosting resources and costs, while still providing a smooth, fast player experience.

To use auto scaling, you create scaling policies that tell Amazon GameLift Servers when to scale up or down. There are two types of scaling policies: target-based and rule-based. The target-based approach—target tracking—is a complete solution. We recommend it as the simplest and most effective option. Rule-based scaling policies require you to define each aspect of the auto scaling decision-making process, which is useful for addressing specific issues. This solution works best as a supplement to target-based auto scaling.

You can manage target-based auto scaling using the Amazon GameLift Servers console, the AWS Command Line Interface (AWS CLI), or an AWS SDK. You can manage rule-based auto scaling using the AWS CLI or an AWS SDK only, although you can view rule-based scaling policies in the console.

Topics

- [Target-based auto scaling](#)
- [Auto scale with rule-based policies](#)

Target-based auto scaling

Target-based auto scaling for Amazon GameLift Servers adjusts capacity levels based on the fleet metric `PercentAvailableGameSessions`. This metric represents the fleet's available buffer for sudden increases in player demand.

The primary reason for maintaining a capacity buffer is player wait time. When game session slots are ready and waiting, it takes seconds to get new players into game sessions. If no resources are available, players must wait for existing game sessions to end or for new resources to become available. It can take minutes to start up new instances and server processes.

When setting up target-based auto scaling, specify the size of the buffer that you want the fleet to maintain. Because `PercentAvailableGameSessions` measures the percentage of available resources, the actual buffer size is a percentage of the fleet location's capacity. A scaling policy is applied at the fleet level and propagated to each location, where each location scales

independently based on its own metrics. Amazon GameLift Servers adds or removes instances in each location to maintain the target buffer size. With a large buffer, you minimize wait time, but you also pay for extra resources that you may not use. If your players are more tolerant of wait times, you can lower costs by setting a small buffer.

To set target-based auto scaling

Console

1. Open the [Amazon GameLift Servers console](#).
2. In the navigation pane, choose **Hosting, Fleets**.
3. On the **Fleets** page, choose the name of an active fleet to open the fleet's detail page.
4. Choose the **Scaling** tab. This tab displays the fleet's historical scaling metrics and contains controls for adjusting current scaling settings.
5. Under **Scaling capacity**, check that the **Min size** and **Max size** limits are appropriate for the fleet. With auto scaling enabled, capacity adjusts between these two limits.
6. In **Target-based auto scaling policy**, choose **Edit**.
7. In the **Edit target-based auto scaling policy** dialog box, for **Percent available game sessions**, set the percentage that you want to maintain, and then choose **Confirm**. After you've confirmed the settings, Amazon GameLift Servers adds a new target-based policy under **Target-based auto scaling policy**.

AWS CLI

1. **Set capacity limits.** Set the limit values using the [update-fleet-capacity](#) command. For more information, see [Set Amazon GameLift Servers capacity limits](#).
2. **Create a new policy.** Open a command-line window and use the [put-scaling-policy](#) command with your policy's parameter settings. To update an existing policy, specify the policy's name and provide a complete version of the updated policy.

```
--fleet-id <unique fleet identifier>
--name "<unique policy name>"
--policy-type <target- or rule-based policy>
--metric-name <name of metric>
--target-configuration <buffer size>
```

Example:

```
aws gamelift put-scaling-policy \  
  --fleet-id "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa" \  
  --name "My_Target_Policy_1" \  
  --policy-type "TargetBased" \  
  --metric-name "PercentAvailableGameSessions" \  
  --target-configuration "TargetValue=5"
```

Auto scale with rule-based policies

Rule-based scaling policies in Amazon GameLift Servers provide fine-grained control when auto scaling a fleet's capacity in response to player activity. For each policy, you can link scaling to one of several fleet metrics, identify a trigger point, and customize the responding scale-up or scale-down event. Rule-based policies are useful for supplementing [target-based scaling](#) to handle special circumstances.

A rule-based policy states the following: "If a fleet metric meets or crosses a threshold value for a certain length of time, then change the fleet's capacity by a specified amount." This topic describes the syntax used to construct a policy statement and provides help with creating and managing your rule-based policies.

Manage rule-based policies

Create, update, or delete rule-based policies using an AWS SDK or the AWS Command Line Interface (AWS CLI) with the [Service API for Amazon GameLift Servers](#). You can view all active policies in the Amazon GameLift Servers console.

To temporarily stop all scaling policies for a fleet, use the AWS CLI command [stop-fleet-actions](#).

To create or update a rule-based scaling policy (AWS CLI):

1. **Set capacity limits.** Set either or both limit values using the [update-fleet-capacity](#) command. For more information, see [Set Amazon GameLift Servers capacity limits](#).
2. **Create a new policy.** Open a command line window and use the [put-scaling-policy](#) command with your policy's parameter settings. To update an existing policy, specify the policy's name and provide a complete version of the updated policy.

```
--fleet-id <unique fleet identifier>  
--name "<unique policy name>"
```

```
--policy-type <target- or rule-based policy>
--metric-name <name of metric>
--comparison-operator <comparison operator>
--threshold <threshold integer value>
--evaluation-periods <number of minutes>
--scaling-adjustment-type <adjustment type>
--scaling-adjustment <adjustment amount>
```

Example:

```
aws gamelift put-scaling-policy \
  --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \
  --name "Scale up when AGS<50" \
  --policy-type RuleBased \
  --metric-name AvailableGameSessions \
  --comparison-operator LessThanThreshold \
  --threshold 50 \
  --evaluation-periods 10 \
  --scaling-adjustment-type ChangeInCapacity \
  --scaling-adjustment 1
```

To delete a rule-based scaling policy using the AWS CLI:

- Open a command line window and use the [delete-scaling-policy](#) command with the fleet ID and policy name.

Example:

```
aws gamelift delete-scaling-policy \
  --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \
  --name "Scale up when AGS<50"
```

Syntax for auto scaling rules

To construct a rule-based scaling policy statement, specify six variables:

If *<metric name>* remains *<comparison operator>* *<threshold value>* for *<evaluation period>*, then change fleet capacity using *<adjustment type>* to/by *<adjustment value>*.

For example, this policy statement starts a scale-up event whenever a fleet's extra capacity is less than what's needed to handle 50 new game sessions:

If `AvailableGameSessions` remains at less than 50 for 10 minutes, then change fleet capacity using `ChangeInCapacity` by 1 instances.

Metric name

To start a scaling event, link an auto scaling policy to one of the following fleet-specific metrics. For complete metric descriptions, see [Amazon GameLift Servers metrics for fleets](#).

- Activating game sessions
- Active game sessions
- Available game sessions
- Percent available game sessions
- Active instances
- Available player sessions
- Current player sessions
- Idle instances
- Percent idle instances

If the fleet is in a game session queue, you can use the following metrics:

- Queue depth – The number of pending game session requests this fleet is the best available hosting location for.
- Wait time – Fleet-specific wait time. The length of time that the oldest pending game session request has been waiting to be fulfilled. A fleet's wait time is equal to the oldest current request's time in queue.

Comparison operator

Tells Amazon GameLift Servers how to compare the metric data to the threshold value. Valid comparison operators include greater than (`>`), less than (`<`), greater than or equal (`>=`), and less than or equal (`<=`).

Threshold value

When the specified metric value meets or crosses the threshold value, it starts a scaling event. This value is always a positive integer.

Evaluation period

The metric must meet or cross the threshold value for the full length of the evaluation period before starting a scaling event. The evaluation period length is consecutive; if the metric retreats from the threshold, the evaluation period starts over again.

Adjustment type and value

This set of variables works together to specify how Amazon GameLift Servers should adjust the fleet's capacity when a scaling event starts. Choose from three possible adjustment types:

- **Change in capacity** – Increase or decrease the current capacity by a specified number of instances. Set the adjustment value to the number of instances to add or remove from the fleet. Positive values add instances, while negative values remove instances. For example, a value of "-10" scales down the fleet by 10 instances, regardless of the fleet's total size.
- **Percent change in capacity** – Increase or decrease the current capacity by a specified percentage. Set the adjustment value to the percentage that you want to increase or decrease the fleet capacity by. Positive values add instances, while negative values remove instances. For example, for a fleet with 50 instances, a percentage change of "20" adds 10 instances to the fleet.
- **Exact capacity** – Increase or decrease the current capacity to a specific value. Set the adjustment value to the exact number of instances that you want to maintain in the fleet.

Tips for rule-based auto scaling

The following suggestions can help you get the most out of auto scaling with rule-based policies.

Use multiple policies

You can have multiple auto scaling policies for a fleet at the same time. The most common scenario is to have a target-based policy manage most scaling needs and use rule-based policies to handle edge cases. There are no limits on using multiple policies.

With multiple policies, each policy behaves independently. There is no way to control the sequence of scaling events. For example, if you have multiple policies driving scaling up, it's possible that player activity could start multiple scaling events simultaneously. Avoid policies that start each other. For example, you could create an infinite loop if you create scale-up and scale-down policies that set capacity beyond the threshold of each other.

Set maximum and minimum capacity

Each fleet has a maximum and minimum capacity limit. This feature is important when using auto scaling. Auto scaling never sets capacity to a value outside of this range. By default, newly created fleets have a minimum of 0 and a maximum of 1. For your auto scaling policy to affect capacity as intended, increase the maximum value.

Fleet capacity is also constrained by limits on the fleet's instance type and by service quotas in your AWS account. You can't set a minimum and maximum outside these limits and account quotas.

Track metrics after a change in capacity

After changing capacity in response to an auto scaling policy, Amazon GameLift Servers waits 10 minutes before responding to triggers from the same policy. This wait gives Amazon GameLift Servers time to add the new instances, launch the game servers, connect players, and start collecting data from the new instances. During this time, Amazon GameLift Servers evaluates the policy against the metric and tracks the policy's evaluation period, which restarts after a scaling event occurs. This means that a scaling policy could start another scaling event immediately after the wait time is over.

There is no wait time between scaling events that different auto scaling policies start.

Manage Scaling an Amazon GameLift Servers Fleet To/From Zero

Amazon GameLift Servers supports automatic scaling to and from zero instances based on game session activity. This managed capacity option allows your Fleet locations to scale-in to zero instances following a defined period of no game session activity and automatically scale-out when game sessions are requested.

Scaling to and from zero instances provides several advantages:

- **Cost optimization** – Eliminate compute costs during periods of inactivity by running zero instances when there is no game session activity.
- **Automatic reactive scale-out** – Fleets locations automatically scale-out to one instance when a game session is requested, eliminating the need for manual intervention.
- **Simplified management** – No need to manually adjust Fleet capacity to/from zero based on anticipated player demand or development needs.

When you enable Scale To/From Zero on a fleet, Amazon GameLift Servers monitors game session activity and automatically adjusts Fleet capacity:

- **Scale-in to zero** – After a configured period with no game sessions activity, Amazon GameLift Servers scales-in the Fleet location to zero instances.
- **Scale-out from zero** – When a game session creation request is received, Amazon GameLift Servers scales-out the Fleet location to one instance, allowing auto scaling to resume.
- **Continued scaling** – Following scale-out the fleet resumes using configured auto scaling policies to manage capacity.

Scale-in behavior

Amazon GameLift Servers begins the scale-in process for a Fleet location after the configured inactivity period has elapsed with no game session activity. This is defined as a period where::

- There are no active game sessions in the Fleet location.
- No requests have been made to create new game sessions in the Fleet location.

During scale-in, Amazon GameLift Servers will set minimum and desired capacity for the Fleet location to zero, rapidly scaling-in for cost savings.

Scale-out behavior

When a game session creation request is received while the Fleet location is at zero instances:

- Amazon GameLift Servers immediately initiates scale-out of one instance.
- Attempted placement of the game session may continue with other Fleets or Fleet locations, depending on configuration of Queues (if used).

Note

Scale-out from zero takes time to provision and initialize instances. Players may experience longer wait times for the first game session after a period of inactivity. For this reason this feature is best paired with multi-location Fleets and/or Queues.

Configuring Scale To/From Zero

Scaling To/From Zero is configured by updating an existing fleet.

Console

1. Open the [Amazon GameLift Servers console](#).
2. In the navigation pane, choose **Hosting, Fleets**.
3. On the **Fleets** page, choose the name of an active fleet to open the fleet's detail page.
4. Under **Scaling**, select each location for which you want to configure scale to/from zero, and then choose **Edit**.
5. In the **Edit scaling capacity** dialog box, select "Automatic" for **Minimum capacity strategy**, set your preferred value in minutes for **Set minimum capacity to 0 after** and then choose **Confirm**.

Amazon GameLift Servers will then scale-in the selected fleet locations to 0 instances once there has been no game session activity for the configured duration. Following this, when a request for a game session is made in this location, Amazon GameLift Servers will scale-out one instance as quickly as possible. This process will take a little time.

AWS CLI

- **Configure scale to/from zero.** In a command line window, use the [update-fleet-capacity](#) command with the fleet ID, location and managed capacity configuration to configure scale to/from zero.

```
aws gamelift update-fleet-capacity \
    --fleet-id <fleet identifier> \
    --location <location name> \
    --managed-capacity-configuration
    ScaleInAfterInactivityMinutes=60,ZeroCapacityStrategy=SCALE_TO_AND_FROM_ZERO
```

Example:

```
aws gamelift update-fleet-capacity \
    --fleet-id
    fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \
    --location us-west-2 \
```

```
--desired-instances 5 \  
--max-size 10 \  
--managed-capacity-configuration  
ScaleInAfterInactivityMinutes=60,ZeroCapacityStrategy=SCALE_TO_AND_FROM_ZERO
```

If your request is successful, Amazon GameLift Servers returns the updated capacity configuration. Amazon GameLift Servers will then scale-in the selected fleet locations to 0 instances once there has been no game session activity for the configured duration. Following this, when a request for a game session is made in this location, Amazon GameLift Servers will scale-out one instance as quickly as possible. This process will take a little time.

Best practices

Consider the following recommendations when using Scale To/From Zero:

- **Set appropriate inactivity periods** – Balance cost savings against the frequency of scale-in/scale-out cycles. Shorter wait time to scale to zero may maximize savings, but would result in more frequent cold starts.
- **Use with predictable workloads** – Scale To/From Zero works best for games with clear periods of inactivity, such as development/test environments or games with distinct off-peak hours.
- **Monitor CloudWatch metrics** – Track fleet scaling events and game session placement times to optimize your configuration.
- **Combine with scaling policies** – Use Scale To/From Zero alongside target-based or rule-based auto scaling for comprehensive capacity management.

Scale Amazon GameLift Servers container fleets

One of the most challenging tasks with game hosting is scaling capacity to meet player demand without wasting money on resources that you don't need. In a managed container fleet, you scale your fleet capacity by adding or removing fleet instances.

When you create a new fleet, Amazon GameLift Servers sets the fleet's desired capacity to one instance and deploys one instance in the fleet's home region. For a multi-location fleet, Amazon GameLift Servers deploys one instance to the home region and to each remote location. After

the fleet status reaches **ACTIVE**, you can raise the desired capacity to raise or lower the desired capacity to scale down.

You can use Amazon GameLift Servers scaling features to change capacity manually or set up automatic scaling based on player demand:

- Set up automatic scaling with target tracking. See [Target-based auto scaling](#).
- Manually change the capacity of your fleet. See [Manually set capacity for an Amazon GameLift Servers fleet](#).

When scaling a container fleet, consider how adding or removing instances impacts the fleet's capacity to host game sessions and players.

- Game sessions per instance
 - Each game server process running on an instance represents the capacity to host one game session.
 - Use this formula to calculate the number of game sessions that run concurrently on a container fleet instance:

```
[Game sessions per instance] = [# of game server processes per game server container] * [# of game server container groups per instance]
```

If your container architecture runs one game server process concurrently in the game server container, then game sessions per instance equal the number of game server container groups per instance.

- For game server container groups per instance, call [DescribeContainerFleet](#) to get the `GameServerContainerGroupsPerInstance` or `MaximumGameServerContainerGroupsPerInstance` value.
- The number of game server container groups that fit on an instance depends on how much memory is available after Amazon GameLift Servers reserves a portion for the operating system and supporting services. For details on how available memory is calculated and a reference table of common instance types, see [Understand container fleet memory allocation](#).
- Players per instance

- You decide the number of player slots to allow in each game session. Depending on how your hosting solution handles game session placement, you might define players per game session in your matchmaking configuration or in your calls to start a game session placement.
- Use this formula to calculate the number of players that can play your game concurrently on a container fleet instance:

```
[Players per instance] = [# of game sessions per instance] * [# of player slots per game session]
```

To get the current total capacity of a container fleet, call [DescribeFleetCapacity](#) or [DescribeFleetLocation Capacity](#) to get the number of game server container groups in the fleet. Active groups are those that are currently hosting game sessions. Idle groups are ready to host a new game session. Multiply these values by the number of server processes per game server container group.

Monitoring Amazon GameLift Servers

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon GameLift Servers and your other AWS solutions. There are three primary uses for metrics with Amazon GameLift Servers: to monitor system health and set up alarms, to track game server performance and usage, and to manage capacity using manual or auto-scaling.

AWS provides the following monitoring tools to watch Amazon GameLift Servers, report when something is wrong, and take automatic actions when appropriate:

- **Amazon GameLift Servers console** – Use the graphical interface to manage your Amazon GameLift Servers resources and track game hosting activity.
- **Server telemetry metrics** – Amazon GameLift Servers enables you to emit custom metrics directly from your game servers through SDK and plugin integration. You can define and track your own game-specific metrics alongside built-in performance, network, memory, and timing data. All metrics can be published to [Amazon Managed Service for Prometheus](#) and monitored in Amazon Grafana using fully customizable dashboards. You can also publish to Amazon CloudWatch for integration with other AWS services. The telemetry system is fully customizable - you can create custom Prometheus queries to derive additional metrics in addition to the built-in ones. For implementation guides specific to your technology stack, see [the section called “Monitor with server telemetry metrics”](#).
- **Amazon CloudWatch** – You can monitor Amazon GameLift Servers metrics in real time, as well as metrics for other AWS resources and applications that you're running on AWS services. Amazon CloudWatch offers a suite of monitoring features, including tools to create customized dashboards and the ability to set alarms that notify or take action when a metric reaches a specified threshold.
- **AWS CloudTrail** – captures all API calls and related events made by or on behalf of your AWS account for Amazon GameLift Servers and other AWS services. Data is delivered as log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred.
- **Game session logs** – You can output custom server messages for your game sessions to log files that are stored in Amazon S3.

Metrics comparison across monitoring sources

Amazon GameLift Servers provides metrics through three primary sources: the Amazon GameLift Servers console Fleet Activity metrics, server telemetry metrics, and Amazon CloudWatch Amazon GameLift Servers metrics. Each source has unique capabilities. Server telemetry provides the deepest server-side and OS-level visibility, CloudWatch provides the broadest set of GameLift-side fleet, queue, and matchmaking metrics for alarming and automation, and the console surfaces fleet activity at a glance. The tables below show which metrics each source provides, organized by metric category.

Metrics availability by source

In the tables below:

- **Console** indicates the metric is displayed natively in the Amazon GameLift Servers console (for example, on a fleet's Activity, Scaling, or Locations tab, or in the fleets table).
- **CloudWatch** indicates the metric is published to the AWS/GameLift namespace and can be graphed, alarmed on, or queried through CloudWatch.
- **Telemetry** indicates the metric is collected by the Amazon GameLift Servers OpenTelemetry Collector and is available in management portal and the prebuilt dashboards. A ✓* means the metric is not emitted directly but can be derived through a custom PromQL query against the collected telemetry.

Instance metrics

Instance-level capacity and lifecycle metrics:

Instance metrics availability

Metric	Console	CloudWatch	Telemetry
Active instances	✓	✓	✓*
Idle instances	✓	✓	✓*
Percent idle instances	✓	✓	✓*
Desired instances	✓	✓	

Metric	Console	CloudWatch	Telemetry
Min instances	✓	✓	
Max instances	✓	✓	
Pending instances	✓	✓	
Terminating instances	✓	✓	
Instance Spot interruptions	✓	✓	
Recycled instances (Spot)	✓	✓	
Unhealthy instances replaced	✓	✓	

Instance system and OS metrics

Operating-system-level instance metrics. Server telemetry collects a much richer set of host metrics than CloudWatch, including detailed memory, filesystem, and network breakdowns. EC2 fleets emit a different set of host metrics than container fleets – container fleet metrics are listed under *Container fleet metrics* below.

Instance system and OS metrics availability (managed EC2 fleets)

Metric	Console	CloudWatch	Telemetry
CPU utilization	✓	✓	✓
CPU time by state (user, system, idle, iowait)			✓
CPU load average (1m, 5m, 15m)			✓
Memory usage and utilization			✓
Filesystem usage and utilization			✓
Network in/out (bytes)	✓	✓	✓
Network packets, errors, dropped			✓

Metric	Console	CloudWatch	Telemetry
Active network connections			✓
Disk read/write bytes	✓	✓	✓
Disk read/write operations	✓	✓	✓
Disk operation time and I/O time			✓
Pending disk operations			✓
Per-process CPU time			✓
Per-process memory usage (resident, virtual)			✓

Game session and server process metrics

Game session counts and server process lifecycle metrics:

Game session and server process metrics availability

Metric	Console	CloudWatch	Telemetry
Active game sessions	✓	✓	✓*
Activating game sessions	✓	✓	
Available game sessions (Game Capacity)	✓	✓	✓
Percent available game sessions (Capacity Usage)	✓	✓	✓
Concurrent activatable game sessions	✓	✓	
Game session Spot interruptions	✓	✓	
Active server processes	✓	✓	✓
Healthy server processes	✓	✓	✓

Metric	Console	CloudWatch	Telemetry
Percent healthy server processes	✓	✓	✓
Server process activations	✓	✓	
Server process terminations	✓	✓	
Server process abnormal terminations	✓	✓	
Crashed game sessions			✓

Server performance metrics

In-game server performance metrics emitted directly from the game server through the Amazon GameLift Servers SDKs and plugins. These are exclusive to server telemetry and are not available in the console or in CloudWatch.

Server performance metrics availability

Metric	Console	CloudWatch	Telemetry
Server delta time (and p50, p90, p95)			✓
Server tick time (and p50, p90, p95)			✓
Server tick rate			✓
Server world tick time (and p50, p90, p95)			✓
Server up status			✓
Server connections			✓
Server bytes in/out			✓
Server packets in/out			✓
Server packets lost in/out			✓

Player metrics

Player session and concurrent user metrics:

Player metrics availability

Metric	Console	CloudWatch	Telemetry
Current player sessions	✓	✓	
Available player sessions (max)	✓	✓	
Player session activations	✓	✓	
Concurrent users (CCU), global and per location			✓

Container fleet metrics

Metrics specific to managed container fleets. Server telemetry collects a richer set of network and storage breakdowns than CloudWatch, while CloudWatch tracks container group lifecycle counts that are not exposed by telemetry.

Container fleet metrics availability

Metric	Console	CloudWatch	Telemetry
Active game server container groups	✓	✓	
Idle game server container groups	✓	✓	
Pending game server container groups	✓	✓	
Terminating game server container groups	✓	✓	
Unhealthy game server container groups replaced	✓	✓	
Container CPU utilization	✓	✓	✓

Metric	Console	CloudWatch	Telemetry
Container CPU usage by mode (kernel, user, system)			✓
Container CPU reservation	✓	✓	
Container memory utilization	✓	✓	✓
Container memory reservation	✓	✓	✓
Container memory limit and max usage			✓
Container network in/out (rate)	✓	✓	✓
Container network packets in/out			✓
Container network errors and dropped packets			✓
Container storage read/write bytes	✓	✓	✓
ECS task CPU usage (total, system)			✓
ECS task memory utilized and reserved			✓
ECS task network rate (rx, tx)			✓
ECS task storage read/write bytes			✓

Player gateway metrics

Player gateway traffic and throttling metrics, available for managed container fleets that use a player gateway. Player gateway metrics are exclusive to CloudWatch (and visible in the console as CloudWatch widgets).

Player gateway metrics availability

Metric	Console	CloudWatch	Telemetry
Player gateway packets in/out	✓	✓	

Metric	Console	CloudWatch	Telemetry
Player gateway bytes in/out	✓	✓	
Player gateway packets throttled	✓	✓	
Player gateway bytes throttled	✓	✓	
Player gateway player sessions	✓	✓	

Game session queue metrics

Metrics for game session placement queues. Queue metrics are exclusive to CloudWatch (and visible in the console as CloudWatch widgets on the queue's Metrics tab).

Queue metrics availability

Metric	Console	CloudWatch	Telemetry
Average wait time	✓	✓	
Queue depth	✓	✓	
Game sessions placed	✓	✓	
First choice not viable	✓	✓	
First choice out of capacity	✓	✓	
Lowest latency placement		✓	
Lowest price placement		✓	
Placements started	✓	✓	
Placements succeeded	✓	✓	
Placements canceled	✓	✓	
Placements failed	✓	✓	

Metric	Console	CloudWatch	Telemetry
Placements timed out	✓	✓	

FlexMatch matchmaking metrics

Metrics for FlexMatch matchmaking configurations and rule sets. Matchmaking metrics are exclusive to CloudWatch (and visible in the console as CloudWatch widgets on the matchmaking configuration's Metrics tab).

Matchmaking metrics availability

Metric	Console	CloudWatch	Telemetry
Current tickets	✓	✓	
Tickets started	✓	✓	
Tickets failed	✓	✓	
Tickets timed out	✓	✓	
Players started	✓	✓	
Matches created	✓	✓	
Matches accepted	✓	✓	
Matches rejected	✓	✓	
Matches placed	✓	✓	
Match acceptances timed out	✓	✓	
Matchmaking search time	✓	✓	
Time to match	✓	✓	
Time to ticket cancel	✓	✓	
Time to ticket success	✓	✓	

Metric	Console	CloudWatch	Telemetry
Rule evaluations passed		✓	
Rule evaluations failed		✓	

Choosing the right monitoring source

Select your monitoring approach based on your specific needs:

- **Use server telemetry metrics** to monitor comprehensive performance data from your game servers and emit custom metrics specific to your game. These metrics deliver detailed insights into server performance, network activity, memory usage, and timing data for all game engines and server SDKs. You can define custom metrics for gameplay events, business logic performance, and application-specific data points. All dashboards are fully customizable, and you can create custom Prometheus queries to derive additional metrics from the collected data. For more information, see [the section called “Monitor with server telemetry metrics”](#).
- **Use the Amazon GameLift Servers console** for fleet management, capacity planning, and general operational oversight. The console provides an integrated view of fleet health and player activity. For more information, see [the section called “Manage game hosting resources”](#).
- **Use Amazon CloudWatch** for automated monitoring, alerting, and integration with other AWS services. CloudWatch enables custom dashboards and alarm-based automation. For more information, see [the section called “Monitor with CloudWatch”](#).
- **Use [Amazon Managed Service for Prometheus](#)** for high-performance metrics collection and storage with PromQL querying capabilities. Prometheus provides scalable time-series data storage for server telemetry metrics.
- **Use [Amazon Managed Grafana](#)** for advanced visualization and fully customizable dashboarding. Grafana provides pre-built GameLift dashboards that you can customize and extend, plus supports creating entirely custom dashboards with multiple data sources. You can build custom queries and visualizations to track any metrics important to your game.
- **Use multiple sources** for comprehensive monitoring. Combine console oversight, server telemetry metrics, and automated alerting for complete visibility into your game hosting infrastructure.

For additional monitoring capabilities, you can also use:

- [the section called “Logging API calls”](#) – Track API calls and related events for auditing and compliance.
- [the section called “Logging server messages”](#) – Capture custom server messages and game session logs.

Topics

- [the section called “Manage game hosting resources”](#)
- [the section called “Monitor with CloudWatch”](#)
- [the section called “Monitor with server telemetry metrics”](#)
- [the section called “Logging API calls”](#)
- [the section called “Logging server messages”](#)

Monitor Amazon GameLift Servers with Amazon CloudWatch

You can monitor Amazon GameLift Servers using Amazon CloudWatch, an AWS service that collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months to provide a historical perspective on how your game server hosting with Amazon GameLift Servers is performing. You can set alarms that watch for certain thresholds and send notifications or take actions when those thresholds are met. For more information, see the [Amazon CloudWatch User Guide](#).

For enhanced server-side observability with detailed telemetry metrics, Amazon GameLift Servers also supports monitoring through [Amazon Managed Service for Prometheus](#) and [Amazon Managed Grafana](#). These telemetry metrics provide detailed server performance data including:

- **Server Timings:** Server Delta Time (consistency of server tick rate), Server Tick Rate (updates per second), Server Tick Time (processing time per tick), and Server World Tick Time (game world update time)
- **Network Metrics:** Connections (total network connections), Network I/O in Bytes and Packets (data transfer statistics), and Packet Loss (transmission failure percentage)
- **Memory Metrics:** Memory Usage in Units (total memory consumption) and Physical Memory Usage as percentage (memory utilization)
- **CPU Usage:** CPU Usage percentage (resource utilization by game server process)

For implementation guides specific to your SDK or plugin, see [the section called “Monitor with server telemetry metrics”](#).

The following tables list the metrics and dimensions for Amazon GameLift Servers. All metrics that are available in CloudWatch are also available in the Amazon GameLift Servers console, which provides the data as a set of customizable graphs. To access CloudWatch metrics for your games, use the AWS Management Console, the AWS CLI, or the CloudWatch API.

If a metric does not have a location, it uses the home location.

Dimensions for Amazon GameLift Servers metrics

Amazon GameLift Servers supports filtering metrics by the following dimensions.

Dimension	Description
Location	Filter metrics for a fleet deployment location. If a metric does not have a location, it uses the home location.
FleetId	Filter metrics for a single fleet. This dimension can be used with all fleet metrics for instances, server processes, game sessions, and player sessions.
MetricGroups	Filter metrics for a collection of fleets. Add a fleet to a metric group by adding the metric group name to the fleet's attributes (see UpdateFleetAttributes()). This dimension can be used with all fleet metrics for instances, server processes, game sessions, and player sessions.
QueueName	Filter metrics for a single queue. This dimension is used with metrics for game session queues only.
ConfigurationName	Filter metrics for a single matchmaking configuration. This dimension is used with metrics for matchmaking configurations.
ConfigurationName-RuleName	Filter metrics for an intersect of a matchmaking configuration and matchmaking rule. This

Dimension	Description
	dimension is used with metrics for matchmaking rules only.
InstanceType	Filter metrics for an EC2 instance type designation, such as "c4.large". This dimension is used with metrics for spot instances.
OperatingSystem	Filter metrics for an instance's operating system. This dimension is used with metrics for spot instances.
GameServerGroup	Filter FleetIQ metrics for a game server group.
ContainerGroupType	Filter container fleet metrics to distinguish between GameServer ContainerGroup metrics on the fleet location and PerInstance ContainerGroup metrics on the fleet location. This dimension is included for certain container fleet metrics (such as ContainerNetworkIn, ContainerStorageReadBytes) to allow you to distinguish between metrics for different container group types.

Amazon GameLift Servers metrics for fleets

The Amazon GameLift namespace includes the following metrics related to activity across a fleet or a group of fleets. Fleets are used with a managed Amazon GameLift Servers solution. The Amazon GameLift Servers service sends metrics to CloudWatch every minute.

Instances

Metric	Description
ActiveInstances	Instances with ACTIVE status, which means they are running active server processes. The count includes idle instances and those that are hosting one or more game sessions. This metric measures current

Metric	Description
	total instance capacity. This metric can be used with automatic scaling. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum Dimensions: Location
DesiredInstances	Target number of active instances that Amazon GameLift Servers is working to maintain in the fleet. With automatic scaling, this value is determined based on the scaling policies currently in force. Without automatic scaling, this value is set manually. This metric is not available when viewing data for fleet metric groups. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum Dimensions: Location
IdleInstances	Active instances that are currently hosting zero (0) game sessions. This metric measures capacity that is available but unused. This metric can be used with automatic scaling. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location

Metric	Description
MaxInstances	Maximum number of instances that are allowed for the fleet. A fleet's instance maximum determines the capacity ceiling during manual or automatic scaling up. This metric is not available when viewing data for fleet metric groups. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
MinInstances	Minimum number of instances allowed for the fleet. A fleet's instance minimum determines the capacity floor during manual or automatic scaling down. This metric is not available when viewing data for fleet metric groups. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
PercentIdleInstances	Percentage of all active instances that are idle (calculated as $\text{IdleInstances} / \text{ActiveInstances}$). This metric can be used for automatic scaling. Units: Percent Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location

Metric	Description
RecycledInstances	Number of spot instances that have been recycled and replaced. Amazon GameLift Servers recycles spot instances that are not currently hosting game sessions and have a high probability of interruption. This metric is available for managed EC2 fleets only. Units: Count Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: Location
RecycledSpotInstances	Number of spot instances that have been recycled and replaced. Amazon GameLift Servers recycles spot instances that are not currently hosting game sessions and have a high probability of interruption. This metric is available for container fleets only. Units: Count Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: Location
InstanceSpotInterruptions	Number of spot instances that have been interrupted. Units: Count Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: Location

Metric	Description
CPUUtilization	<i>EC2 metric. For Amazon GameLift Servers this metric represents hardware performance across all active instances in a fleet location. The percentage of physical CPU time that Amazon EC2 uses to run the instance, which includes time spent to run both the user code and Amazon EC2 code. Tools in your operating system can show a different percentage than CloudWatch due to factors such as legacy device simulation, configuration of non-legacy devices, interrupt-heavy workloads, live migration, and live update.</i> Units: Percent
NetworkIn	<i>EC2 metric. For Amazon GameLift Servers this metric represents hardware performance across all active instances in a fleet location. The number of bytes received on all network interfaces by the instance. This metric identifies the volume of incoming network traffic to an application on a single instance.</i> Units: Bytes
NetworkOut	<i>EC2 metric. For Amazon GameLift Servers this metric represents hardware performance across all active instances in a fleet location. The number of bytes sent out on all network interfaces by the instance. This metric identifies the volume of outgoing network traffic to an application on a single instance.</i> Units: Bytes

Metric	Description
DiskReadBytes	<i>EC2 metric. For Amazon GameLift Servers this metric represents hardware performance across all active instances in a fleet location.</i> Bytes read from all instance store volumes available to the instance. This metric is used to determine the volume of the data the application reads from the hard disk of the instance. You can use it to determine the speed of the application. Units: Bytes
DiskWriteBytes	<i>EC2 metric. For Amazon GameLift Servers this metric represents hardware performance across all active instances in a fleet location.</i> Bytes written to all instance store volumes available to the instance. This metric is used to determine the volume of the data the application writes onto the hard disk of the instance. You can use it to determine the speed of the application. Units: Bytes
DiskReadOps	<i>EC2 metric. For Amazon GameLift Servers this metric represents hardware performance across all active instances in a fleet location.</i> Completed read operations from all instance store volumes available to the instance in a specified period of time. To calculate the average I/O operations per second (IOPS) for the period, divide the total operations in the period by the number of seconds in that period. Units: Count

Metric	Description
DiskWriteOps	<i>EC2 metric. For Amazon GameLift Servers this metric represents hardware performance across all active instances in a fleet location. Completed write operations to all instance store volumes available to the instance in a specified period of time. To calculate the average I/O operations per second (IOPS) for the period, divide the total operations in the period by the number of seconds in that period.</i> Units: Count

Server processes

Metric	Description
ActiveServerProcesses	Server processes with ACTIVE status, which means they are running and able to host game sessions. The count includes idle server processes and those that are hosting game sessions. This metric measures current total server process capacity. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
HealthyServerProcesses	Active server processes that are reporting healthy. This metric is useful for tracking the overall health of the fleet's game servers. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum

Metric	Description
	Dimensions: Location
PercentHealthyServerProcesses	Percentage of all active server processes that are reporting healthy (calculated as $\text{HealthyServerProcesses} / \text{ActiveServerProcesses}$). Units: Percent Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
ServerProcessAbnormalTerminations	Server processes that were shut down due to abnormal circumstances since the last report. This metric includes terminations that were initiated by the Amazon GameLift Servers service. This occurs when a server process stops responding, consistently reports failed health checks, or does not terminate cleanly (by calling ProcessEnding()). Units: Count Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: Location

Metric	Description
ServerProcessActivations	Server processes that successfully transitioned from ACTIVATING to ACTIVE status since the last report. Server processes cannot host game sessions until they are active. Units: Count Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: Location
ServerProcessTerminations	Server processes that were shut down since the last report. This includes all server processes that transitioned to TERMINATED status for any reason, including normal and abnormal process terminations. Units: Count Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: Location

Game sessions

Metric	Description
ActivatingGameSessions	Game sessions with ACTIVATING status, which means they are in the process of starting up. Game sessions cannot host players until they are active. High numbers for a sustained period of time may indicate that game sessions are not transitioning from ACTIVATING to ACTIVE status. This metric can be used with automatic scaling.

Metric	Description
	Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
ActiveGameSessions	Game sessions with ACTIVE status, which means they are able to host players, and are hosting zero or more players. This metric measures the total number of game sessions currently being hosted. This metric can be used with automatic scaling. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum Dimensions: Location

Metric	Description
AvailableGameSessions	Active, healthy server processes that are not currently being used to host a game session and can start a new game session without a delay to spin up new server processes or instances. This metric can be used with automatic scaling.  Note For fleets that limit concurrent game session activations, use the metric ConcurrentActivatableGameSessions . That metric more accurately represents the number of new game sessions that can start without any type of delay. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location

Metric	Description
ConcurrentActivatableGameSessions	Active, healthy server processes that are not currently being used to host a game session and can immediately start a new game session. This metric differs from <code>AvailableGameSessions</code> in the following way: it does not count server processes that currently cannot activate a new game session because of limits on game session activations. (See the fleet RuntimeConfiguration optional setting <code>MaxConcurrentGameSessionActivations</code>). For fleets that don't limit game session activations, this metric is identical to <code>AvailableGameSessions</code> . Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
PercentAvailableGameSessions	Percentage of game session slots on all active server processes (healthy or unhealthy) that are not currently being used (calculated as $\text{AvailableGameSessions} / [\text{ActiveGameSessions} + \text{AvailableGameSessions} + \text{unhealthy server processes}]$). This metric can be used with automatic scaling. Units: Percent Relevant CloudWatch statistics: Average Dimensions: Location

Metric	Description
GameSessionInterruptions	Number of game sessions on spot instances that have been interrupted. Units: Count Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: Location

Player sessions

Metric	Description
CurrentPlayerSessions	Player sessions with either ACTIVE status (player is connected to an active game session) or RESERVED status (player has been given a slot in a game session but hasn't yet connected). This metric can be used with automatic scaling. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum
PlayerSessionActivations	Player sessions that transitioned from RESERVED status to ACTIVE since the last report. This occurs when a player successfully connects to an active game session. Units: Count Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum

Container fleet metrics

Container fleet metrics provide monitoring data for container-based game hosting, including container group status, resource utilization, and performance metrics.

 Note

Enhanced dashboards for container fleet monitoring are available through [Amazon Managed Grafana](#) when using the telemetry metrics solution. See [the section called “Monitor with server telemetry metrics”](#) for setup instructions.

Metric	Description
ActiveGameServerContainerGroups	Container groups that are currently active and ready to host game sessions. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
IdleGameServerContainerGroups	Container groups that are active but not currently hosting game sessions. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
PendingGameServerContainerGroups	Container groups that are in the process of starting up. Units: Count

Metric	Description
	Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
TerminatingGameServerContainerGroups	Container groups that are in the process of shutting down. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
UnhealthyGameServerContainerGroupsReplaced	Container groups that were replaced due to health check failures. Units: Count Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: Location
ContainerCPUUtilizationPerInstance	Average CPU utilization across all containers on an instance. Units: Percent Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location

Metric	Description
ContainerCPUReservation	Percentage of CPU resources reserved by containers. Units: Percent Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
ContainerMemoryReservation	Percentage of memory resources reserved by containers. Units: Percent Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
ContainerMemoryUtilization	Percentage of reserved memory being used by containers. Units: Percent Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
ContainerNetworkIn	Network bytes received by containers. Units: Bytes Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: Location

Metric	Description
ContainerNetworkOut	Network bytes sent by containers. Units: Bytes Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: Location
ContainerStorageReadBytes	Bytes read from storage by containers. Units: Bytes Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: Location
ContainerStorageWriteBytes	Bytes written to storage by containers. Units: Bytes Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: Location

DDoS protection (player gateway) metrics

Amazon GameLift Servers player gateway metrics track UDP traffic flowing through the DDoS protection layer when `PlayerGatewayMode` is set to `ENABLED` or `REQUIRED`. These metrics help you monitor traffic patterns, identify potential DDoS attacks, and track relay performance.

Metric	Description
PlayerGatewayPacketsIn	Number of UDP packets received from players and forwarded to game servers through the DDoS protection layer.

Metric	Description
	Units: Count Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: FleetId, Location
PlayerGatewayBytesIn	Number of bytes received from players and forwarded to game servers through the DDoS protection layer. Units: Bytes Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: FleetId, Location
PlayerGatewayPacketsOut	Number of UDP packets received from game servers and forwarded to players through the DDoS protection layer. Units: Count Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: FleetId, Location
PlayerGatewayBytesOut	Number of bytes received from game servers and forwarded to players through the DDoS protection layer. Units: Bytes Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: FleetId, Location

Metric	Description
PlayerGatewayPacketsThrottled	Number of inbound packets dropped due to rate limiting by the DDoS protection layer. Units: Count Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: FleetId, Location
PlayerGatewayBytesThrottled	Number of inbound bytes dropped due to rate limiting by the DDoS protection layer. Units: Bytes Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: FleetId, Location
PlayerGatewayPlayerSessions	Player sessions that are currently protected by the DDoS protection layer. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: FleetId, Location

Amazon GameLift Servers metrics for queues

The Amazon GameLift namespace includes the following metrics related to activity across a game session placement queue. Queues are used with a managed Amazon GameLift Servers solution. The Amazon GameLift Servers service sends metrics to CloudWatch every minute.

Metric	Description
AverageWaitTime	Average amount of time that game session placement requests in the queue with status PENDING have been waiting to be fulfilled. Units: Seconds Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum Dimensions: Location
FirstChoiceNotViable	Game sessions that were successfully placed but NOT in the first-choice fleet, because that fleet was considered not viable (such as a spot fleet with a high interruption rate). This metric is based on cost, not latency. The first-choice fleet is either the first fleet listed in the queue or—when a placement request includes player latency data—it is the first fleet chosen by FleetIQ prioritization. If there are no viable spot fleets, any fleet in that region may be selected. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum
FirstChoiceOutOfCapacity	Game sessions that were successfully placed but NOT in the first-choice fleet, because that fleet had no available resources. The first-choice fleet is either the first fleet listed in the queue or—when a placement request includes player latency data — it is the first fleet chosen by your defined FleetIQ prioritization. Units: Count

Metric	Description
	Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum
LowestLatencyPlacement	Game sessions that were successfully placed in a region that offers the queue's lowest possible latency for the players. This metric is emitted only when player latency data is included in the placement request. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum
LowestPricePlacement	Game sessions that were successfully placed in a fleet with the queue's lowest possible price for the chosen region. This fleet can be either a spot fleet or an on-demand instance if the queue has no spot instances. This metric is emitted only when player latency data is included in the placement request. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum
Placement <region name>	Game sessions that are successfully placed in fleets located in the specified region. This metric breaks down the <code>PlacementsSucceeded</code> metric by region. Units: Count Relevant CloudWatch statistics: Sum

Metric	Description
PlacementsCanceled	Game session placement requests that were canceled before timing out since the last report. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum
PlacementsFailed	Game session placement requests that failed for any reason since the last report. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum
PlacementsStarted	New game session placement requests that were added to the queue since the last report. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum
PlacementsSucceeded	Game session placement requests that resulted in a new game session since the last report. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum

Metric	Description
PlacementsTimedOut	Game session placement requests that reached the queue's timeout limit without being fulfilled since the last report. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum
QueueDepth	Number of game session placement requests in the queue with status PENDING. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum Dimensions: Location

Amazon GameLift Servers metrics for matchmaking

The Amazon GameLift namespace includes metrics on FlexMatch activity for matchmaking configurations and matchmaking rules. FlexMatch matchmaking is used with a managed Amazon GameLift Servers solution. The Amazon GameLift Servers service sends metrics to CloudWatch every minute.

For more information on the sequence of matchmaking activity, see [How Amazon GameLift Servers FlexMatch works](#).

Matchmaking configurations

Metric	Description
CurrentTickets	Matchmaking requests currently being processed or waiting to be processed. Units: Count

Metric	Description
	Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum
MatchAcceptancesTimedOut	For matchmaking configurations that require acceptance, the potential matches that timed out during acceptance since the last report. Units: Count Relevant CloudWatch statistics: Sum
MatchesAccepted	For matchmaking configurations that require acceptance, the potential matches that were accepted since the last report. Units: Count Relevant CloudWatch statistics: Sum
MatchesCreated	Potential matches that were created since the last report. Units: Count Relevant CloudWatch statistics: Sum
MatchesPlaced	Matches that were successfully placed into a game session since the last report. Units: Count Relevant CloudWatch statistics: Sum

Metric	Description
MatchesRejected	For matchmaking configurations that require acceptance, the potential matches that were rejected by at least one player since the last report. Units: Count Relevant CloudWatch statistics: Sum
PlayersStarted	Players in matchmaking tickets that were added since the last report. Units: Count Relevant CloudWatch statistics: Sum
TicketsFailed	Matchmaking requests that resulted in failure since the last report. Units: Count Relevant CloudWatch statistics: Sum
TicketsStarted	New matchmaking requests that were created since the last report. Units: Count Relevant CloudWatch statistics: Sum
TicketsTimedOut	Matchmaking requests that reached the timeout limit since the last report. Units: Count Relevant CloudWatch statistics: Sum

Metric	Description
TimeToMatch	For matchmaking requests that were put into a potential match before the last report, the amount of time between ticket creation and potential match creation. Units: Seconds Relevant CloudWatch statistics: Data Samples, Average, Minimum, Maximum
TimeToTicketCancel	For matchmaking requests that were canceled before the last report, the amount of time between ticket creation and cancellation. Units: Seconds Relevant CloudWatch statistics: Data Samples, Average, Minimum, Maximum
TimeToTicketSuccess	For matchmaking requests that succeeded before the last report, the amount of time between ticket creation and successful match placement. Units: Seconds Relevant CloudWatch statistics: Data Samples, Average, Minimum, Maximum

Matchmaking rules

Metric	Description
RuleEvaluationsPassed	Rule evaluations during the matchmaking process that passed since the last report. This metric is limited to the top 50 rules. Units: Count

Metric	Description
	Relevant CloudWatch statistics: Sum
RuleEvaluationsFailed	Rule evaluations during matchmaking that failed since the last report. This metric is limited to the top 50 rules. Units: Count Relevant CloudWatch statistics: Sum

Amazon GameLift Servers metrics for FleetIQ

The Amazon GameLift namespace includes metrics for FleetIQ game server group and game server activity as part of a FleetIQ standalone solution for game hosting. The Amazon GameLift Servers service sends metrics to CloudWatch every minute. Also see [Monitoring your Auto Scaling groups and instances using amazon CloudWatch](#) in the *Amazon EC2 Auto Scaling User Guide*.

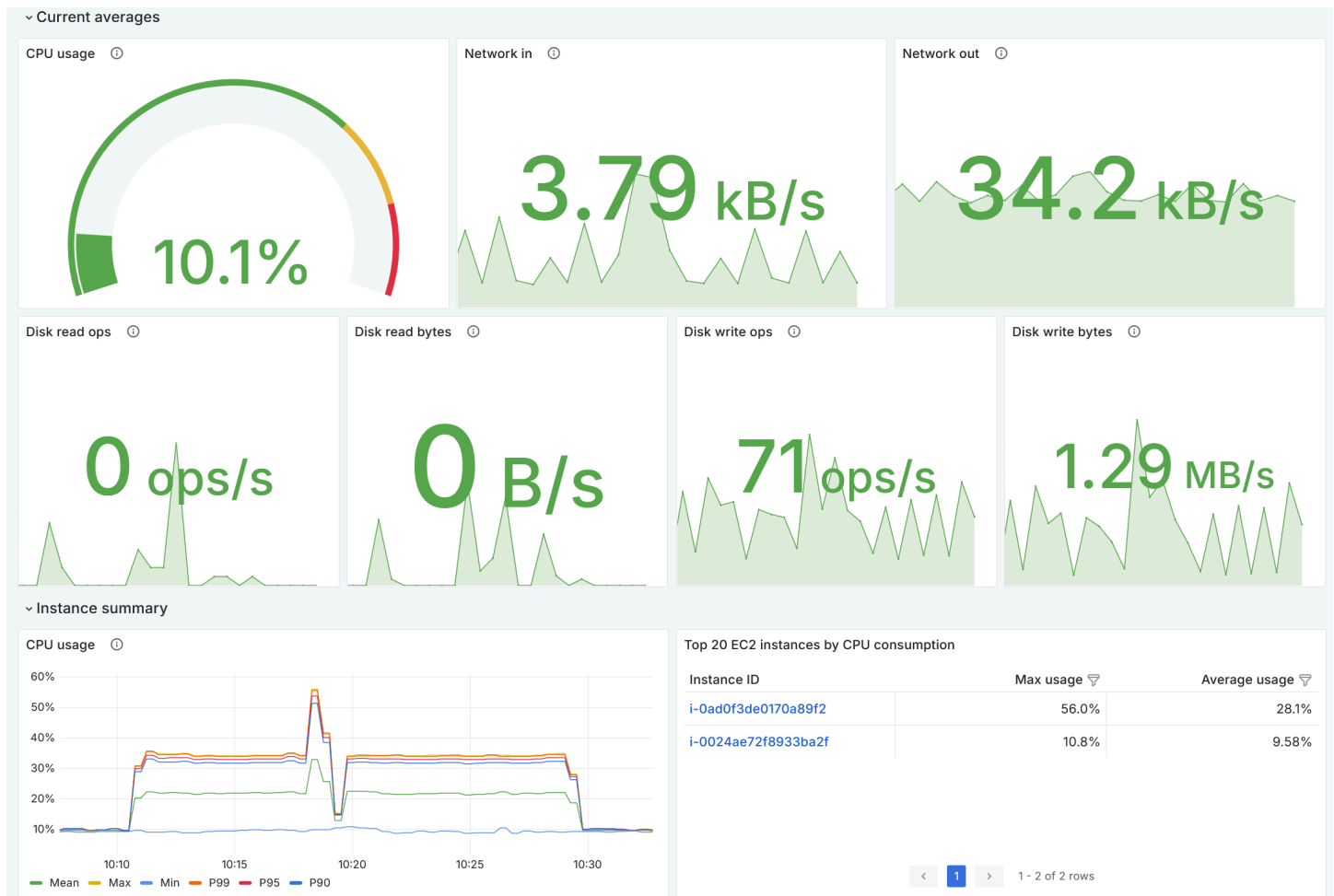
Metric	Description
AvailableGameServers	Game servers that are available to run a game execution and are not currently occupied with gameplay. This number includes game servers that have been claimed but are still in AVAILABLE status. Units: Count Relevant Amazon CloudWatch statistics: Sum Dimensions: GameServerGroup
UtilizedGameServers	Game servers that are currently occupied with gameplay. This number includes game servers that are in UTILIZED status. Units: Count Relevant Amazon CloudWatch statistics: Sum

Metric	Description
DrainingAvailableGameServers	Game servers on instances scheduled for termination that are currently not supporting gameplay. These game servers are the lowest priority to be claimed in response to a new claim request. Units: Count Relevant Amazon CloudWatch statistics: Sum Dimensions: GameServerGroup
DrainingUtilizedGameServers	Game servers on instances scheduled for termination that are currently supporting gameplay. Units: Count Relevant Amazon CloudWatch statistics: Sum Dimensions: GameServerGroup
PercentUtilizedGameServers	Portion of game servers that are currently supporting game executions. This metric indicates the amount of game server capacity that is currently in use. It is useful for driving an Auto Scaling policy that can dynamically add and remove instances to match with player demand. Units: Percent Relevant Amazon CloudWatch statistics: Average, Minimum, Maximum Dimensions: GameServerGroup

Metric	Description
GameServerInterruptions	Game servers on Spot Instances that were interrupted due to limited Spot availability. Units: Count Relevant Amazon CloudWatch statistics: Sum Dimensions: GameServerGroup, InstanceType
InstanceInterruptions	Spot Instances that were interrupted due to limited availability. Units: Count Relevant Amazon CloudWatch statistics: Sum Dimensions: GameServerGroup, InstanceType

Monitor with server telemetry metrics

Amazon GameLift Servers can be configured to collect and publish telemetry metrics for game servers running on managed Amazon EC2 and Container fleets. These metrics become available after deploying the telemetry collector with your server build. The metrics system supports all SDKs (C++, C#, Go), all plugins (Unreal, Unity), and the Amazon GameLift Servers Game Server Wrapper. Metrics data flows to [Amazon Managed Service for Prometheus](#), [the section called "Monitor with CloudWatch"](#), and [Amazon Managed Grafana](#) dashboards (recommended for visualization).



Monitor performance across your game servers using pre-built dashboards in Amazon Managed Grafana or Amazon CloudWatch.

Benefits of telemetry metrics

The telemetry metrics system offers five key benefits:

- **Game engine-specific metrics** — Game engine plugins (Unreal, Unity) provide native integration with engine-specific performance metrics such as server tick time, frame rate, and engine-level resource utilization that are critical for game performance optimization.
- **Custom metrics support** — Define and track your own game-specific metrics using server SDK function calls to monitor custom gameplay events, business logic performance, and application-specific data points that matter to your game.
- **Automated collection** — Metrics flow automatically after telemetry collector deployment with no additional instrumentation required and direct integration with [Amazon Managed Service for Prometheus](#) and Amazon CloudWatch.

- **Multi-level monitoring** — Fleet-level metrics for capacity and scaling, instance-level metrics for resource utilization, and game session metrics for performance tracking.
- **Universal compatibility** — Works with all Amazon GameLift Servers-supported development environments, integrated with all server SDKs, and native support in game engine plugins.

Note

Telemetry metrics are available for Amazon GameLift Servers managed Amazon EC2 or container fleets running Amazon Linux 2023 or Windows.

Before you begin

Required AWS resources

- AWS account configured for Amazon GameLift Servers.
- Managed fleet running on:
 - Amazon EC2 with supported operating systems, OR
 - Containers with Amazon Linux 2023
- Appropriate IAM permissions

IAM requirements

The following IAM permissions are required only if you plan to use the corresponding service:

- [Amazon Managed Service for Prometheus](#) (required only if publishing metrics to Prometheus)
 - `aps:RemoteWrite` permission
 - Access to your Prometheus workspace
- **Amazon CloudWatch** (required only if publishing metrics to Amazon CloudWatch)
 - `cloudwatch:PutMetricData` permission
 - Access to metrics namespaces
- [Amazon Managed Grafana](#) (required only if using Grafana dashboards)
 - `grafana:Read` permission
 - SSO configuration for dashboard access

Implementation

Select your implementation path based on your development environment:

SDK implementation

SDK Type	SDK Setup	Custom Metrics	API Reference
Go SDK	Complete Setup Guide	Go Metrics API	Go Actions and Data Types
C# SDK	Complete Setup Guide	C# Metrics API	C# Actions and Data Types
C++ SDK	Complete Setup Guide	C++ Metrics API	C++ Actions and Data Types

Plugin implementation

Plugin	Plugin Setup	Custom Metrics	API Reference
Unreal	Complete Setup Guide	Unreal Metrics API	Unreal Actions and Data Types
Unity	Complete Setup Guide	Unity Metrics API	C# Actions and Data Types

Implementation workflow

Each implementation follows a two-step process:

1. **Complete Setup Guide (METRICS.md)** — Infrastructure deployment, AWS infrastructure configuration, fleet setup, and Grafana dashboard configuration.
2. **API Implementation Guide (CUSTOM_METRICS.md)** — Language-specific SDK usage, metric types, custom metrics creation, and advanced configuration.

Verification

1. Validate metrics flow by checking your [Amazon Managed Service for Prometheus](#) workspace or Amazon CloudWatch console for incoming telemetry data.
2. Check dashboard visibility in [Amazon Managed Grafana](#) using the pre-built dashboards.
3. Test custom metrics by verifying they appear in your monitoring dashboards.

Note

After completing implementation, return to this page and go to the [Available metrics](#) page.

Available metrics

Metrics fall into three categories:

- Automatically collected metrics
- SDK-provided metrics
- Custom metrics

Automatic metrics collection

No code changes required for these metrics:

Instance metrics

Metric Type	Description	Use Case
CPU	Percentage utilization per instance	Resource monitoring
Memory	Physical memory usage and percentage	Capacity planning
Network I/O	Bytes and packets sent/received	Connection health
Disk I/O	Read/write operations and throughput	Storage performance

Fleet metrics

Metric Type	Description	Use Case
Active Instances	Running instances count	Fleet scaling
Game Sessions	Active and available sessions	Capacity management
Crashed game sessions	Game sessions that have crashed	Error monitoring

SDK-provided metrics

Requires SDK function calls in your code:

Server timing metrics

Metric	Description	Implementation
Server Delta Time	Difference in time between the current server tick and the previous server tick. Measures the consistency of the server's tick rate	Call <code>GetDeltaTime()</code>
Server Tick Rate	Shows the number of times per second the server is processing updates	Automatically calculated
Server Tick Time	The amount of time it takes for the server to process a single tick or update	Call <code>GetTickTime()</code>
Server World Tick Time	The amount of time it takes for the server to update the game world with each tick	Call <code>GetWorldUpdateTime()</code>

Implementation: For engine-agnostic SDKs (C++, C#, Go), you implement these metrics by calling SDK functions from your game loop with calculated timing values. For engine plugins (Unreal, Unity), these metrics are captured automatically through engine integration.

Network metrics

Metric	Description	Implementation
Connections	The total number of network connections the server has established	Automatic after InitMetrics()
Network I/O (Bytes)	The total number of bytes being sent and received by the server over the network	Automatic after InitMetrics()
Network I/O (Packets)	The total number of network packets being sent and received by the server	Automatic after InitMetrics()
Packet Loss	The percentage of network packets that are being lost during transmission	Automatic after InitMetrics()

Implementation: Integrate SDK function calls with your networking library. The SDK provides guidance for different network implementations.

Process metrics

Metric	Description	Implementation
CPU Usage (%)	The percentage of CPU resources being utilized by the game server process	Automatic after InitMetrics()
Memory Usage (Units)	The total amount of memory being consumed by the server processes	Automatic after InitMetrics()
Physical Memory Usage (%)	The percentage of the server's total physical memory that is currently being utilized	Automatic after InitMetrics()
Server Status	Game server health state	Automatic after InitMetrics()

Implementation: These metrics are automatically collected by the SDK for each game session process.

Per-process dashboard organization

Per-process metrics are available in two specialized dashboards:

- **Server Performance dashboard** — Contains server timings (delta time, tick rate, tick time, world tick time), network metrics (connections, I/O bytes/packets, packet loss), memory usage, and CPU usage for individual game sessions.
- **Instance Performance dashboard** — Features "Top N Memory Consuming Game Sessions" and "Top N CPU Consuming Game Sessions" tables that help identify which processes contribute most to instance resource consumption. Clicking on Game Session links enables deeper investigation of detailed metrics.

Per-process metrics use cases

The per-process/per-game-session metrics support the following monitoring scenarios:

- **Dive deep performance investigation** — When a host/instance has degraded performance due to specific processes or game sessions, per-process metrics help identify which process caused the issue through Top CPU and Memory consuming Game Sessions tables.
- **Game server crash investigation** — When a game session crashes, these metrics help determine if the crash was due to out of memory, CPU overload, or network bandwidth problems.
- **Investigate player reported issues** — When players report lag or interruptions during gameplay, per-process metrics help identify bottlenecks in CPU, memory, network, tick time, or world update time.
- **Identify performance changes in different builds** — Tick time, tick rate, and world update time metrics allow developers to measure how game performance changes across different server builds.
- **Detect delays and slowness in gameplay** — Tick time, tick rate, and world update time metrics reflect how fast the server updates the game, directly impacting customer experience.
- **Benchmarking** — Identify how different game scenarios affect server performance based on factors like player count, game mode, and other variables.

Dashboard organization

Metrics are organized into specialized dashboards in [Amazon Managed Grafana](#) for different monitoring scenarios. The available dashboards depend on your fleet type:

EC2 Fleet dashboards

- **EC2 Fleet Overview dashboard** — High-level fleet capacity, scaling insights, concurrent players (CCU), instances, player capacity, and crashed game sessions.
- **Instances Overview dashboard** — Aggregated host-level metrics across all instances including average CPU, memory, network, and disk utilization.
- **Instance Performance dashboard** — Detailed metrics for individual instances with "Top N Memory Consuming Game Sessions" and "Top N CPU Consuming Game Sessions" tables for identifying resource-intensive processes.
- **Server Performance dashboard (EC2)** — Game loop timing, network performance, memory, and CPU metrics for individual game sessions on EC2 instances.

Container Fleet dashboards

- **Container Fleet Overview dashboard** — High-level overview of container fleet resource utilization including CPU reservation, memory utilization, and container group status.
- **Container Performance dashboard** — Detailed metrics for individual containers within specific ECS tasks including CPU utilization, memory usage, network I/O, and storage performance.
- **Server Performance dashboard (Container)** — Game loop timing, network performance, memory, and CPU metrics for individual game sessions in containers.

For detailed dashboard information and usage instructions, see [the section called “Dashboard organization and usage”](#).

How it works

The telemetry metrics system follows a simple four-stage data flow from your game servers to visualization dashboards.

Collection: Your game server, integrated with the GameLift Server SDK or plugin, automatically emits metrics to a local telemetry collector running on the same instance. The SDK captures both automatic metrics (server lifecycle, resource usage) and custom metrics you define in your code.

Processing: The telemetry collector aggregates metrics from your game server and combines them with instance-level performance data (CPU, memory, network, disk usage). This provides a complete picture of both your game's performance and the underlying infrastructure.

Storage: Processed metrics are exported to your choice of metrics warehouse - [Amazon Managed Service for Prometheus](#) for high-performance time-series storage, Amazon CloudWatch for AWS service integration, or both. All data transmission is authenticated and encrypted.

Visualization: [Amazon Managed Grafana](#) connects to your metrics warehouse to display pre-built GameLift dashboards. These dashboards provide fleet overviews, server performance details, and container monitoring views that help you monitor and troubleshoot your game hosting infrastructure.

 Note

All metric transmission between your game server and the telemetry collector occurs locally on the instance for security. Only the collector communicates with AWS services using proper authentication.

Dashboard organization and usage

View your metrics on comprehensive dashboards in [Amazon Managed Grafana](#). The available dashboards depend on your fleet type:

Dashboard availability by fleet type

The following table shows which dashboards are available for each fleet type:

Dashboard	Fleet Type	Description
EC2 Fleet Overview	EC2 Fleet	Displays information on concurrent players (CCU), instances and player capacity
Instances Overview	EC2 Fleet	Displays average CPU, memory, and network utilization across all fleet instances
Instance Performance	EC2 Fleet	Displays detailed metrics (CPU, memory, disk, network) for an individual instance

Dashboard	Fleet Type	Description
Container Fleet Overview	Container Fleet	Displays average resource utilization of all containers in a managed container fleet
Container Performance	Container Fleet	Displays detailed metrics of individual containers within a specific ECS task
Server Performance	Both	Displays the network, memory and runtime performance of a specified game server process (separate versions for EC2 and Container fleets)

Managed EC2 Fleets:

- EC2 Fleet Overview provides high-level fleet capacity and scaling insights.
- Use Instances Overview and Instance Performance dashboards for host-level monitoring.
- Metrics collected via hostmetrics receiver for system-level visibility.
- Focus on EC2 instance resource utilization and performance.
- Server Performance (EC2) monitors game server application metrics independent of underlying infrastructure.

Managed Container Fleets:

- Use Container Fleet Overview and Container Performance dashboards for ECS task and container-level monitoring.
- Metrics collected via ECS Container Receiver for containerized workload visibility.
- Focus on task-level aggregation and container resource isolation.
- Server Performance (Container) monitors game server application metrics independent of underlying infrastructure.

EC2 Fleet Overview dashboard

This dashboard provides a high-level overview of your fleet's utilization and capacity globally and by location. It features graphs showing counts for game server stops, starts, and crashes, as well as the percentage of healthy game servers. You can filter by FleetID and Location.

Fleet Overview metrics

The following table shows the metrics available on the Fleet Overview dashboard:

Dashboard	Metric name	Definition
Global CCU and Capacity	CCU	Shows the number of concurrent users in all the game servers over all the instances in all locations
	Global CCU	Shows the number of concurrent users in all the game servers over all the instances globally
	Active Instances	Shows the total number of instances in the fleet that are active
	Active Processes	Shows the number of active game server processes that are ready to host a game session
	Game Server Starts	Shows the number of game sessions that started across the fleet
Global CCU and Capacity	Healthy Game Servers	Shows the average percentage of game servers that report healthy to Amazon GameLift Servers across the fleet
	Crashed Game Sessions	Shows the game session IDs of the crashed game sessions. Click the link of game session ARN to navigate to the Server Performance dashboard

Dashboard	Metric name	Definition
Location CCU	Location Players	Shows the number of concurrent players in a location, including all the game servers over all the instances in the location
	Location Capacity	Shows capacity utilization (%) in a location, and the percentage of game servers in use in the location

Note

CCU metrics require implementation in your game server code. These metrics are not automatically collected and must be implemented and reported by your application.

Instances Overview dashboard

This dashboard provides aggregated host-level metrics across all instances in your fleet. Current averages show overall health of the instances. When performance degrades, check CPU usage, memory consumption, network and disk consumption for bottlenecks. You can filter by FleetID and Location.

Instances Overview metrics

The following table shows the metrics available on the Instances Overview dashboard:

Dashboard	Metric name	Definition
Instance Summary	Average CPU Usage	Instance summary tracks average resource usage by instances. Average percentage of CPU resources being used across all fleet instances. High utilization patterns require checking individual instance metrics and resource distribution
	Peak CPU Usage	Highest CPU utilization recorded across any instance in the fleet during the time period

Dashboard	Metric name	Definition
	CPU Usage by Instance	Individual CPU utilization for each instance in the fleet, enabling identification of resource hotspots
Instance Summary	Average Memory Usage	Average percentage of RAM being used across all fleet instances. High utilization patterns require checking individual instance metrics and resource distribution like memory consumption
	Peak Memory Usage	Highest memory utilization recorded across any instance in the fleet during the time period
	Memory Usage by Instance	Individual memory utilization for each instance in the fleet, helping identify memory-intensive workloads
Network Summary	Network I/O (Bytes)	Network summary reflects average instance connectivity. Average network traffic volume (sent and received) across all fleet instances . For latency or connectivity issues, check individual instance metrics like Network I/O and Packet rates
	Network I/O (Packets)	Average network packet rate (sent and received) across all fleet instances
	Network Usage by Instance	Individual network utilization for each instance, useful for identifying network bottlenecks

Dashboard	Metric name	Definition
Disk Summary	Disk I/O Operations	Disk summary indicates average disk performance of the instances. Average disk read/write operations across all fleet instances . Slow response times suggest examining Disk operations, I/O wait times, and Pending operations at the individual instance level
	Disk I/O Throughput	Average disk read/write throughput across all fleet instances
	Disk Usage by Instance	Individual disk utilization for each instance, helping identify storage performance issues

Note

Instance-level metrics are collected via the hostmetrics receiver and provide system-level visibility into your fleet's infrastructure performance. Use this dashboard to identify overall fleet health trends and drill down to individual instances when performance issues are detected.

Instance Performance dashboard

This dashboard provides detailed performance metrics for individual instances. Current averages show overall instance health. When performance degrades, check CPU usage, memory consumption, and file system consumption for bottlenecks. It features "Top N Memory Consuming Game Sessions" and "Top N CPU Consuming Game Sessions" tables that help identify which processes contribute the most to instance resource consumption. Clicking on Game Session links enables deeper investigation of detailed metrics. You can filter by specific Instance ID.

Instance Performance metrics

The following table shows the metrics available on the Instance Performance dashboard:

Dashboard	Metric name	Definition
Load Summary	Top N Memory Consuming Game Sessions	Load summary tracks resource usage by game servers. Ranked list of game sessions consuming the most memory resources on this instance. High utilization patterns require checking individual session metrics and resource distribution. Clicking on Game Session links enables deeper investigation of detailed metrics
	Top N CPU Consuming Game Sessions	Ranked list of game sessions consuming the most CPU resources on this instance. High utilization patterns require checking individual session metrics and resource distribution like CPU and memory. Clicking on Game Session links enables deeper investigation of detailed metrics
	CPU Usage per Game Session	Individual CPU utilization breakdown showing resource consumption by each active game session
	Memory Usage per Game Session	Individual memory utilization breakdown showing resource consumption by each active game session
Current Averages	Instance CPU Usage	Overall CPU utilization for the selected instance over time
	Instance Memory Usage	Overall memory utilization for the selected instance over time
	Instance File System Usage	File system consumption for the selected instance, useful for identifying storage capacity issues

Dashboard	Metric name	Definition
Network	Instance Network I/O	Network reflects instance connectivity. Network traffic volume and packet rates for the selected instance. For latency or connectivity issues, investigate Network I/O, Packet rates, and Error counts
	Network Connections	Number of active network connections on the selected instance
	Network Error Counts	Network error statistics for identifying connectivity problems
Disk	Disk I/O Operations	Disk indicates disk performance. Disk read/write activity and utilization for the selected instance. Slow response times suggest examining Disk operations, I/O wait times, and Pending operations
	Disk I/O Wait Times	Time spent waiting for disk operations to complete
	Pending Disk Operations	Number of disk operations waiting to be processed

Note

The Top N Memory and CPU Consuming Game Sessions tables are essential for identifying performance bottlenecks and resource-intensive processes that may impact overall instance performance. These rankings enable quick identification of problematic game sessions for further investigation.

Container Fleet Overview dashboard

This dashboard provides a high-level overview of your container fleet's resource utilization and capacity. It displays average resource utilization of all containers in a managed container fleet,

including CPU reservation, memory utilization, and container group status. You can filter by FleetID and Location.

Container Fleet Overview metrics

The following table shows the metrics available on the Container Fleet Overview dashboard:

Dashboard	Metric name	Definition
Container Group Status	Active Game Server Container Groups	Container groups that are currently active and ready to host game sessions
	Idle Game Server Container Groups	Container groups that are active but not currently hosting game sessions
	Pending Game Server Container Groups	Container groups that are in the process of starting up
	Terminating Game Server Container Groups	Container groups that are in the process of shutting down
Resource Utilization	Container CPU Utilization	Average CPU utilization across all containers in the fleet
	Container Memory Utilization	Average memory utilization across all containers in the fleet
	Container CPU Reservation	Percentage of CPU resources reserved by containers across the fleet
Network Activity	Container Network In	Average network bytes received by containers across the fleet
	Container Network Out	Average network bytes sent by containers across the fleet

Note

Container fleet metrics are collected via ECS Container Receiver and provide containerized workload visibility with focus on task-level aggregation and container resource isolation.

Container Performance dashboard

This dashboard provides detailed performance metrics for individual containers within specific ECS tasks. It displays detailed metrics of individual containers including CPU utilization, memory usage, network I/O, and storage performance. You can filter by specific Container ID or ECS Task.

Container Performance metrics

The following table shows the metrics available on the Container Performance dashboard:

Dashboard	Metric name	Definition
Resource Performance	Container CPU Utilization	CPU utilization for the selected container over time
	Container Memory Utilization	Memory utilization for the selected container over time
	Container Memory Reservation	Percentage of memory resources reserved by the selected container
Network Performance	Container Network I/O	Network traffic volume (bytes in/out) for the selected container
	Network Connections	Number of active network connections for the selected container
Storage Performance	Container Storage Read/Write	Storage read and write activity for the selected container
	Container Storage I/O Operations	Number of storage I/O operations performed by the selected container

 **Note**

Container performance metrics provide detailed visibility into individual container resource consumption and performance characteristics within ECS tasks.

Server Performance dashboard

The Server Performance dashboard shows metrics related to server timings, network activity, memory, and CPU usage for individual game sessions. You can filter by Game Session ID and export metrics directly to Amazon CloudWatch or [Amazon Managed Grafana](#).

Server Performance metrics

The following table shows the metrics available on the Server Performance dashboard:

Dashboard	Metric name	Definition
Server Timings	Server Delta Time	Difference in time between the current server tick and the previous server tick. It's a measure of the consistency of the server's tick rate
	Server Tick Rate	Shows the number of times per second the server is processing updates
	Server Tick Time	The amount of time it takes for the server to process a single tick or update
	Server World Tick Time	The amount of time it takes for the server to update the game world with each tick
Network	Connections	The total number of network connections the server has established
	Network I/O (Bytes)	The total number of bytes being sent and received by the server over the network
	Network I/O (Packets)	The total number of network packets being sent and received by the server

Dashboard	Metric name	Definition
Memory	Packet Loss	The percentage of network packets that are being lost during transmission
	Memory Usage (Units)	The total amount of memory being consumed by the server processes
	Physical Memory Usage (%)	The percentage of the server's total physical memory that is currently being utilized
CPU Usage	CPU Usage (%)	The percentage of CPU resources being utilized by the game server process

Common monitoring scenarios

Dive deep performance investigation

Scenario: A host/instance is having degraded performance due to specific processes or game sessions

Investigation steps:

- Access the Instance Performance dashboard.
- Review "Top N Memory Consuming Game Sessions" table to identify which processes contribute the most to instance memory consumption.
- Review "Top N CPU Consuming Game Sessions" table to identify which processes contribute the most to instance CPU utilization.
- Click on Game Session links to enable deeper investigation of detailed metrics.
- Analyze server timings (Server Delta Time, Server Tick Rate, Server Tick Time, Server World Tick Time) to identify performance bottlenecks.

Game server crash investigation

Scenario: A game session has crashed and you need to determine the root cause

Investigation steps:

- Access the Server Performance dashboard for the crashed game session.
- Check Memory Usage (Units) and Physical Memory Usage (%) to determine if the crash was due to out of memory.
- Review CPU Usage (%) to identify if CPU overload caused the crash.
- Analyze Network I/O (Bytes) and Network I/O (Packets) to determine if network bandwidth problems contributed to the crash.
- Examine Packet Loss percentage to identify network-related issues.

Investigate player reported issues

Scenario: Players report lag or interruption during gameplay

Investigation steps:

- Access the Server Performance dashboard for the affected game session.
- Review Server Tick Time and Server World Tick Time to identify delays in game updates.
- Check Server Tick Rate to ensure consistent server update frequency.
- Analyze CPU Usage (%) to identify processing bottlenecks.
- Review Memory Usage metrics to identify memory-related performance issues.
- Check Network I/O metrics and Packet Loss to identify network bottlenecks.

Identify performance changes in different game server builds

Scenario: You want to measure how game performance changes across different server builds

Investigation steps:

- Compare Server Tick Time metrics between different builds to measure processing efficiency changes.
- Analyze Server Tick Rate consistency across builds to identify performance regressions.
- Review Server World Tick Time to measure game world update performance changes.
- Compare Memory Usage patterns between builds to identify memory optimization improvements or regressions.
- Monitor CPU Usage trends to assess computational efficiency changes.

Detect delays and slowness in gameplay

Scenario: You need to monitor server responsiveness and game update speed

Investigation steps:

- Monitor Server Tick Time to measure how fast the server processes each update cycle.
- Track Server Tick Rate to ensure consistent game state updates per second.
- Analyze Server World Tick Time to measure game world update speed, which directly impacts customer experience.
- Set up alerts for Server Delta Time variations to detect inconsistent server performance.

Benchmarking different game scenarios

Scenario: You want to identify how different game scenarios affect server performance

Investigation steps:

- Compare server performance metrics across different player counts to understand scaling impact.
- Analyze performance differences between game modes using Server Tick Time and CPU Usage metrics.
- Monitor Memory Usage patterns across different game scenarios to identify resource-intensive features.
- Track Network I/O metrics to understand bandwidth requirements for different gameplay scenarios.
- Use the Instance Performance dashboard to identify which game scenarios produce the highest resource-consuming game sessions.

High resource utilization response

Scenario: Unusual resource spikes (CPU >85%, Memory >90%)

Investigation steps:

Identify affected resources

- Use DescribeGameSessionDetails API.

- Filter by Status if needed.
- Document affected instances.

Analyze resource usage

- Review Instance Overview dashboard.
- Compare utilization across fleet.
- Check historical patterns.

Monitor game server impact

- Check Server Performance metrics.
- Review tick times and packet loss.
- Monitor memory leaks.

Resolution steps

- Download session logs.
- Address build issues.
- Monitor improvements.

Game server crash analysis

Scenario: Multiple error-status game sessions across fleet

Investigation steps:

Initial assessment

- Access Fleet Overview dashboard.
- Review crashed session table.
- Note patterns in timing/location.

Performance analysis

- Check server timing metrics.

- Review resource utilization.
- Monitor network performance.

Infrastructure review

- Verify fleet capacity.
- Check instance health.
- Review scaling policies.

Resolution path

- Analyze server logs.
- Review code optimization.
- Implement fixes.

Fleet capacity optimization

Scenario: Game launch or benchmark study

Analysis steps:

Resource utilization

- Filter by location.
- Review P50/P95/P99 metrics.
- Analyze usage patterns.

Instance type analysis

- Compare performance by type.
- Identify scaling candidates.
- Document utilization patterns.

Optimization actions

- Adjust scaling policies.

- Modify instance types.
- Update fleet configuration.

Troubleshooting guide

Common issues and resolution steps

Missing or incomplete metrics

Symptoms

- No metrics appearing in dashboards.
- Partial metric collection.
- Delayed metric updates.

Resolution steps

A. Verify collector status

Check systemd service:

```
sudo systemctl status gamelift-telemetry-collector
```

Review collector logs:

```
sudo journalctl -u gamelift-telemetry-collector
```

- Confirm collector configuration.

B. IAM permission verification

- Check instance role permissions.
- Verify required policies:
 - `aps:RemoteWrite`
 - `cloudwatch:PutMetricData`
- Validate role trust relationships.

C. Network connectivity

- Verify endpoint access.
- Check security group rules.
- Review network ACLs.

Authentication errors

Symptoms

- SigV4 authentication failures.
- Access denied messages.
- Credential refresh issues.

Resolution steps

A. SigV4 authentication

- Verify temporary credentials.
- Check credential rotation.
- Validate instance profile.

B. AMP access

- Review workspace configuration.
- Verify remote write URL.
- Check IAM role bindings.

Dashboard issues

Symptoms

- Empty dashboards.
- Missing data points.
- Authentication failures.

Resolution steps

A. Data source configuration

- Verify Prometheus connection.
- Check Amazon CloudWatch integration.
- Test data source permissions.

B. Grafana access

- Confirm SSO configuration.
- Verify 2FA setup if required.
- Check user permissions.

Windows-specific issues

Symptoms

- Service startup failures.
- Metric collection gaps.
- Permission errors.

Resolution steps

- Verify Windows service status.
- Check Windows Event Logs.
- Review collector configuration.
- Validate Windows-specific paths.

Logging Amazon GameLift Servers API calls with AWS CloudTrail

Amazon GameLift Servers is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon GameLift Servers. CloudTrail captures all API calls for Amazon GameLift Servers as events. The calls captured include calls from the Amazon

GameLift Servers console and code calls to the Amazon GameLift Servers API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon GameLift Servers. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon GameLift Servers, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

Amazon GameLift Servers information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon GameLift Servers, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing events with CloudTrail Event history](#).

For an ongoing record of events in your AWS account, including events for Amazon GameLift Servers, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- [Overview for creating a trail](#)
- [CloudTrail supported services and integrations](#)
- [Configuring Amazon SNS notifications for CloudTrail](#)
- [Receiving CloudTrail log files from multiple regions](#) and [Receiving CloudTrail log files from multiple accounts](#)

All Amazon GameLift Servers actions are logged by CloudTrail and are documented in the [Amazon GameLift Servers API Reference](#). For example, calls to `CreateGameSession`, `CreatePlayerSession` and `UpdateGameSession` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail `userIdentity` element](#).

Understanding Amazon GameLift Servers log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateFleet` and `DescribeFleetAttributes` actions.

```
{
  "Records": [
    {
      "eventVersion": "1.04",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/myUserName",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "myUserName"
      },
      "eventTime": "2015-12-29T23:40:15Z",
      "eventSource": "gamelift.amazonaws.com",
      "eventName": "CreateFleet",
      "awsRegion": "us-west-2",
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "[]",
      "requestParameters": {
        "buildId": "build-92b6e8af-37a2-4c10-93bd-4698ea23de8d",
        "eC2InboundPermissions": [
          {
            "ipRange": "10.24.34.0/23",
```

```

        "fromPort": 1935,
        "protocol": "TCP",
        "toPort": 1935
    },
    ],
    "logPaths": [
        "C:\\game\\serverErr.log",
        "C:\\game\\serverOut.log"
    ],
    "eC2InstanceType": "c5.large",
    "serverLaunchPath": "C:\\game\\MyServer.exe",
    "description": "Test fleet",
    "serverLaunchParameters": "-paramX=baz",
    "name": "My_Test_Server_Fleet"
},
"responseElements": {
    "fleetAttributes": {
        "fleetId": "fleet-0bb84136-4f69-4bb2-bfec-a9b9a7c3d52e",
        "serverLaunchPath": "C:\\game\\MyServer.exe",
        "status": "NEW",
        "logPaths": [
            "C:\\game\\serverErr.log",
            "C:\\game\\serverOut.log"
        ],
        "description": "Test fleet",
        "serverLaunchParameters": "-paramX=baz",
        "creationTime": "Dec 29, 2015 11:40:14 PM",
        "name": "My_Test_Server_Fleet",
        "buildId": "build-92b6e8af-37a2-4c10-93bd-4698ea23de8d"
    },
    },
    "requestID": "824a2a4b-ae85-11e5-a8d6-61d5cafb25f2",
    "eventID": "c8fbea01-fbf9-4c4e-a0fe-ad7dc205ce11",
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
},
{
    "eventVersion": "1.04",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/myUserName",
        "accountId": "111122223333",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",

```

```
        "userName": "myUserName"
    },
    "eventTime": "2015-12-29T23:40:15Z",
    "eventSource": "gamelift.amazonaws.com",
    "eventName": "DescribeFleetAttributes",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.0",
    "userAgent": "[]",
    "requestParameters": {
        "fleetIds": [
            "fleet-0bb84136-4f69-4bb2-bfec-a9b9a7c3d52e"
        ]
    },
    "responseElements": null,
    "requestID": "82e7f0ec-ae85-11e5-a8d6-61d5cafb25f2",
    "eventID": "11daabcb-0094-49f2-8b3d-3a63c8bad86f",
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
},
]
```

Logging server messages in Amazon GameLift Servers

You can capture custom server messages from your Amazon GameLift Servers servers in log files. The way you configure logging depends on whether you use custom servers or Amazon GameLift Servers Realtime, and whether you're using managed EC2 fleets or container fleets (see the appropriate subsections in this chapter).

Topics

- [Logging differences: Managed EC2 vs Container fleets](#)
- [Logging server messages \(custom servers\)](#)

Logging differences: Managed EC2 vs Container fleets

Logging behavior differs significantly between managed EC2 fleets and container fleets. Understanding these differences is crucial when migrating from EC2 to containers or choosing the right fleet type for your logging requirements.

Key differences overview

Feature	Managed EC2 Fleets	Container Fleets
Log storage	Automatically uploaded to Amazon S3 after game session ends	Must configure Amazon CloudWatch Logs or custom logging solution
GetGameSessionLogUrl API	Available - returns Amazon S3 URL for log download	Not available - logs not automatically stored in Amazon S3
Log retention	14 days in Amazon S3 (automatic)	Depends on your logging configuration
Real-time monitoring	Limited - logs available only after game session ends	Available with Amazon CloudWatch Logs integration
Setup complexity	Automatic - no additional configuration required	Requires explicit logging configuration

Managed EC2 fleet logging

For managed EC2 fleets, Amazon GameLift Servers provides automatic log management:

- **Automatic upload:** Server logs are automatically uploaded to Amazon S3 when a game session ends
- **GetGameSessionLogUrl API:** Use this API to retrieve a pre-signed URL for downloading logs from Amazon S3
- **14-day retention:** Logs are retained in Amazon S3 for 14 days before automatic deletion
- **Size limits:** Log files have size limits per game session (see [Amazon GameLift Servers endpoints and quotas](#))

For more information, see [the section called “Logging for custom servers”](#) and the [GetGameSessionLogUrl API reference](#).

Container fleet logging

Container fleets require you to configure logging explicitly:

- **No automatic Amazon S3 upload:** Logs are not automatically uploaded to Amazon S3
- **GetGameSessionLogUrl not available:** This API does not work with container fleets
- **Amazon CloudWatch Logs integration:** Configure your container to send logs to Amazon CloudWatch Logs for centralized logging
- **Custom logging solutions:** Implement your own logging infrastructure using log drivers or sidecar containers
- **Real-time access:** With proper configuration, logs can be accessed in real-time during game sessions

For detailed container logging options, see [How container fleets work](#).

Migration considerations

When migrating from managed EC2 to container fleets, consider these logging changes:

- **Update log retrieval code:** Replace GetGameSessionLogUrl API calls with Amazon CloudWatch Logs queries or your custom logging solution
- **Configure log retention:** Set up appropriate retention policies in Amazon CloudWatch Logs or your logging system
- **Implement real-time monitoring:** Take advantage of real-time log access for better observability
- **Review log volume and costs:** Amazon CloudWatch Logs pricing differs from the included Amazon S3 storage in managed EC2 fleets

Recommended container logging setup

For container fleets, we recommend:

1. **Amazon CloudWatch Logs integration:** Configure your container definition to use the `awslogs` log driver
2. **Structured logging:** Use structured log formats (JSON) for better searchability and analysis
3. **Log levels:** Implement appropriate log levels to control verbosity and costs

4. Retention policies: Set retention periods based on your compliance and debugging needs

Logging server messages (custom servers)

You can capture custom server messages from your Amazon GameLift Servers custom servers in log files.

Important

There is a limit on the size of a log file per game session (see [Amazon GameLift Servers endpoints and quotas](#) in the *AWS General Reference*). When a game session ends, Amazon GameLift Servers uploads the server logs to Amazon Simple Storage Service (Amazon S3). Amazon GameLift Servers will not upload logs that exceed the limit. Logs can grow very quickly and exceed the size limit. You should monitor your logs and limit the log output to necessary messages only.

Configuring logging for custom servers

With Amazon GameLift Servers custom servers, you write your own code to perform logging, which you configure as part of your server process configuration. Amazon GameLift Servers uses your logging configuration to identify the files that it must upload to S3 at the end of each game session.

The following instructions show how to configure logging using simplified code examples:

C++

To configure logging (C++)

1. Create a vector of strings that are directory paths to game server log files.

```
std::string serverLog("serverOut.log");           // Example server log file
std::vector<std::string> logPaths;
logPaths.push_back(serverLog);
```

2. Provide your vector as the [LogParameters](#) of your [ProcessParameters](#) object.

```
Aws::GameLift::Server::ProcessParameters processReadyParameter =
    Aws::GameLift::Server::ProcessParameters(
```

```
std::bind(&Server::onStartGameSession, this, std::placeholders::_1),
std::bind(&Server::onProcessTerminate, this),
std::bind(&Server::OnHealthCheck, this),
std::bind(&Server::OnUpdateGameSession, this),
listenPort,
Aws::GameLift::Server::LogParameters(logPaths));
```

3. Provide the [ProcessParameters](#) object when you call [ProcessReady\(\)](#).

```
Aws::GameLift::GenericOutcome outcome =
    Aws::GameLift::Server::ProcessReady(processReadyParameter);
```

For a more complete example, see [ProcessReady\(\)](#).

C#

To configure logging (C#)

1. Create a list of strings that are directory paths to game server log files.

```
List<string> logPaths = new List<string>();
logPaths.Add("C:\\game\\serverOut.txt");    // Example of a log file that the
game server writes
```

2. Provide your list as the [LogParameters](#) of your [ProcessParameters](#) object.

```
var processReadyParameter = new ProcessParameters(
    this.OnGameSession,
    this.OnProcessTerminate,
    this.OnHealthCheck,
    this.OnGameSessionUpdate,
    port,
    new LogParameters(logPaths));
```

3. Provide the [ProcessParameters](#) object when you call [ProcessReady\(\)](#).

```
var processReadyOutcome =
    GameLiftServerAPI.ProcessReady(processReadyParameter);
```

For a more complete example, see [ProcessReady\(\)](#).

Writing to logs

Your log files exist after your server process has started. You can write to the logs using any method to write to files. To capture all of your server's standard output and error output, remap the output streams to log files, as in the following examples:

C++

```
std::freopen("serverOut.log", "w+", stdout);  
std::freopen("serverErr.log", "w+", stderr);
```

C#

```
Console.SetOut(new StreamWriter("serverOut.txt"));  
Console.SetError(new StreamWriter("serverErr.txt"));
```

Accessing server logs

Log access varies by fleet type:

Managed EC2 fleets

When a game session ends, Amazon GameLift Servers automatically stores the logs in an S3 bucket and retains them for 14 days. To get the location of the logs for a game session, you can use the [GetGameSessionLogUrl](#) API operation. To download the logs, use the URL that the operation returns.

Alternatively, you can set up your own logging solution by configuring your game server to send logs directly to your preferred logging service or storage location. For more information, see [Connect your Amazon GameLift Servers hosted game server to other AWS resources](#).

Container fleets

Container fleets capture standard output and error streams from all containers. You can configure logging using one of these options:

- Save container output as CloudWatch log streams in a specified log group
- Save container output to an S3 storage bucket
- Turn off logging (container output isn't saved)

For detailed information about configuring logging options when creating container fleets, see [LogConfiguration](#) in the Amazon GameLift Servers API Reference.

Security in Amazon GameLift Servers

If you're using Amazon GameLift Servers FleetIQ as a standalone feature with Amazon EC2, see [Security in Amazon EC2](#) in the *Amazon EC2 User Guide*.

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS compliance programs](#). To learn about the compliance programs that apply to Amazon GameLift Servers, see [AWS services in scope by compliance program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using Amazon GameLift Servers. The following topics show you how to configure Amazon GameLift Servers to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon GameLift Servers resources.

Topics

- [Data protection in Amazon GameLift Servers](#)
- [Identity and access management for Amazon GameLift Servers](#)
- [Logging and monitoring with Amazon GameLift Servers](#)
- [Compliance validation for Amazon GameLift Servers](#)
- [Resilience in Amazon GameLift Servers](#)
- [Infrastructure security in Amazon GameLift Servers](#)
- [Configuration and vulnerability analysis in Amazon GameLift Servers](#)

- [Security best practices for Amazon GameLift Servers](#)

Data protection in Amazon GameLift Servers

If you're using Amazon GameLift Servers FleetIQ as a standalone feature with Amazon EC2, see [Security in Amazon EC2](#) in the *Amazon EC2 User Guide*.

The AWS [shared responsibility model](#) applies to data protection in Amazon GameLift Servers. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see [Data Privacy FAQ](#). For information about data protection in Europe, see the [General Data Protection Regulation \(GDPR\) Center](#).

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see [Working with CloudTrail trails](#) in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-3](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with Amazon GameLift Servers or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names

may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Amazon GameLift Servers-specific data is handled as follows:

- Game server builds and scripts that you upload to Amazon GameLift Servers are stored in Amazon S3. There is no direct customer access to this data once it is uploaded. An authorized user can get temporary access to upload files, but can't view or update the files in Amazon S3 directly. To delete scripts and builds, use the Amazon GameLift Servers console or the service API.
- Game session log data is stored in Amazon S3 for a limited period of time after the game session is completed. Authorized users can access the log data by downloading it via a link in the Amazon GameLift Servers console or by calls to the service API.
- Metric and event data is stored in Amazon GameLift Servers and can be accessed through the Amazon GameLift Servers console or by calls to the service API. Data can be retrieved on fleets, instances, game session placements, matchmaking tickets, game sessions, and player sessions. Data can also be accessed through Amazon CloudWatch and CloudWatch Events.
- Customer-supplied data is stored in Amazon GameLift Servers . Authorized users can access it by calls to the service API. Potentially sensitive data might include player data, player session and game session data (including connection info), matchmaker data, and so on.

 Note

If you provide custom player IDs in your requests, it is expected that these values are anonymized UUIDs and contain no identifying player information.

For more information about data protection, see the [AWS shared responsibility model and GDPR](#) blog post on the *AWS Security Blog*.

Encryption at rest

At-rest encryption of Amazon GameLift Servers-specific data is handled as follows:

- Game server builds and scripts are stored in Amazon S3 buckets with server-side encryption.
- Customer-supplied data is stored in Amazon GameLift Servers in an encrypted format.

Encryption in transit

Connections to the Amazon GameLift Servers APIs are made over a secure (SSL) connection and authenticated using [AWS Signature Version 4](#) (when connecting through the AWS CLI or AWS SDK, signing is handled automatically). Authentication is managed using the IAM-defined access policies for the security credentials that are used to make the connection.

Direct communication between game clients and game servers is as follows:

- For custom game servers being hosted on Amazon GameLift Servers resources, communication does not involve the Amazon GameLift Servers service. Encryption of this communication is the responsibility of the customer. You can use TLS-enabled fleets to have your game clients authenticate the game server on connection and to encrypt all communication between your game client and game server.
- Certificates for TLS-enabled fleets are created at the same time as the fleet, and their expiration dates are based on the fleet creation date.
- For Amazon GameLift Servers Realtime with TLS certificate generation enabled, traffic between game client and Realtime servers using the client SDK for Realtime is encrypted in flight. TCP traffic is encrypted using TLS 1.2, and UDP traffic is encrypted using DTLS 1.2.

Note

To meet Certificate Authority requirements for TLS certificates, Amazon GameLift Servers will adjust the maximum certificate lifetimes for fleets configured to generate a certificate. Certificate lifetimes begin on fleet creation and will be changed on the following schedule:

- Until March 11, 2026, the maximum lifetime for a TLS certificate issued is 398 days.
- As of March 1, 2026, the maximum lifetime for a TLS certificate issued will be 200 days.
- As of March 1, 2027, the maximum lifetime for a TLS certificate issued will be 100 days.
- As of March 1, 2029, the maximum lifetime for a TLS certificate issued will be 47 days.

To ensure that your certificates are renewed, and to maintain up-to-date game server runtime environments, Amazon GameLift Servers recommends regularly replacing your game server fleets.

Internet network traffic privacy

You can remotely access your Amazon GameLift Servers instances securely. For instances that use Linux, SSH provides a secure communications channel for remote access. For instances that are running Windows, use a remote desktop protocol (RDP) client. With Amazon GameLift Servers FleetIQ, remote access to your instances using AWS Systems Manager Session Manager and Run Command is encrypted using TLS 1.2, and requests to create a connection are signed using SigV4. For help with connecting to a managed Amazon GameLift Servers instance, see [Connect to fleet instances](#).

Identity and access management for Amazon GameLift Servers

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon GameLift Servers resources. IAM is an AWS service that you can use with no additional charge.

Topics

- [Audience](#)
- [Authenticating with identities](#)
- [Managing access using policies](#)
- [How Amazon GameLift Servers works with IAM](#)
- [Identity-based policy examples for Amazon GameLift Servers](#)
- [Troubleshooting Amazon GameLift Servers identity and access](#)
- [AWS managed policies for Amazon GameLift Servers](#)

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see [Troubleshooting Amazon GameLift Servers identity and access](#))
- **Service administrator** - determine user access and submit permission requests (see [How Amazon GameLift Servers works with IAM](#))

- **IAM administrator** - write policies to manage access (see [Identity-based policy examples for Amazon GameLift Servers](#))

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see [How to sign in to your AWS account](#) in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see [AWS Signature Version 4 for API requests](#) in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see [Tasks that require root user credentials](#) in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see [What is IAM Identity Center?](#) in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An [IAM user](#) is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more

information, see [Require human users to use federation with an identity provider to access AWS using temporary credentials](#) in the *IAM User Guide*.

An [IAM group](#) specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see [Use cases for IAM users](#) in the *IAM User Guide*.

IAM roles

An [IAM role](#) is an identity with specific permissions that provides temporary credentials. You can assume a role by [switching from a user to an IAM role \(console\)](#) or by calling an AWS CLI or AWS API operation. For more information, see [Methods to assume a role](#) in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see [Overview of JSON policies](#) in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Define custom IAM permissions with customer managed policies](#) in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between

managed and inline policies, see [Choose between managed policies and inline policies](#) in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must [specify a principal](#) in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see [Permissions boundaries for IAM entities](#) in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see [Service control policies](#) in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see [Resource control policies \(RCPs\)](#) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. For more information, see [Session policies](#) in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see [Policy evaluation logic](#) in the *IAM User Guide*.

How Amazon GameLift Servers works with IAM

Before you use IAM to manage access to Amazon GameLift Servers, learn what IAM features are available to use with Amazon GameLift Servers.

IAM features you can use with Amazon GameLift Servers

IAM feature	Amazon GameLift Servers support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Yes
Temporary credentials	Yes
Principal permissions	Yes
Service roles	Yes
Service-linked roles	No

To get a high-level view of how Amazon GameLift Servers and other AWS services work with most IAM features, see [AWS services that work with IAM](#) in the *IAM User Guide*.

Identity-based policies for Amazon GameLift Servers

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can

perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Define custom IAM permissions with customer managed policies](#) in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

Identity-based policy examples for Amazon GameLift Servers

To view examples of Amazon GameLift Servers identity-based policies, see [Identity-based policy examples for Amazon GameLift Servers](#).

Resource-based policies within Amazon GameLift Servers

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. For more information, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

Policy actions for Amazon GameLift Servers

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The **Action** element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

For a list of Amazon GameLift Servers actions, see [Actions defined by Amazon GameLift Servers](#) in the *Service Authorization Reference*.

Policy actions in Amazon GameLift Servers use the following prefix before the action:

```
gamelift
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [  
    "gamelift:action1",  
    "gamelift:action2"  
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word Describe, include the following action:

```
"Action": "gamelift:Describe*"
```

To view examples of Amazon GameLift Servers identity-based policies, see [Identity-based policy examples for Amazon GameLift Servers](#).

Policy resources for Amazon GameLift Servers

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Resource JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its [Amazon Resource Name \(ARN\)](#). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

For a list of Amazon GameLift Servers resource types and their ARNs, see [Resources defined by Amazon GameLift Servers](#) in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see [Actions defined by Amazon GameLift Servers](#).

Some Amazon GameLift Servers resources have ARN values, which allows the resources to have their access managed using IAM policies. The Amazon GameLift Servers fleet resource has an ARN with the following syntax:

```
arn:${Partition}:gamelift:${Region}:${Account}:fleet/${FleetId}
```

For more information about the format of ARNs, see [Amazon Resource Names \(ARNs\)](#) in the *AWS General Reference*.

For example, to specify the `fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa` fleet in your statement, use the following ARN:

```
"Resource": "arn:aws:gamelift:us-west-2:123456789012:fleet/fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa"
```

To specify all fleets that belong to a specific account, use a wildcard (*):

```
"Resource": "arn:aws:gamelift:us-west-2:123456789012:fleet/*"
```

To view examples of Amazon GameLift Servers identity-based policies, see [Identity-based policy examples for Amazon GameLift Servers](#).

Policy condition keys for Amazon GameLift Servers

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

For a list of Amazon GameLift Servers condition keys, see [Condition keys for Amazon GameLift Servers](#) in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see [Actions defined by Amazon GameLift Servers](#).

To view examples of Amazon GameLift Servers identity-based policies, see [Identity-based policy examples for Amazon GameLift Servers](#).

ACLs in Amazon GameLift Servers

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Amazon GameLift Servers

Supports ABAC (tags in policies): Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see [Define permissions with ABAC authorization](#) in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see [Use attribute-based access control \(ABAC\)](#) in the *IAM User Guide*.

For an example identity-based policy that limits access to a resource based on the tags on that resource, see [View Amazon GameLift Servers fleets based on tags](#).

Using temporary credentials with Amazon GameLift Servers

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate

temporary credentials instead of using long-term access keys. For more information, see [Temporary security credentials in IAM](#) and [AWS services that work with IAM](#) in the *IAM User Guide*.

Cross-service principal permissions for Amazon GameLift Servers

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see [Forward access sessions](#).

Service roles for Amazon GameLift Servers

Supports service roles: Yes

A service role is an [IAM role](#) that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Create a role to delegate permissions to an AWS service](#) in the *IAM User Guide*.

Warning

Changing the permissions for a service role might break Amazon GameLift Servers functionality. Edit service roles only when Amazon GameLift Servers provides guidance to do so.

Allow your Amazon GameLift Servers-hosted game servers to access other AWS resources, such as an AWS Lambda function or an Amazon DynamoDB database. Because game servers are hosted on fleets that Amazon GameLift Servers manages, you need a service role that gives Amazon GameLift Servers limited access to your other AWS resources. For more information, see [Connect your Amazon GameLift Servers hosted game server to other AWS resources](#).

Service-linked roles for Amazon GameLift Servers

Supports service-linked roles: No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see [AWS services that work with IAM](#) in the *IAM User Guide*. Find a service in the table that includes a Yes in the **Service-linked roles** column. Choose **Yes** to view the service-linked role documentation for that service.

Identity-based policy examples for Amazon GameLift Servers

By default, users and roles don't have permission to create or modify Amazon GameLift Servers resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see [Create IAM policies \(console\)](#) in the *IAM User Guide*.

For details about actions and resource types defined by Amazon GameLift Servers, including the format of the ARNs for each of the resource types, see [Actions, resources, and condition keys for Amazon GameLift Servers](#) in the *Service Authorization Reference*.

Topics

- [Policy best practices](#)
- [Using the Amazon GameLift Servers console](#)
- [Allow users to view their own permissions](#)
- [Allow player access for game sessions](#)
- [Allow access to one Amazon GameLift Servers queue](#)
- [View Amazon GameLift Servers fleets based on tags](#)
- [Access a game build file in Amazon S3](#)

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon GameLift Servers resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies

that are specific to your use cases. For more information, see [AWS managed policies](#) or [AWS managed policies for job functions](#) in the *IAM User Guide*.

- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see [Policies and permissions in IAM](#) in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see [Validate policies with IAM Access Analyzer](#) in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see [Secure API access with MFA](#) in the *IAM User Guide*.

For more information about best practices in IAM, see [Security best practices in IAM](#) in the *IAM User Guide*.

Using the Amazon GameLift Servers console

To access the Amazon GameLift Servers console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon GameLift Servers resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

To ensure that those entities can still use the Amazon GameLift Servers console, add permissions to users and groups with the syntax in the following examples and in [Administration permission examples](#). For more information, see [Set user permissions for Amazon GameLift Servers](#).

Users that work with Amazon GameLift Servers through AWS CLI or AWS API operations don't require minimum console permissions. Instead, you can limit access to only the operations the user needs to perform. For example, a player user, acting on behalf of game clients, requires access to request game sessions, place players into games, and other tasks.

For information about the permissions required to use all Amazon GameLift Servers console features, see permissions syntax for administrators in [Administration permission examples](#).

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsWithUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"]
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

```
    }  
  ]  
}
```

Allow player access for game sessions

To place players into game sessions, game clients and backend services need permissions. For policy examples for these scenarios, see [Player user permission examples](#).

Allow access to one Amazon GameLift Servers queue

The following example provides a user with access to a specific Amazon GameLift Servers queues.

This policy grants the user permissions to add, update, and delete queue destinations with the following actions: `gamelift:UpdateGameSessionQueue`, `gamelift>DeleteGameSessionQueue`, and `gamelift:DescribeGameSessionQueues`. As shown, this policy uses the `Resource` element to limit access to a single queue: `gamesessionqueue/examplequeue123`.

JSON

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "ViewSpecificQueueInfo",  
      "Effect": "Allow",  
      "Action": [  
        "gamelift:DescribeGameSessionQueues"  
      ],  
      "Resource": "arn:aws:gamelift:us-east-1:555555555555:gamesessionqueue/  
examplequeue123"  
    },  
    {  
      "Sid": "ManageSpecificQueue",  
      "Effect": "Allow",  
      "Action": [  
        "gamelift:UpdateGameSessionQueue",  
        "gamelift>DeleteGameSessionQueue"  
      ],  
      "Resource": "arn:aws:gamelift:us-east-1:111122223333:gamesessionqueue/  
examplequeue123"
```



```

    }
  ]
}
```

View Amazon GameLift Servers fleets based on tags

You can use conditions in your identity-based policy to control access to Amazon GameLift Servers resources based on tags. This example shows how you can create a policy that allows viewing a fleet if the `Owner` tag matches the user's user name. This policy also grants the permissions necessary to complete this operation in the console.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ListFleetsInConsole",
      "Effect": "Allow",
      "Action": "gamelift:ListFleets",
      "Resource": "*"
    },
    {
      "Sid": "ViewFleetIfOwner",
      "Effect": "Allow",
      "Action": "gamelift:DescribeFleetAttributes",
      "Resource": "arn:aws:gamelift:*:*:fleet/*",
      "Condition": {
        "StringEquals": {"aws:ResourceAccount": "${aws:username}"}
      }
    }
  ]
}
```

Access a game build file in Amazon S3

After you integrate your game server with Amazon GameLift Servers, upload the build files to Amazon S3. For Amazon GameLift Servers to access the build files, use the following policy.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectVersion"
      ],
      "Resource": "arn:aws:s3:::bucket-name/object-name"
    }
  ]
}
```

For more information about uploading Amazon GameLift Servers game files, see [Create a game server build for Amazon GameLift Servers](#).

Troubleshooting Amazon GameLift Servers identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon GameLift Servers and AWS Identity and Access Management (IAM).

Topics

- [I am not authorized to perform an action in Amazon GameLift Servers](#)
- [I am not authorized to perform iam:PassRole](#)
- [I want to allow people outside of my AWS account to access my Amazon GameLift Servers resources](#)

I am not authorized to perform an action in Amazon GameLift Servers

If the AWS Management Console tells you that you're not authorized to perform an action, contact your AWS account administrator for assistance. Your administrator is the person that provided you with your sign-in credentials.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a queue but doesn't have `gamelift:DescribeGameSessionQueues` permissions:

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
gamelift:DescribeGameSessionQueues on resource: examplequeue123
```

In this case, Mateo asks his administrator to update his policies to allow him read access for the `examplequeue123` resource using the `gamelift:DescribeGameSessionQueues` action.

I am not authorized to perform `iam:PassRole`

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to Amazon GameLift Servers.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon GameLift Servers. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my Amazon GameLift Servers resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support

resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon GameLift Servers supports these features, see [How Amazon GameLift Servers works with IAM](#).
- To learn how to provide access to your resources across AWS accounts that you own, see [Providing access to an IAM user in another AWS account that you own](#) in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see [Providing access to AWS accounts owned by third parties](#) in the *IAM User Guide*.
- To learn how to provide access through identity federation, see [Providing access to externally authenticated users \(identity federation\)](#) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

AWS managed policies for Amazon GameLift Servers

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining [customer managed policies](#) that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see [AWS managed policies](#) in the *IAM User Guide*.

AWS managed policy: GameLiftContainerFleetPolicy

You can attach GameLiftContainerFleetPolicy to your IAM roles.

The policy grants permissions for compute actions in an Amazon GameLift Servers container fleet. A container fleet is a set of hosting resources that Amazon GameLift Servers manages for you. Amazon GameLift Servers needs permissions to connect to the Amazon GameLift Servers service and other AWS services on your behalf.

When creating a container fleet with Amazon GameLift Servers, provide an IAM service role with the `GameLiftContainerFleetPolicy` managed policy attached. For instructions on creating the service role, see [Set up an IAM service role for Amazon GameLift Servers](#).

For more information, see [GameLiftContainerFleetPolicy](#).

Permissions details

This policy includes the following permissions.

- `cloudwatch` – Allows Amazon GameLift Servers to write game session logs to an Amazon CloudWatch Events log stream in your AWS account.
- `cloudwatch` – Allows Amazon GameLift Servers to create a CloudWatch log group to organize game session data in the log stream.
- `s3` – Allows Amazon GameLift Servers to write game session logs to an Amazon Simple Storage Service bucket in your AWS account.
- `s3` – Allows Amazon GameLift Servers to retrieve the AWS Region where a specified Amazon S3 bucket resides, using the API action `s3:GetBucketLocation`.
- `gamelift` – Allows Amazon GameLift Servers to retrieve an authentication token that allows a hosted game server to communicate with the Amazon GameLift Servers service through your AWS account.

Amazon GameLift Servers updates to AWS managed policies

View details about updates to AWS managed policies for Amazon GameLift Servers since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the [Amazon GameLift Servers Release notes page](#).

Change	Description	Date
GameLiftContainerFleetPolicy – Change	Amazon GameLift Servers added new permissions to	February 5, 2024

Change	Description	Date
	retrieve the AWS Region of an Amazon S3 bucket.	
GameLiftContainerFleetPolicy – New policy	Amazon GameLift Servers added new permissions to enable game server containers to run on Amazon GameLift Servers managed fleets.	November 12, 2024
Amazon GameLift Servers started tracking changes	Amazon GameLift Servers started tracking changes for its AWS managed policies.	November 12, 2024

Logging and monitoring with Amazon GameLift Servers

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon GameLift Servers and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs.

AWS and Amazon GameLift Servers provide several tools for monitoring your game hosting resources and responding to potential incidents.

Amazon CloudWatch Alarms

Using Amazon CloudWatch alarms, you watch a single metric over a time period that you specify. If the metric exceeds a given threshold, a notification is sent to an Amazon SNS topic or AWS Auto Scaling policy. CloudWatch alarms are triggered when their state changes and is maintained for a specified number of periods, not by being in a particular state. For more information, see [Monitor Amazon GameLift Servers with Amazon CloudWatch](#).

AWS CloudTrail Logs

CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon GameLift Servers. Using the information collected by CloudTrail, you can determine the request that was made to Amazon GameLift Servers, the IP address from which the request was made, who made

the request, when it was made, and additional details. For more information, see [Logging Amazon GameLift Servers API calls with AWS CloudTrail](#).

Compliance validation for Amazon GameLift Servers

Amazon GameLift Servers is not in scope of any AWS compliance programs.

To learn whether an AWS service is within the scope of specific compliance programs, see [AWS services in Scope by Compliance Program](#) and choose the compliance program that you are interested in. For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. For more information about your compliance responsibility when using AWS services, see [AWS Security Documentation](#).

Resilience in Amazon GameLift Servers

If you're using Amazon GameLift Servers FleetIQ as a standalone feature with Amazon EC2, see [Security in Amazon EC2](#) in the *Amazon EC2 User Guide*.

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see [AWS global infrastructure](#).

In addition to the AWS global infrastructure, Amazon GameLift Servers offers the following features to help support your data resiliency needs:

- **Multi-region queues** – Amazon GameLift Servers game session queues are used to place new game sessions with available hosting resources. Queues that span multiple Regions are able to redirect game session placements in the event of a regional outage. For more information and best practices on creating game session queues, see [Customize a game session queue](#).

- **Automatic capacity scaling** – Maintain the health and availability of your hosting resources by using Amazon GameLift Servers scaling tools. These tools provide a range of options that let you adjust fleet capacity to fit the needs of your game and players. For more information on scaling, see [Scaling game hosting capacity with Amazon GameLift Servers](#).
- **Distribution across instances** – Amazon GameLift Servers distributes incoming traffic across multiple instances, depending on fleet size. As a best practice, games in production should have multiple instances to maintain availability in case an instance becomes unhealthy or unresponsive.
- **Amazon S3 storage** – Game server builds and scripts that are uploaded to Amazon GameLift Servers are stored in Amazon S3 using the Standard storage class, which uses multiple data center replications to increase resilience. Game session logs are also stored in Amazon S3 using the Standard storage class.

Infrastructure security in Amazon GameLift Servers

If you're using Amazon GameLift Servers FleetIQ as a standalone feature with Amazon EC2, see [Security in Amazon EC2](#) in the *Amazon EC2 User Guide*.

As a managed service, Amazon GameLift Servers is protected by the AWS global network security procedures that are described in the [Amazon Web Services: Overview of security processes](#) whitepaper.

You use AWS published API calls to access Amazon GameLift Servers through the network. Clients must support Transport Layer Security (TLS) 1.2 or later. We recommend TLS 1.3 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [AWS Security Token Service](#) (AWS STS) to generate temporary security credentials to sign requests.

The Amazon GameLift Servers service places all fleets into Amazon virtual private clouds (VPCs) so that each fleet exists in a logically isolated area in the AWS Cloud. You can use Amazon GameLift Servers policies to control access from specific VPC endpoints or specific VPCs. Effectively, this isolates network access to a given Amazon GameLift Servers resource from only the specific VPC within the AWS network. When you create a fleet, you specify a range of port numbers and IP

addresses. These ranges limit how inbound traffic can access hosted game servers on a fleet VPC. Use standard security best practices when choosing fleet access settings.

Configuration and vulnerability analysis in Amazon GameLift Servers

If you're using Amazon GameLift Servers FleetIQ as a standalone feature with Amazon EC2, see [Security in Amazon EC2](#) in the *Amazon EC2 User Guide*.

Configuration and IT controls are a shared responsibility between AWS and you, our customer. For more information, see the AWS [shared responsibility model](#). AWS handles basic security tasks like guest operating system (OS) and database patching, firewall configuration, and disaster recovery. These procedures have been reviewed and certified by the appropriate third parties. For more details, see the following resource: [Amazon Web Services: Overview of security processes](#) (whitepaper).

The following security best practices also address configuration and vulnerability analysis in Amazon GameLift Servers:

- Customers are responsible for the management of software that is deployed to Amazon GameLift Servers instances for game hosting. Specifically:
 - Customer-provided game server application software should be maintained, including updates and security patches. To update game server software, upload a new build to Amazon GameLift Servers or provide it in a new container image, create a new fleet for it, and redirect traffic to the new fleet.
 - The base Amazon Machine Image (AMI), which includes the operating system, is updated only when a new fleet is created. To patch, update, and secure the operating system and other applications that are part of the AMI, replace fleets on a regular basis, regardless of game server updates. At minimum, production fleets should be replaced at least once a year. See [Keep fleet runtime environments up to date](#) for more details.
- Customers should consider regularly updating their games with the latest SDK versions, including the AWS SDK, the Amazon GameLift Servers Server SDK, and the Amazon GameLift Servers Client SDK for Realtime Servers.

Security best practices for Amazon GameLift Servers

If you're using Amazon GameLift Servers FleetIQ as a standalone feature with Amazon EC2, see [Security in Amazon EC2](#) in the *Amazon EC2 User Guide*.

Amazon GameLift Servers provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don't represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

Keep fleet runtime environments up to date

Amazon GameLift Servers highly recommends that you regularly replace managed fleets (including managed EC2 and managed container fleets) to maintain secure runtime environments for your game servers. Fleets that run for extended periods without runtime updates can contain outdated dependencies and security vulnerabilities that could compromise your game servers. For details on how responsibility is shared for software deployed to Amazon GameLift Servers fleets, see [Configuration and vulnerability analysis in Amazon GameLift Servers](#).

A managed fleet's runtime environment is determined by its Amazon Machine Image (AMI) version. When a new fleet is created, Amazon GameLift Servers assigns the latest available AMI version to the fleet, and all compute instances in that fleet are deployed with that version. To update the AMI version, you must create a new fleet. For details on current AMI versions, see [Amazon GameLift Servers AMI versions](#).

Recommended practices:

- **Monitor fleet age and replace fleets that are over 30 days old** – You can track a fleet's creation date in the Amazon GameLift Servers console or use the CLI to retrieve fleet attributes. Amazon GameLift Servers displays warnings in the console for fleets that are over 90 days old, and notifies account holders by email for fleets that are older than one year.

Note

Updating a fleet (such as using [UpdateFleetAttributes](#) or [UpdateContainerFleet](#)) doesn't change the AMI version. You must create a new fleet.

- **Replace fleets regularly based on security health** – Set up a regular schedule to create new fleets and retire old fleets. Consider using a service like [Amazon Q](#) to review your game code with the current AMI version, detect security issues, and suggest remediation steps.
- **Test server builds with the latest AMI versions before deployment** – You might need to modify your server build and upload it to Amazon GameLift Servers before creating a new fleet.
- **Manage fleet quotas for your AWS account** – You can request limit increases if needed to create replacement fleets. For more information, see [Amazon GameLift Servers endpoints and quotas](#).
- **Consider automating fleet replacement** – You can automate processes to create new fleets and migrate player traffic from older fleets. For example:
 - Use AWS CloudFormation to automate fleet creation and management. Maintain your fleet configurations as CloudFormation templates and use them to launch resource stacks.
 - Take advantage of the Amazon GameLift Servers alias feature to abstract specific fleet IDs. Fleet aliases make it easy to switch player traffic from an existing fleet to a new one with no disruption to game sessions in progress. For details, see [Abstract an Amazon GameLift Servers fleet designation with an alias](#).
 - Use blue/green deployment strategies to reduce migration risk and maintain zero downtime. With two identical production environments, you can take advantage of a full production-like testing environment, exert greater control over migration process, and ensure instant rollbacks.

Secure your port configurations

We strongly recommend against opening ports to the Internet because doing so poses a security risk. For example, the following configuration opens a remote desktop port that allows anyone on the Internet to access the instance:

```
{
  "FleetId": "<fleet identifier>",
  "InboundPermissionAuthorizations": [
    {
      "FromPort": 3389,
      "IpRange": "0.0.0.0/0",
      "Protocol": "RDP",
      "ToPort": 3389
    }
  ]
}
```

Instead, use [UpdateFleetPortSettings](#) to open a port with a specific IP address or range of addresses, as shown in this example:

```
{
  "FleetId": "<fleet identifier>",
  "InboundPermissionAuthorizations": [
    {
      "FromPort": 3389,
      "IpRange": "54.186.139.221/32",
      "Protocol": "TCP",
      "ToPort": 3389
    }
  ]
}
```

Additional security resources

For more information about how you can make your use of Amazon GameLift Servers more secure, see the [AWS Well-Architected Tool Security pillar](#).

Amazon GameLift Servers pricing and cost planning

Understanding the Amazon GameLift Servers pricing structure and implementing cost-effective strategies in your game hosting solution is essential for optimizing your gaming infrastructure budget. For detailed service information, including example pricing scenarios and pricing tools, see [Amazon GameLift Servers Pricing](#).

New AWS customers can use Amazon GameLift Servers without incurring charges under the Free Tier for up to 12 months. Stay within Free Tier usage limits to avoid charges.

Pricing models

Amazon GameLift Servers offers several pricing models designed to accommodate different usage scenarios and business needs. Each model has a different cost basis. You can control costs by adjusting the cost factors in each model.

Managed hosting

Managed EC2 and managed container fleets use Amazon EC2 instances that Amazon GameLift Servers manage. When you set up game hosting, you choose the type of Amazon EC2 instance to use.

- **Compute costs** – Pay for hourly EC2 instance usage for hosting game sessions. Cost factors for instances include:
 - AWS Region
 - Instance family, type, and size
 - Use of Spot versus On-Demand instances
 - Operating system (managed container fleets run on Linux only)
- **Data transfer** – Pay for traffic between game clients and hosted game servers

Network bandwidth is free for all instance types from generation 6 and later.

Anywhere hosting

Anywhere hosting runs on game hosting resources that you supply and manage, so there are no compute costs. Instead costs are based on:

- **Game sessions** – Pay based on the number of game sessions placed on Anywhere computes

- **Connection minutes** – Pay for server process connection minutes

Use the **Pricing calculator for Amazon GameLift Servers** to estimate costs for different Amazon GameLift Servers configurations. Use the calculator when designing your game hosting solution or to forecast costs for configuration changes. See [Generate Amazon GameLift Servers pricing estimates](#).

Generate Amazon GameLift Servers pricing estimates

With AWS Pricing Calculator, you can [create a pricing estimate for Amazon GameLift Servers](#). You don't need an AWS account or in-depth knowledge of AWS to use the calculator.

AWS Pricing Calculator calculator guides you through the decisions that affect service costs to give you an idea of how much Amazon GameLift Servers might cost for your game project. If you're not yet sure how you plan to use Amazon GameLift Servers, then use the default values to generate an estimate. When planning for production usage, the calculator can help you test out potential scenarios and generate more accurate estimates.

Estimate Amazon GameLift Servers managed hosting

This option provides a cost estimate for hosting your games on Amazon GameLift Servers managed servers, including the costs for server instance usage and data transfer. With Amazon GameLift Servers managed hosting, there is no additional cost for FlexMatch matchmaking.

If you are hosting or plan to host game servers in more than one AWS Region or on more than one instance type, create an estimate for each Region and instance type.

Amazon GameLift Servers instances

This section helps you estimate the type and number of compute resources that you need to host game sessions for your players. Amazon GameLift Servers uses [Amazon Elastic Compute Cloud \(Amazon EC2\) instances](#) to manage game servers. In Amazon GameLift Servers, you deploy a fleet of instances with a specific instance type and operating system. If you have or plan to have multiple fleets, create an estimate for each fleet.

To get started, open the [Configure Amazon GameLift Servers page](#) of AWS Pricing Calculator. Add a **Description**, choose a **Region**, and then choose **Estimate Amazon GameLift Servers hosting (Instance + Data Transfer Out)**. Under **Amazon GameLift Servers instances**, complete the following fields:

- **Peak concurrent players (peak CCU)**

This is the maximum number of players who can connect to your game servers at the same time. This field indicates how much hosting capacity Amazon GameLift Servers needs to meet peak player demand. Enter the daily peak number of players that you expect to host using instances in your chosen AWS Region.

For example, if you want to let 1,000 players connect to your game at any one time, keep the default value of **1000**.

- **Average CCU per hour as a percentage of peak daily CCU**

This is the average number of concurrent players per hour over a 24-hour period. We use this value to estimate the amount of sustained hosting capacity that Amazon GameLift Servers needs to maintain for your players. If you're not sure what percentage value to use, keep the default value of **50** percent. For games with stable player demand, we recommend entering a value of **70** percent.

For example, if your game has an average hourly CCU of 6,000 and a peak CCU of 10,000, then enter the value of **60** percent.

- **Game sessions per instance**

This is the number of game sessions that each of your game server instances can host concurrently. Factors that can affect this number include the resource requirements of your game server, the number of players to host in each game session, and player performance expectations. If you know the number of concurrent game sessions for your game, then enter that value. Alternatively, keep the default value of **20**.

- **Players per game session**

This is the average number of players who connect to a game session, as defined in your game design. If you have game modes with different number of players, estimate an average number of players per game session across your entire game. The default value is **8**.

- **Instance idle buffer %**

This is the percentage of unused hosting capacity to maintain in reserve to handle sudden spikes in player demand. Buffer size is a percentage of the total number of instances in a fleet. The default value is **10** percent.

For example, with a 20 percent idle buffer, a fleet supporting players with 100 active instances maintains 20 idle instances.

- **Spot instance %**

Amazon GameLift Servers fleets can use a combination of On-Demand Instances and Spot Instances. While On-Demand Instances offer more reliable availability, Spot Instances offer a highly cost-efficient alternative. We recommend using a combination to optimize both cost savings and availability. For information about how Amazon GameLift Servers uses Spot Instances, see [On-Demand Instances versus Spot Instances](#).

For this field, enter the percentage of Spot Instances to maintain in a fleet. We recommend a Spot Instance percentage between 50 and 85 percent. The default value is **50** percent.

For example, if you deploy a fleet with 100 instances and specify **40** percent, Amazon GameLift Servers works to maintain 60 On-Demand Instances and 40 Spot Instances.

- **Instance type**

Amazon GameLift Servers fleets can use a range of Amazon EC2 instance types that vary in computing power, memory, storage, and networking capabilities. When you configure an Amazon GameLift Servers fleet, choose an instance type that best fits your game's needs. For information about selecting an instance type with Amazon GameLift Servers, see [Choose compute resources for a managed fleet](#).

If you know the instance type that you're using or plan to use in your Amazon GameLift Servers fleet, choose that type. If you're not sure what type to choose, consider choosing **c5.large**. This is a high-availability type with average size and capabilities.

- **Operating system**

This field specifies the operating system that your game servers run on—either Linux or Windows. The default value is **Linux**.

Data transfer out (DTO)

This section helps you estimate the cost for traffic between your game clients and the game servers. Data transfer fees apply to outbound traffic only. Inbound data transfer has no cost.

Network bandwidth is free for all instance types from generation 6 and later.

On the [Configure Amazon GameLift Servers page](#) of AWS Pricing Calculator, expand **Data transfer out (DTO)**, and then complete the following fields:

- **DTO estimate type**

You can choose to estimate DTO in either of the following two ways, depending on how you track data transfer for your game.

- **Per month (in GB)** – If you track monthly traffic for your game servers, choose this type.
- **Per player** – If you track data transfer by player, choose this type. This is the default type.

In the following field, you estimate per-player DTO based on the number of player hours that you calculated in the previous section.

- **DTO per month (in GB)**

If you chose the **Per month (in GB)** DTO estimate type, then enter your estimated monthly DTO usage in GB from each instance, per Region.

- **DTO per player**

If you chose the **Per player** DTO estimate type, then enter your game's estimated DTO usage per player in KB/sec. The default value is **4**.

When you're done configuring your Amazon GameLift Servers pricing estimate, choose **Add to my estimate**. For more information about creating and managing estimates in AWS Pricing Calculator, see [Create an estimate, configure a service, and add more services](#) in the *AWS Pricing Calculator User Guide*.

Cost optimization strategies

Use these strategies with a managed hosting solution to help reduce your cloud hosting costs while maintaining high performance and player experience.

Best practices

Follow these tips to incorporate cost optimization practices for your game hosting. At a minimum, we recommend that you schedule regular reviews of your Amazon GameLift Servers costs and usage patterns. If you have pricing-related questions, reach out to your account manager.

Fleet configuration

- **Use the pricing calculator** – Use this tool with your game hosting data to explore options and test out potential configuration scenarios for cost savings. See [AWS Pricing Calculator for Amazon GameLift Servers](#).
- **Save in development** – For development and testing, use smaller, less expensive instance types and run them only when you need to.
- **Match instance type to game** – Choose Amazon EC2 instance families, types, and sizes for your fleets that best suit your game's requirements. See the Instance types section in [Choose compute resources for a managed fleet](#).
- **Use Graviton instance types** – Graviton instances are powered by ARM-based processors. They offer better price-performance, higher energy efficiency, and lower costs than comparable x86-based instances with the On-Demand pricing model.
- **Use Spot instances** – Spot instances are usually lower cost than On-Demand but may not always be available. Combine Spot and On-Demand fleets to balance low cost and high availability (50-85% Spot usage recommended). Use Spot instances for non-critical game modes or during off-peak hours. To monitor Spot instance viability, track metrics such as InstanceInterruptions and GameServerInterruptions. See [On-Demand Instances versus Spot Instances](#).
- **Run servers on Linux** – Build your game server runtime for Linux. Instances that are deployed with Linux are generally more cost-efficient than those with Windows.
- **Optimize resource utilization** – Configure fleets for maximum efficiency by running as many concurrent game server processes as possible while maintaining performance. See [Optimize game server runtime configuration on managed Amazon GameLift Servers](#).
- **Create a multi-region strategy** – Deploying game hosting to multiple locations can give your players lower latency and better backup options. Balance regional coverage and cost efficiency by deploying hosting in your largest player markets first and using secondary locations for overflow capacity during peak times. See [Build a multi-location queue](#).

Fleet capacity scaling

- **Scale to zero** – When fleets are not in use, manually set fleet capacity to zero to avoid unnecessary charges. See [Manually set capacity for an Amazon GameLift Servers fleet](#).

- **Add auto scaling** – Avoid over-provisioning hosting resources by using auto scaling to adjust game hosting capacity. Match capacity to fluctuating player demand and other key metrics. See [Auto-scale fleet capacity with Amazon GameLift Servers](#).
- **Maintain a buffer** – To handle sudden spikes in player demand without making players wait, use target tracking to maintain a buffer of idle game servers. Customize the buffer based on the size and usage patterns of your player base. See [Target-based auto scaling](#).

Game session placement

- **Use placement queues** – Amazon GameLift Servers queues rely on an algorithm to determine the "best possible" hosting locations for game sessions, based on the cost of hosting resources and other factors. See [Configure game session placement](#).
- **Customize queue priorities** – You can change how a queue prioritizes hosting costs when placing game sessions. See [Prioritize game session placement](#).
- **Monitor placement metrics** – Track how queues are choosing locations for game sessions to identify optimization opportunities. See [Amazon GameLift Servers metrics for queues](#).

Data transfer

- **Track data transfers** – Monitor how data is transferred between your game clients and servers and take steps to optimize activity.
- **Use data compression** – Consider implementing data compression techniques for data transfers between game clients and servers. Compression plays a crucial role in reducing bandwidth usage as well as improving gameplay performance and latency.

Resource cost and utilization tools

Explore how to use AWS tools to monitor and optimize your game hosting costs with Amazon GameLift Servers. For information on additional tools, see [AWS Billing and Cost Management](#).

AWS cost management tools

- **Billing console** – Review your AWS bills and usage. See [Getting set up with Billing](#).
- **Free Tier usage alerts** – Set up notifications when approaching Free Tier limits. See [Tracking your AWS Free Tier usage](#).

- **Amazon CloudWatch billing alerts** – Configure alerts when usage hits custom thresholds. See [Create a billing alarm to monitor your estimated AWS charges](#).

Cost allocation and tracking

- **Cost allocation tags** – Tag your fleets and other resources to organize and track hosting costs. See [Organizing and tracking costs using AWS cost allocation tags](#).
- **Cost reports** – Create reports categorized by assigned tags. See [Using the monthly cost allocation report](#).
- **AWS Cost Explorer** – Analyze costs, trends, and forecasts with filtering and customizable views. See [Analyzing your costs and usage with AWS Cost Explorer](#).
- **AWS Budgets** – Track and take action on your AWS costs and usage. See [Managing your costs with AWS Budgets](#).

Performance monitoring with Amazon GameLift Servers and Amazon CloudWatch

Monitor these key metrics to optimize resource utilization. View metrics in the Amazon GameLift Servers console or use Amazon CloudWatch dashboards. For details on all available metrics for Amazon GameLift Servers, see [Monitor Amazon GameLift Servers with Amazon CloudWatch](#).

- **Instance metrics** – ActiveInstances, IdleInstances, PercentIdleInstances
- **Server process metrics** – ActiveServerProcesses, HealthyServerProcesses
- **Game session metrics** – ActiveGameSessions, AvailableGameSessions
- **Player session metrics** – CurrentPlayerSessions
- **Queue metrics** – AverageWaitTime, QueueDepth
- **Matchmaking metrics** – CurrentTickets, MatchesPlaced
- **Hardware performance** – CPUUtilization, NetworkIn/NetworkOut, DiskReadBytes/DiskWriteBytes
- **Resource utilization metrics**
 - Use PercentIdleInstances to determine optimal fleet size.
 - Track PercentAvailableGameSessions to ensure sufficient game hosting capacity.
 - Monitor InstanceInterruptions and GameServerInterruptions to determine Spot Instance viability.

Amazon GameLift Servers reference guides

This section contains reference documentation for using Amazon GameLift Servers.

Topics

- [Service API for Amazon GameLift Servers](#)
- [Server SDK 5.x for Amazon GameLift Servers](#)
- [Server SDK for Amazon GameLift Servers version 4 and earlier](#)
- [Game session placement events](#)
- [Amazon GameLift Servers AMI versions](#)
- [Amazon GameLift Servers service locations](#)
- [Amazon GameLift Servers endpoints and quotas](#)
- [Amazon GameLift Servers API limits](#)
- [Amazon GameLift Servers SDK API limits](#)
- [Amazon GameLift Servers UDP ping beacons](#)

Service API for Amazon GameLift Servers

Use this task-based list to find API operations when building your Amazon GameLift Servers game hosting solutions and other features. The AWS SDK includes these operations in the `aws.gamelift` namespace. [Download the AWS SDK](#) or [view the Amazon GameLift Servers API reference documentation](#). You can also use the API with the AWS command line interface (AWS CLI), as documented in the [AWS CLI command reference](#).

The API includes two sets of operations for managed game hosting:

- [Manage Amazon GameLift Servers hosting resources](#)
- [Start game sessions and join players](#)

The Amazon GameLift Servers Service API also contains operations for use with other Amazon GameLift Servers tools and solutions. For a list of FleetIQ APIs, see [FleetIQ API operations](#). For a list of FlexMatch APIs for matchmaking, see [FlexMatch API operations](#).

Manage Amazon GameLift Servers hosting resources

Call these operations to configure hosting resources for your game servers, scale capacity to meet player demand, get performance and usage metrics, and more. Use these API operations when hosting game servers with Amazon GameLift Servers, including Amazon GameLift Servers Realtime. You can also work in [Amazon GameLift Servers console](#) for most resource management tasks, or you can make calls with the AWS Command Line Interface (AWS CLI) tool.

Prepare game servers for deployment

Upload and configure your game's game server code in preparation for deployment and launching on hosting resources.

Manage custom game server builds

- [upload-build](#) – Upload build files from a local path and create a new Amazon GameLift Servers build resource. This operation, available as an AWS CLI command, is the most common way to upload game server builds.
- [CreateBuild](#) – Create a new build using files stored in an Amazon S3 bucket.
- [ListBuilds](#) – Get a list of all builds uploaded to an Amazon GameLift Servers region.
- [DescribeBuild](#) – Retrieve information associated with a build.
- [UpdateBuild](#) – Change build metadata, including build name and version.
- [DeleteBuild](#) – Remove a build from Amazon GameLift Servers.

Manage Amazon GameLift Servers Realtime configuration scripts

- [CreateScript](#) – Upload JavaScript files and create a new Amazon GameLift Servers script resource.
- [ListScripts](#) – Get a list of all Realtime scripts uploaded to an Amazon GameLift Servers region.
- [DescribeScript](#) – Retrieve information associated with a Realtime script.
- [UpdateScript](#) – Change script metadata and upload revised script content.
- [DeleteScript](#) – Remove a Realtime script from Amazon GameLift Servers.

Set up computing resources for hosting

Configure hosting resources and deploy them with your game server build or Realtime configuration script.

Create and manage fleets

- [CreateFleet](#) – Configure and deploy a new Amazon GameLift Servers fleet of computing resources to run your game servers. Once deployed, game servers are automatically launched as configured and ready to host game sessions.
- [ListFleets](#) – Get a list of all fleets in an Amazon GameLift Servers region.
- [DeleteFleet](#) – Remove a fleet that's no longer running game servers or hosting players.
- View / update fleet locations.
 - [CreateFleetLocations](#) – Add remote locations to an existing fleet that supports multiple locations
 - [DescribeFleetLocationAttributes](#) – Get a list of all remote locations for a fleet and view the current status of each location.
 - [DeleteFleetLocations](#) – Remove remote locations from a fleet that supports multiple locations.
- View / update fleet configurations.
 - [DescribeFleetAttributes](#) / [UpdateFleetAttributes](#) – View or change a fleet's metadata and settings for game session protection and resource creation limits.
 - [DescribeFleetPortSettings](#) / [UpdateFleetPortSettings](#) – View or change the inbound permissions (IP address and port setting ranges) allowed for a fleet.
 - [DescribeRuntimeConfiguration](#) / [UpdateRuntimeConfiguration](#) – View or change what server processes (and how many) to run on each instance in a fleet.

Manage fleet capacity

- [DescribeEC2InstanceLimits](#) – Retrieve maximum number of instances allowed for the current AWS account and the current usage level.
- [DescribeFleetCapacity](#) – Retrieve the current capacity settings for a fleet's home Region.
- [DescribeFleetLocationCapacity](#) – Retrieve the current capacity settings for each location a multi-location fleet.
- [UpdateFleetCapacity](#) – Manually adjust capacity settings for a fleet.
- Set up :
 - [PutScalingPolicy](#) – Turn on target-based auto scaling or create a custom auto scaling policy, or update an existing policy.
 - [DescribeScalingPolicies](#) – Retrieve an existing auto scaling policy.

- [DeleteScalingPolicy](#) – Delete an auto scaling policy and stop it from affecting a fleet's capacity.
- [StartFleetActions](#) – Restart a fleet's auto scaling policies.
- [StopFleetActions](#) – Suspend a fleet's auto scaling policies.

Monitor fleet activity.

- [DescribeFleetUtilization](#) – Retrieve statistics on the number of server processes, game sessions, and players that are currently active on a fleet.
- [DescribeFleetLocationUtilization](#) – Retrieve utilization statistics for each location in a multi-location fleet.
- [DescribeFleetEvents](#) – View logged events for a fleet during a specified time span.
- [DescribeGameSessions](#) – Retrieve game session metadata, including a game's running time and current player count.

Set up queues for game session placement

Set up multi-fleet, multi-region queues to place game sessions with the best available hosting resources for cost, latency, and resiliency.

- [CreateGameSessionQueue](#) – Create a queue for use when processing requests for game session placements.
- [DescribeGameSessionQueues](#) – Retrieve game session queues defined in an Amazon GameLift Servers region.
- [UpdateGameSessionQueue](#) – Change the configuration of a game session queue.
- [DeleteGameSessionQueue](#) – Remove a game session queue from the region.

Manage aliases

Use aliases to represent your fleets or create a terminal alternative destination. Aliases are useful when transitioning game activity from one fleet to another, such as during game server build updates.

- [CreateAlias](#) – Define a new alias and optionally assign it to a fleet.
- [ListAliases](#) – Get all fleet aliases defined in an Amazon GameLift Servers region.
- [DescribeAlias](#) – Retrieve information on an existing alias.

- [UpdateAlias](#) – Change settings for an alias, such as redirecting it from one fleet to another.
- [DeleteAlias](#) – Remove an alias from the region.
- [ResolveAlias](#) – Get the fleet ID that a specified alias points to.

Connect to managed hosting instances

View information on individual instances in a fleet, or request remote access to a specified fleet instance for troubleshooting.

- [DescribeInstances](#) – Get information on each instance in a fleet, including instance ID, IP address, location, and status.
- [GetInstanceAccess](#) – Request access credentials needed to remotely connect to a specified instance in a fleet.

Set up VPC peering

Create and manage VPC peering connections between your Amazon GameLift Servers hosting resources and other AWS resources.

- [CreateVpcPeeringAuthorization](#) – Authorize a peering connection to one of your VPCs.
- [DescribeVpcPeeringAuthorizations](#) – Retrieve valid peering connection authorizations.
- [DeleteVpcPeeringAuthorization](#) – Delete a peering connection authorization.
- [CreateVpcPeeringConnection](#) – Establish a peering connection between the VPC for an Amazon GameLift Servers fleet and one of your VPCs.
- [DescribeVpcPeeringConnections](#) – Retrieve information on active or pending VPC peering connections with an Amazon GameLift Servers fleet.
- [DeleteVpcPeeringConnection](#) – Delete a VPC peering connection with an Amazon GameLift Servers fleet.

Start game sessions and join players

Call these operations from a backend service to start new game sessions, get information on existing game sessions, and join players to game sessions. These operations are for use with custom game servers that are hosted on Amazon GameLift Servers. If you're using Amazon

GameLift Servers Realtime, manage game sessions using the [Amazon GameLift Servers Realtime client API \(C#\)](#).

- **Start new game sessions for one or more players.**
 - [StartGameSessionPlacement](#) – Ask Amazon GameLift Servers to find the best available hosting resources and start a new game session. This is the preferred method for creating new game sessions. It relies on game session queues to track hosting availability across multiple regions, and uses FleetIQ algorithms to prioritize placements based on player latency, hosting cost, location, etc.
 - [DescribeGameSessionPlacement](#) – Get details and status on a placement request.
 - [StopGameSessionPlacement](#) – Cancel a placement request.
 - [CreateGameSession](#) – Start a new, empty game session on a specific fleet location. This operation gives you greater control over where to start the game session, instead of using FleetIQ to evaluate placement options. You must add players to the new game session in a separate step.
- **Get players into existing game sessions.** Find running game sessions with available player slots and reserve them for new players.
 - [CreatePlayerSession](#) – Reserve an open slot for a player to join a game session.
 - [CreatePlayerSessions](#) – Reserve open slots for multiple players to join a game session.
- **Work with game session and player session data.** Manage information on game sessions and player sessions.
 - [SearchGameSessions](#) – Request a list of active game sessions based on a set of search criteria.
 - [DescribeGameSessions](#) – Retrieve metadata for specific game sessions, including length of time active and current player count.
 - [DescribeGameSessionDetails](#) – Retrieve metadata, including the game session protection setting, for one or more game sessions.
 - [DescribePlayerSessions](#) – Get details on player activity, including status, playing time, and player data.
 - [UpdateGameSession](#) – Change game session settings, such as maximum player count and join policy.
 - [GetGameSessionLogUrl](#) – Get the location of saved logs for a game session.

Server SDK 5.x for Amazon GameLift Servers

This section provides reference documentation for the server SDK 5.x for Amazon GameLift Servers. The server SDK provides core functionality that your game server uses to interact with the Amazon GameLift Servers service. For example, your game server receives prompts from the service to start and stop game sessions and it provides regular game session status updates to the service. Integrate your game servers with the server SDK before you deploy them for hosting.

Use this server SDK reference to integrate your custom multiplayer game servers for hosting with Amazon GameLift Servers. For guidance about the integration process, see [Add Amazon GameLift Servers to your game server with the server SDK](#).

The latest major version of the server SDK for Amazon GameLift Servers is 5.x. The following hosting features require the use of version 5.x:

- Amazon GameLift Servers Anywhere
- Amazon GameLift Servers plugin for Unreal Engine and Unity

Note

If you need to use server SDK version 4.x or earlier, see [Server SDK for Amazon GameLift Servers version 4 and earlier](#) for documentation and download information.

Topics

- [Updates in server SDK 5 for Amazon GameLift Servers](#)
- [Migrate to server SDK 5.x for Amazon GameLift Servers](#)
- [C++ server SDK 5.x for Amazon GameLift Servers -- Actions](#)
- [C# server SDK 5.x for Amazon GameLift Servers -- Actions](#)
- [Go server SDK for Amazon GameLift Servers -- Actions](#)
- [C++ \(Unreal\) server SDK 5.x for Amazon GameLift Servers -- Actions](#)

Updates in server SDK 5 for Amazon GameLift Servers

Your hosted game servers use the server SDK for Amazon GameLift Servers to communicate with the Amazon GameLift Servers service to start and manage game sessions for players. The latest

version, Amazon GameLift Servers server SDK 5, offers a number of improvements and support for new Amazon GameLift Servers features. If your game server build currently uses Amazon GameLift Servers server SDK 4 or earlier, follow the guidance in this topic to update your games.

Amazon GameLift Servers server SDK version 5.0.0 and above includes these updates:

- Expanded languages – Libraries are available in the following languages: C++, C#, Go. You can build the C++ libraries for use with Unreal Engine.
- Game engine plugin support – The Amazon GameLift Servers standalone plugins for Unreal Engine and Unity require Amazon GameLift Servers server SDK 5 libraries. These plugins offer guided workflows for integrating, testing, and deploying your games to Amazon GameLift Servers for hosting. See [Amazon GameLift Servers plugin for Unity \(server SDK 5.x\)](#) and [Amazon GameLift Servers plugin for Unreal Engine](#) documentation.
- Amazon GameLift Servers Anywhere support – With Anywhere fleets you can set up your own hosting resources to use Amazon GameLift Servers features (including matchmaking). Add the Amazon GameLift Servers Agent to automate game session life cycle management. Use Anywhere fleets for production hosting with on- premises hardware, or set up test environments for fast iterative game development. See [Anywhere hosting](#) and the [Amazon GameLift Servers Agent](#).
- Updated testing tools – The Amazon GameLift Servers Anywhere feature lets you set up local or cloud-based test environments for your games. Set up testing with or without the Amazon GameLift Servers Agent. These tools replace Amazon GameLift Servers Local. See [Set up local testing with Amazon GameLift Servers Anywhere](#).
- Consolidated .NET solution for C# – The C# server SDK 5.1+ supports .NET Framework 4.6.2 (upgraded from 4.6.1) and .NET 6.0 in a single solution. .NET Standard 2.1 is available with the Unity-built libraries.
- New Compute resource – This new resource combines different types of hosting resources. It includes cloud-based hosting resources (managed EC2 or container fleets) and customer-controlled hosting resources (Anywhere fleets). It includes the following updates:
 - New API calls for the Compute resource include: [ListCompute\(\)](#), [DescribeCompute\(\)](#), and [GetComputeAccess\(\)](#). These actions return hosting resource information for any type of Amazon GameLift Servers fleet. In general, for fleets with game servers that use server SDK 5.x, use the compute-specific actions to replace instance-specific actions. In addition, these actions are for use in Anywhere fleets without the Amazon GameLift Servers Agent: [RegisterCompute\(\)](#), [DeregisterCompute\(\)](#), and [GetComputeAuthToken\(\)](#).

- New metric `ActiveCompute` with CloudWatch dimensions `FleetId`, `Location`, and `ComputeType`. This metric replaces the previous metric `ActiveInstances`.
- Amazon EC2 Systems Manager (SSM) for remote access – For added security, use SSM instead of SSH when connecting to instances in Amazon GameLift Servers managed fleets. See [Connect to fleet instances](#).

Migrate to server SDK 5.x for Amazon GameLift Servers

To update a game project to use server SDK version 5.x, make the following changes:

Note

If you need to continue using server SDK version 4.x or earlier, see [Server SDK for Amazon GameLift Servers version 4 and earlier](#) for documentation and download information. For SDK version 4.0.2 specifically, you can download it from the [official GitHub releases](#).

1. Get the latest server SDK for Amazon GameLift Servers package for your development environment [[Download site](#)]. Follow the installation instructions in the Readme file for your downloaded package and version. See these instructions for using the server SDKs with your game project.
 - [For development environments using C++, C#, or Go](#)
 - [For Unreal Engine projects \(C++ server SDK for Unreal libraries only\)](#)
 - [For Unity projects \(C# server SDK for Unity libraries only\)](#)
 - [For use with the Amazon GameLift Servers plugin for Unreal Engine](#)
 - [For use with the Amazon GameLift Servers plugin for Unity](#)
2. Update your server code as follows:
 - Change the server code callback function `onCreateGameSession()` to `onStartGameSession()`.
 - Update the `InitSDK()` inputs as appropriate:
 - If you plan to deploy the game server build to either an Amazon GameLift Servers managed EC2 fleet or an Anywhere fleet with the Amazon GameLift Servers Agent:

Call `InitSDK()` with no parameters ([C++](#)) ([C#](#)) ([Unreal](#)). This call sets up the compute environment and a WebSocket connection to the Amazon GameLift Servers service.

- If you plan to deploy the game server build to an Anywhere fleet without the Amazon GameLift Servers Agent:

Call `InitSDK()` with server parameters ([C++](#)) ([C#](#)) ([Unreal](#)). A game server process uses these parameters to establish a connection with the Amazon GameLift Servers service.

3. If your game server build or other hosted applications communicate with other AWS resources while running, you'll need to change how the application gets access to those resources. Replace the use of `AssumeRoleCredentials` with the new server SDK action `GetFleetRoleCredentials()` (for game servers) or use shared credentials (for other applications). For more on how to implement this change, see [Connect your Amazon GameLift Servers hosted game server to other AWS resources](#).
4. If your project called the server SDK action `GetInstanceCertificate()` to retrieve a TLS certificate, modify your code to use the new `GetComputeCertificate()` ([C++](#)) ([C#](#)) ([Unreal](#)) instead.
5. When uploading your game build to Amazon GameLift Servers (such as with [upload-build](#) or [CreateBuild\(\)](#)), set the `ServerSdkVersion` parameter to the 5.x version you're using (this parameter currently defaults to 4.0.2). This parameter must match the actual server SDK libraries in the game server build. If you specify the wrong version for an uploaded game server build, any fleets you create with that build will fail. See [Create a game server build for Amazon GameLift Servers](#).

The following example illustrates how to specify the server SDK version:

```
aws gamelift upload-build \  
  --operating-system AMAZON_LINUX_2023 \  
  --server-sdk-version "5.0.0" \  
  --build-root "~/mygame" \  
  --name "My Game Nightly Build" \  
  --build-version "build 255" \  
  --region us-west-2
```

6. If you use scripts to remotely connect to managed fleets, update the scripts to use the new process, as described in [Connect to fleet instances](#).

C++ server SDK 5.x for Amazon GameLift Servers -- Actions

Use the server SDK 5.x reference to integrate your multiplayer game for hosting with Amazon GameLift Servers. For guidance about the integration process, see [Add Amazon GameLift Servers to your game server with the server SDK](#).

Note

This topic describes the Amazon GameLift Servers C++ API that you can use when you build with the C++ Standard Library (std). Specifically, this documentation applies to code that you compile with the `-DDGAMELIFT_USE_STD=1` option.

C++ server SDK 5.x for Amazon GameLift Servers -- Data types

Use the Amazon GameLift Servers C++ server SDK 5.x reference to integrate your multiplayer game for hosting with Amazon GameLift Servers. For guidance about the integration process, see [Add Amazon GameLift Servers to your game server with the server SDK](#).

Note

This topic describes the Amazon GameLift Servers C++ API that you can use when you build with the C++ Standard Library (std). Specifically, this documentation applies to code that you compile with the `-DDGAMELIFT_USE_STD=1` option.

[C++ server SDK 5.x for Amazon GameLift Servers -- Actions](#)

Data types

- [LogParameters](#)
- [MetricsParameters](#)
- [ProcessParameters](#)
- [UpdateGameSession](#)
- [GameSession](#)
- [ServerParameters](#)
- [StartMatchBackfillRequest](#)

- [Player](#)
- [DescribePlayerSessionsRequest](#)
- [StopMatchBackfillRequest](#)
- [AttributeValue](#)
- [GetFleetRoleCredentialsRequest](#)
- [AwsLongOutcome](#)
- [AwsStringOutcome](#)
- [DescribePlayerSessionsOutcome](#)
- [DescribePlayerSessionsResult](#)
- [GenericOutcome](#)
- [GenericOutcomeCallable](#)
- [PlayerSession](#)
- [StartMatchBackfillOutcome](#)
- [StartMatchBackfillResult](#)
- [GetComputeCertificateOutcome](#)
- [GetComputeCertificateResult](#)
- [GetFleetRoleCredentialsOutcome](#)
- [GetFleetRoleCredentialsResult](#)
- [ListContainersNetworkInfoOutcome](#)
- [ListContainersNetworkInfoResult](#)
- [ContainerNetworkInfo](#)
- [InitSDKOutcome](#)
- [GameLiftError](#)
- [Enums](#)

LogParameters

An object identifying files generated during a game session that you want Amazon GameLift Servers to upload and store after the game session ends. The game server provides `LogParameters` to Amazon GameLift Servers as part of a `ProcessParameters` object in a [ProcessReady\(\)](#) call.

Properties	Description
LogPaths	The list of directory paths to game server log files you want Amazon GameLift Servers to store for future access. The server process generates these files during each game session. You define file paths and names in your game server and store them in the root game build directory. The log paths must be absolute. For example, if your game build stores game session logs in a path like <code>MyGame\sessionLogs\</code>, then the path would be <code>c:\game\MyGame\sessionLogs</code> on a Windows instance. Type: <code>std::vector<std::string></code> Required: No

MetricsParameters

Use this data type to configure metrics collection and crash reporting for the game server. The game server communicates `MetricsParameters` to Amazon GameLift Servers in an [InitMetrics\(\)](#) call.

Properties	Description
StatsdHost	The hostname or IP address of the StatsD server for metrics collection. Type: <code>std::string</code> Required: No
StatsdPort	The port number of the StatsD server for metrics collection.

	Type: <code>int</code> Required: No
CrashReporterHost	The hostname or IP address of the crash reporter server. Type: <code>std::string</code> Required: No
CrashReporterPort	The port number of the crash reporter server. Type: <code>int</code> Required: No
FlushIntervalMs	The interval in milliseconds for flushing metrics data to the server. Type: <code>int</code> Required: No
MaxPacketSize	The maximum size in bytes for metrics packets sent to the server. Type: <code>int</code> Required: No

ProcessParameters

This data type contains the set of parameters sent to Amazon GameLift Servers in a [ProcessReady\(\)](#).

Properties	Description
LogParameters	An object with directory paths to files that are generated during a game session. Amazon

	GameLift Servers copies and stores the files for future access. Type: <code>Aws::GameLift::Server::</code> LogParameters Required: No
OnHealthCheck	The callback function that Amazon GameLift Servers invokes to request a health status report from the server process. Amazon GameLift Servers calls this function every 60 seconds and waits 60 seconds for a response. The server process returns TRUE if healthy, FALSE if not healthy. If no response is returned, Amazon GameLift Servers records the server process as not healthy. Type: <code>std::function<bool(>> onHealthCheck</code> Required: No
OnProcessTerminate	The callback function that Amazon GameLift Servers invokes to force the server process to shut down. After calling this function, Amazon GameLift Servers waits 5 minutes for the server process to shut down and respond with a ProcessEnding() call before it shuts down the server process. Type: <code>std::function<void(>> onProcessTerminate</code> Required: Yes

OnStartGameSession

The callback function that Amazon GameLift Servers invokes to activate a new game session. Amazon GameLift Servers calls this function in response to a client request [CreateGameSession](#). The callback function passes a [GameSession](#) object.

Type: `const std::function<void (Aws::GameLift::Model::GameSession)> onStartGameSession`

Required: Yes

OnUpdateGameSession

The callback function that Amazon GameLift Servers invokes to pass an updated game session object to the server process. Amazon GameLift Servers calls this function when a match backfill request has been processed to provide updated matchmaker data. It passes a [GameSession](#) object, a status update (`updateReason`), and the match backfill ticket ID.

Type: `std::function<void(Aws::GameLift::Server::Model::UpdateGameSession)> onUpdateGameSession`

Required: No

Port

The port number the server process listens on for new player connections. The value must fall into the port range configured for any fleet deploying this game server build. This port number is included in game session and player session objects, which game sessions use when connecting to a server process.

Type: Integer

Required: Yes

UpdateGameSession

This data type updates to a game session object, which includes the reason that the game session was updated and the related backfill ticket ID if backfill is used to fill player sessions in the game session.

Properties	Description
GameSession	A GameSession object. The GameSession object contains properties describing a game session. Type: <code>Aws::GameLift::Server::GameSession</code> Required: Yes
UpdateReason	The reason that the game session is being updated. Type: <code>Aws::GameLift::Server::UpdateReason</code> Required: Yes
BackfillTicketId	The ID of the backfill ticket attempting to update the game session.

Properties	Description
	Type: <code>std::string</code> Required: No

GameSession

This data type provides details of a game session.

Properties	Description
GameSessionId	A unique identifier for the game session. A game session ARN has the following format: <code>arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token> .</code> Type: <code>std::string</code> Required: No
Name	A descriptive label of the game session. Type: <code>std::string</code> Required: No
FleetId	A unique identifier for the fleet that the game session is running on. Type: <code>std::string</code> Required: No
MaximumPlayerSessionCount	The maximum number of player connections to the game session. Type: <code>int</code>

Properties	Description
	Required: No
Port	The port number for the game session. To connect to an Amazon GameLift Servers game server, an app needs both the IP address and port number. Type: <code>in</code> Required: No
IpAddress	The IP address of the game session. To connect to an Amazon GameLift Servers game server, an app needs both the IP address and port number. Type: <code>std::string</code> Required: No
GameSessionData	A set of custom game session properties, formatted as a single string value. Type: <code>std::string</code> Required: No
MatchmakerData	Information about the matchmaking process that was used to create the game session, in JSON syntax, formatted as a string. In addition to the matchmaking configuration used, it contains data on all players assigned to the match, including player attributes and team assignments. Type: <code>std::string</code> Required: No

Properties	Description
GameProperties	A set of custom properties for a game session, formatted as key:value pairs. These properties are passed with a request to start a new game session. Type: <code>std :: vector < GameProperty ></code> Required: No
DnsName	The DNS identifier assigned to the instance that's running the game session. Values have the following format: • TLS-enabled fleets: <code><unique identifier>.<region identifier>.amazon gamelift.com</code> . • Non-TLS-enabled fleets: <code>ec2-<unique identifier>.compute.amazonsys.com</code> . When connecting to a game session that's running on a TLS-enabled fleet, you must use the DNS name, not the IP address. Type: <code>std::string</code> Required: No

ServerParameters

Information that a game server process uses to establish a connection with the Amazon GameLift Servers service. Include these parameters when calling [InitSDK\(\)](#) only if the game server build will be deployed to an Anywhere fleet or a container fleet without the Amazon GameLift Servers Agent. For all other deployment scenarios, call [InitSDK\(\)](#) without parameters.

Properties	Description
websocketUrl	The <code>GameLiftServerSdkEndpoint</code> Amazon GameLift Servers returns when you RegisterCompute for a Amazon GameLift Servers Anywhere compute resource. Type: <code>std::string</code> Required: Yes
processId	A unique identifier registered to the server process hosting your game. Type: <code>std::string</code> Required: Yes
hostId	The <code>HostID</code> is the <code>ComputeName</code> used when you registered your compute. For more information see, RegisterCompute. Type: <code>std::string</code> Required: Yes
fleetId	The unique identifier of the fleet that the compute is registered to. For more information see, RegisterCompute. Type: <code>std::string</code> Required: Yes
authToken	The authentication token generated by Amazon GameLift Servers that authenticates your server to Amazon GameLift Servers. For more information see, GetComputeAuthToken. Type: <code>std::string</code>

Properties	Description
	Required: Yes

StartMatchBackfillRequest

Information used to create a matchmaking backfill request. The game server communicates this information to Amazon GameLift Servers in a [StartMatchBackfill\(\)](#) call.

Properties	Description
GameSessionArn	A unique game session identifier. The API operation GetGameSessionId returns the identifier in ARN format. Type: <code>std::string</code> Required: Yes
MatchmakingConfigurationArn	A unique identifier, in the form of an ARN, for the matchmaker to use for this request. The matchmaker ARN for the original game session is in the game session object in the matchmaker data property. Learn more about matchmaker data in Work with matchmaker data. Type: <code>std::string</code> Required: Yes
Players	A set of data representing all players who are in the game session. The matchmaker uses this information to search for new players who are good matches for the current players. Type: <code>std::vector<Player></code> Required: Yes

Properties	Description
TicketId	A unique identifier for a matchmaking or match backfill request ticket. If you don't provide a value, Amazon GameLift Servers generates one. Use this identifier to track the match backfill ticket status or cancel the request if needed. Type: <code>std::string</code> Required: No

Player

This data type represents a player in matchmaking. When starting a matchmaking request, a player has a player ID, attributes, and possibly latency data. Amazon GameLift Servers adds team information after a match is made.

Properties	Description
LatencyInMS	A set of values expressed in milliseconds that indicate the amount of latency that a player experiences when connected to a location. If this property is used, the player is only matched for locations listed. If a matchmaker has a rule that evaluates player latency, players must report latency to be matched. Type: <code>Dictionary<string,int></code> Required: No
PlayerAttributes	A collection of key:value pairs containing player information for use in matchmaking. Player attribute keys must match the PlayerAttributes used in a matchmaking rule set.

Properties	Description
	For more information about player attributes, see AttributeValue. Type: <code>std::map<std::string, AttributeValue></code> Required: No
PlayerId	A unique identifier for a player. Type: <code>std::string</code> Required: No
Team	The name of the team that the player is assigned to in a match. You define team name in the matchmaking rule set. Type: <code>std::string</code> Required: No

DescribePlayerSessionsRequest

An object that specifies which player sessions to retrieve. The server process provides this information with a [DescribePlayerSessions\(\)](#) call to Amazon GameLift Servers.

Properties	Description
GameSessionId	A unique game session identifier. Use this parameter to request all player sessions for the specified game session. Game session ID format is <code>arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string></code>. The <code>GameSessionId</code> is a custom ID string or a

Properties	Description
	Type: <code>std::string</code> Required: No
PlayerSessionId	The unique identifier for a player session. Use this parameter to request a single specific player session. Type: <code>std::string</code> Required: No
PlayerId	The unique identifier for a player. Use this parameter to request all player sessions for a specific player. See Generate player IDs . Type: <code>std::string</code> Required: No

Properties	Description
PlayerSessionStatusFilter	The player session status to filter results on. Possible player session statuses include: • RESERVED – The player session request was received, but the player hasn't connected to the server process or been validated. • ACTIVE – The player was validated by the server process and is connected. • COMPLETED – The player connection dropped. • TIMEDOUT – A player session request was received, but the player didn't connect or wasn't validated within the time-out limit (60 seconds). Type: <code>std::string</code> Required: No
NextToken	The token indicating the start of the next page of results. To specify the start of the result set, don't provide a value. If you provide a player session ID, this parameter is ignored. Type: <code>std::string</code> Required: No
Limit	The maximum number of results to return. If you provide a player session ID, this parameter is ignored. Type: <code>int</code> Required: No

StopMatchBackfillRequest

Information used to cancel a matchmaking backfill request. The game server communicates this information to Amazon GameLift Servers service in a [StopMatchBackfill\(\)](#) call.

Properties	Description
GameSessionArn	A unique game session identifier of the request being canceled. Type: <code>char[]</code> Required: No
MatchmakingConfigurationArn	A unique identifier of the matchmaker this request was sent to. Type: <code>char[]</code> Required: No
TicketId	A unique identifier of the backfill request ticket to be canceled. Type: <code>char[]</code> Required: No

AttributeValue

Use these values in [Player](#) attribute key-value pairs. This object lets you specify an attribute value using any of the valid data types: string, number, string array, or data map. Each `AttributeValue` object must use exactly one of the available properties: S, N, SL, or SDM.

Properties	Description
AttrType	Specifies the type of attribute value. Possible attribute value types include: • NONE

Properties	Description
	• STRING • DOUBLE • STRING_LIST • STRING_DOUBLE_MAP Required: No
S	Represents a string attribute value. Type: <code>std::string</code> Required: No
N	Represents a numeric attribute value. Type: <code>double</code> Required: No
SL	Represents an array of string attribute values. Type: <code>std::vector<std::string></code> Required: No
SDM	Represents a dictionary of string keys and double values. Type: <code>std::map<std::string, double></code> Required: No

GetFleetRoleCredentialsRequest

This data type gives the game server limited access to your other AWS resources. For more information see, [Set up an IAM service role for Amazon GameLift Servers](#).

Properties	Description
RoleArn	The Amazon Resource Name (ARN) of the service role that extends limited access to your AWS resources. Type: <code>std::string</code> Required: No
RoleSessionName	The role session name that you can use to uniquely identify an AWS Security Token Service AssumeRole session. This name is exposed in audit logs such as those in CloudTrail. Type: <code>std::string</code> Required: No

AwsLongOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: <code>long</code> Required: No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. Type: <code>long&&</code> Required: No
Success	Whether the action was successful or not.

Properties	Description
	Type: <code>bool</code> Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "GameLiftError" Required: No

AwsStringOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: <code>std::string</code> Required: No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. Type: <code>long&&</code> Required: No
Success	Whether the action was successful or not. Type: <code>bool</code> Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "GameLiftError"

Properties	Description
	Required: No

DescribePlayerSessionsOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: the section called “DescribePlayerSessionsResult” Required: No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. Type: <code>Aws::GameLift::Server::Model::DescribePlayerSessionsResult&&</code> Required: No
Success	Whether the action was successful or not. Type: <code>bool</code> Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called “GameLiftError” Required: No

DescribePlayerSessionsResult

A collection of objects containing properties for each player session that matches the request.

Properties	Description
NextToken	A token that indicates the start of the next sequential page of results. Use the token that is returned with a previous call to this operation. To start at the beginning of the result set, do not specify a value. If a player session ID is specified, this parameter is ignored. Type: <code>std::string</code> Required: Yes
PlayerSessions	Type: <code>IList<the section called "PlayerSession" ></code> Required:
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. Type: <code>std::string&&</code> Required: No
Success	Whether the action was successful or not. Type: <code>bool</code> Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "GameLiftError" Required: No

GenericOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "GameLiftError" Required: No

GenericOutcomeCallable

This data type is an asynchronous generic outcome. It has the following properties:

Properties	Description
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "GameLiftError" Required: No

PlayerSession

This data type represents a player session that Amazon GameLift Servers passes to the game server. For more information, see [PlayerSession](#).

Properties	Description
CreationTime	Type: long Required: No
FleetId	Type: std::string Required: No
GameSessionId	Type: std::string Required: No
IpAddress	Type: std::string Required: No
PlayerData	Type: std::string Required: No
PlayerId	Type: std::string Required: No
PlayerSessionId	Type: std::string Required: No
Port	Type: int Required: No
Status	Player session status to filter results on. When a PlayerSessionId or PlayerId is provided, then the PlayerSessionStatusFilter has no effect on the response. Type: A PlayerSessionStatus enum. Possible values include the following: • ACTIVE

Properties	Description
	• COMPLETED • NOT_SET • RESERVED • TIMEDOUT Required: No
TerminationTime	Type: long Required: No
DnsName	Type: std::string Required: No

StartMatchBackfillOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: the section called “StartMatchBackfillResult” Required: No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. Type: StartMatchBackfillResult&& Required: No
Success	Whether the action was successful or not. Type: bool

Properties	Description
	Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "GameLiftError" Required: No

StartMatchBackfillResult

This data type results from an action and produces an object with the following properties:

Properties	Description
TicketId	A unique identifier for a matchmaking ticket. If no ticket ID is specified here, Amazon GameLift Servers will generate one in the form of a UUID. Use this identifier to track the match backfill ticket status and retrieve match results. Type: <code>std::string</code> Required: No

GetComputeCertificateOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: the section called "GetComputeCertificateResult" Required: No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object.

Properties	Description
	Type: <code>Aws::GameLift::Server::Model::GetComputeCertificateResult</code> Required: No
Success	Whether the action was successful or not. Type: <code>bool</code> Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "GameLiftError" Required: No

GetComputeCertificateResult

The path to the TLS certificate on your compute and the compute's host name.

Properties	Description
CertificatePath	The path to the TLS certificate on your compute resource. When using an Amazon GameLift Servers managed fleet, this path contains: <code>certificate.pem</code> : The end-user certificate. The full certificate chain is the combination of <code>certificateChain.pem</code> appended to this certificate. <code>certificateChain.pem</code> : The certificate chain that contains the root certificate and intermediate certificates. <code>rootCertificate.pem</code> : The root certificate. <code>privateKey.pem</code> : The private key for the end-user certificate.

Properties	Description
	Type: <code>std::string</code> Required: No
ComputeName	The name of your compute resource. Type: <code>std::string</code> Required: No

GetFleetRoleCredentialsOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: the section called "GetFleetRoleCredentialsResult" Required: No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. Type: <code>Aws::GameLift::Server::Model::GetFleetRoleCredentialsResult</code> Required: No
Success	Whether the action was successful or not. Type: <code>bool</code> Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "GameLiftError"

Properties	Description
	Required: No

GetFleetRoleCredentialsResult

Properties	Description
AccessKeyId	The access key ID to authenticate and provide access to your AWS resources. Type: string Required: No
AssumedRoleId	The ID of the user that the service role belongs to. Type: string Required: No
AssumedRoleUserArn	The Amazon Resource Name (ARN) of the user that the service role belongs to. Type: string Required: No
Expiration	The amount of time until your session credentials expire. Type: DateTime Required: No
SecretAccessKey	The secret access key ID for authentication. Type: string Required: No

Properties	Description
SessionToken	A token to identify the current active session interacting with your AWS resources. Type: <code>string</code> Required: No
Success	Whether the action was successful or not. Type: <code>bool</code> Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called “GameLiftError” Required: No

ListContainersNetworkInfoOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: the section called “ListContainersNetworkInfoResult” Required: No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. Type: <code>Aws::GameLift::Server::Model::ListContainersNetworkInfoResult&&</code> Required: No

Properties	Description
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called “GameLiftError” Required: No

ListContainersNetworkInfoResult

Network information for all containers running on the same instance as the calling game server process.

Properties	Description
ContainersNetworkInfo	The list of network information for each container running on the instance. Type: std::vector< the section called “ContainerNetworkInfo” > when GAMELIFT_USE_STD is defined; otherwise a fixed array with an accessor named GetContainersNetworkInfoCount(). Required: No

ContainerNetworkInfo

Network information for a single container running on the instance.

Properties	Description
ContainerName	The name of the container, as defined in the container group definition.

Properties	Description
	Type: <code>std::string</code> when <code>GAMELIFT_USE_STD</code> is defined; otherwise <code>char[]</code>. Required: No
ContainerId	The unique identifier of the container. Type: <code>std::string</code> when <code>GAMELIFT_USE_STD</code> is defined; otherwise <code>char[]</code>. Required: No
IpAddress	The container's local IPv4 address on the Docker bridge network. Type: <code>std::string</code> when <code>GAMELIFT_USE_STD</code> is defined; otherwise <code>char[]</code>. Required: No
ContainerGroupType	The type of container group that the container belongs to. Type: A <code>ContainerGroupType</code> enum. Required: No

InitSDKOutcome

Note

`InitSDKOutcome` is returned only when you build the SDK with the `std` flag. If you build with the `nostd` flag, then [the section called "GenericOutcome"](#) is returned instead.

Properties	Description
Success	Whether the action was successful or not.

Properties	Description
	Type: <code>bool</code> Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called “GameLiftError” Required: No

GameLiftError

Properties	Description
ErrorType	The type of error. Type: A <code>GameLiftErrorType</code> enum . Required: No
ErrorMessage	The error message. Type: <code>std::string</code> Required: No

Enums

Enums defined for the server SDK for Amazon GameLift Servers (C++) are defined as follows:

GameLiftErrorType

String value indicating the error type. Valid values include:

- **BAD_REQUEST_EXCEPTION**
- **GAMESESSION_ID_NOT_SET** – The game session ID has not been set.
- **INTERNAL_SERVICE_EXCEPTION**
- **UNSUPPORTED_COMPUTE_TYPE_EXCEPTION** – The API that was called was unsupported on the compute type.
- **LOCAL_CONNECTION_FAILED** – The local connection to Amazon GameLift Servers failed.
- **NETWORK_NOT_INITIALIZED** – The network has not been initialized.
- **SERVICE_CALL_FAILED** – A call to an AWS service has failed.
- **WEBSOCKET_CONNECT_FAILURE**
- **WEBSOCKET_CONNECT_FAILURE_FORBIDDEN**
- **WEBSOCKET_CONNECT_FAILURE_INVALID_URL**
- **WEBSOCKET_CONNECT_FAILURE_TIMEOUT**
- **ALREADY_INITIALIZED** – The Amazon GameLift Servers Server or Client has already been initialized with `Initialize()`.
- **FLEET_MISMATCH** – The target fleet does not match the fleet of a `gameSession` or `playerSession`.
- **GAMELIFT_CLIENT_NOT_INITIALIZED** – The Amazon GameLift Servers client has not been initialized.
- **GAMELIFT_SERVER_NOT_INITIALIZED** – The Amazon GameLift Servers server has not been initialized.
- **GAME_SESSION_ENDED_FAILED** – the server SDK for Amazon GameLift Servers could not contact the service to report the game session ended.
- **GAME_SESSION_NOT_READY** – The Amazon GameLift Servers Server Game Session was not activated.
- **GAME_SESSION_READY_FAILED** – the server SDK for Amazon GameLift Servers could not contact the service to report the game session is ready.
- **INITIALIZATION_MISMATCH** – A client method was called after `Server::Initialize()`, or vice versa.
- **NOT_INITIALIZED** – The Amazon GameLift Servers Server or Client has not been initialized with `Initialize()`.
- **NO_TARGET_ALIASID_SET** – A target `aliasId` has not been set.
- **NO_TARGET_FLEET_SET** – A target fleet has not been set.

- **PROCESS_ENDING_FAILED** – the server SDK for Amazon GameLift Servers could not contact the service to report the process is ending.
- **PROCESS_NOT_ACTIVE** – The server process is not yet active, not bound to a GameSession, and cannot accept or process PlayerSessions.
- **PROCESS_NOT_READY** – The server process is not yet ready to be activated.
- **PROCESS_READY_FAILED** – the server SDK for Amazon GameLift Servers could not contact the service to report the process is ready.
- **SDK_VERSION_DETECTION_FAILED** – SDK version detection failed.
- **STX_CALL_FAILED** – A call to the XStx server backend component has failed.
- **STX_INITIALIZATION_FAILED** – The XStx server backend component has failed to initialize.
- **UNEXPECTED_PLAYER_SESSION** – An unregistered player session was encountered by the server.
- **WEBSOCKET_CONNECT_FAILURE**
- **WEBSOCKET_CONNECT_FAILURE_FORBIDDEN**
- **WEBSOCKET_CONNECT_FAILURE_INVALID_URL**
- **WEBSOCKET_CONNECT_FAILURE_TIMEOUT**
- **WEBSOCKET_RETRIABLE_SEND_MESSAGE_FAILURE** – Retriable failure to send a message to the GameLift Service WebSocket.
- **WEBSOCKET_SEND_MESSAGE_FAILURE** – Failure to send a message to the GameLift Service WebSocket.
- **MATCH_BACKFILL_REQUEST_VALIDATION** – Validation of the request failed.
- **PLAYER_SESSION_REQUEST_VALIDATION** – Validation of the request failed.

ContainerGroupType

The type of container group that a container belongs to. Valid values include:

- **GAME_SERVER** – A game server replica container group. An instance can have multiple game server container groups.
- **PER_INSTANCE** – A per-instance daemon container group. An instance has exactly one per-instance container group.

PlayerSessionCreationPolicy

String value indicating whether the game session accepts new players. Valid values include:

- **ACCEPT_ALL** – Accept all new player sessions.
- **DENY_ALL** – Deny all new player sessions.
- **NOT_SET** – The game session is not set to accept or deny new player sessions.

[C++ server SDK 5.x for Amazon GameLift Servers -- Data types](#)

Topics

- [GetSdkVersion\(\)](#)
- [InitMetrics\(\)](#)
- [InitSDK\(\)](#)
- [InitSDK\(\)](#)
- [ProcessReady\(\)](#)
- [ProcessReadyAsync\(\)](#)
- [ProcessEnding\(\)](#)
- [ActivateGameSession\(\)](#)
- [UpdatePlayerSessionCreationPolicy\(\)](#)
- [GetGameSessionId\(\)](#)
- [GetTerminationTime\(\)](#)
- [AcceptPlayerSession\(\)](#)
- [RemovePlayerSession\(\)](#)
- [DescribePlayerSessions\(\)](#)
- [StartMatchBackfill\(\)](#)
- [StopMatchBackfill\(\)](#)
- [GetComputeCertificate\(\)](#)
- [GetFleetRoleCredentials\(\)](#)
- [ListContainersNetworkInfo\(\)](#)
- [Destroy\(\)](#)

GetSdkVersion()

Returns the current version number of the SDK built into the server process.

Syntax

```
Aws::GameLift::AwsStringOutcome Server::GetSdkVersion();
```

Return value

If successful, returns the current SDK version as an [the section called “AwsStringOutcome”](#) object. The returned object includes the version number (example 5.0.0). If not successful, returns an error message.

Example

```
Aws::GameLift::AwsStringOutcome SdkVersionOutcome =  
    Aws::GameLift::Server::GetSdkVersion();
```

InitMetrics()

Initializes the metrics system for collecting and reporting server performance data. For best results, call this method before [InitSDK\(\)](#) to enable metrics collection during SDK initialization.

Syntax

```
Aws::GameLift::GenericOutcome InitMetrics();  
Aws::GameLift::GenericOutcome InitMetrics(const  
    Aws::GameLift::Server::MetricsParameters &metricsParameters);
```

Parameters

[MetricsParameters](#) (optional)

Configuration parameters for metrics collection. If not provided, uses default values that can be overridden by environment variables.

Return value

If successful, returns a [the section called “GenericOutcome”](#) object indicating success. If not successful, returns an error message.

Example

```
// Initialize with default parameters (uses environment variables if available)
```

```
// Defaults: localhost:8125 for StatsD, localhost:8126 for crash reporter
// FlushInterval: 10000ms, MaxPacketSize: 512 bytes
Aws::GameLift::GenericOutcome outcome = Aws::GameLift::Server::InitMetrics();
if (outcome.IsSuccess()) {
    // Metrics system initialized successfully
}

// Initialize with custom parameters
Aws::GameLift::Server::MetricsParameters metricsParams("localhost", 8125, "crash-host",
    8126, 5000, 1024);
Aws::GameLift::GenericOutcome customOutcome =
    Aws::GameLift::Server::InitMetrics(metricsParams);
if (customOutcome.IsSuccess()) {
    // Metrics system initialized with custom parameters
}
```

InitSDK()

Initializes the Amazon GameLift Servers SDK. Call this method on launch before any other initialization steps related to Amazon GameLift Servers. This action reads server parameters from the host environment to set up communication between the game server process and the Amazon GameLift Servers service. It uses an idempotency token, so you safely retry this call when it fails.

If the game server build will be deployed without the Amazon GameLift Servers Agent to a Amazon GameLift Servers Anywhere fleet or container fleet, call [InitSDK\(\)](#) and specify a set of server parameters.

Syntax

```
Server::InitSDKOutcome Server::initSdkOutcome = InitSDK();
```

Return value

Returns an [the section called "InitSDKOutcome"](#) object that indicates whether the server process is ready to call [ProcessReady\(\)](#).

Example

```
//Call InitSDK to establish a local connection with the Amazon GameLift Servers Agent
//to enable further communication.
Aws::GameLift::Server::InitSDKOutcome initSdkOutcome =
    Aws::GameLift::Server::InitSDK();
```

InitSDK()

Initializes the Amazon GameLift Servers SDK. Call this method on launch before any other initialization steps related to Amazon GameLift Servers. This action requires a set of server parameters to set up communication between the game server process and the Amazon GameLift Servers service. It uses an idempotency token, so you safely retry this call when it fails.

If the game server build will be deployed to an Amazon GameLift Servers managed EC2 fleet or to an Amazon GameLift Servers Anywhere fleet or container fleet with the Amazon GameLift Servers Agent, call [InitSDK\(\)](#) without server parameters.

Syntax

```
Server::InitSDKOutcome Server::initSdkOutcome = InitSDK(serverParameters);
```

Parameters

[ServerParameters](#)

To initialize a game server on an Amazon GameLift Servers Anywhere fleet, construct a `ServerParameters` object with the following information:

- The URL of the WebSocket used to connect to your game server.
- The ID of the process used to host your game server.
- The ID of the compute hosting your game server processes.
- The ID of the Amazon GameLift Servers fleet containing your Amazon GameLift Servers Anywhere compute.
- The authorization token generated by the Amazon GameLift Servers operation.

Return value

Returns an [the section called “InitSDKOutcome”](#) object that indicates whether the server process is ready to call [ProcessReady\(\)](#).

Note

If calls to `InitSDK()` are failing for game builds deployed to Anywhere fleets, check the `ServerSdkVersion` parameter used when creating the build resource. You must explicitly set this value to the server SDK version in use. The default value for this parameter is 4.x,

which is not compatible. To resolve this issue, create a new build and deploy it to a new fleet.

Example

Amazon GameLift Servers Anywhere example

```
//Define the server parameters
std::string websocketUrl = "wss://us-west-1.api.amazongamelift.com";
std::string processId = "PID1234";
std::string fleetId = "arn:aws:gamelift:us-west-1:111122223333:fleet/
fleet-9999ffff-88ee-77dd-66cc-5555bbbb44aa";
std::string hostId = "HardwareAnywhere";
std::string authToken = "1111aaaa-22bb-33cc-44dd-5555eeee66ff";
Aws::GameLift::Server::Model::ServerParameters serverParameters =
    Aws::GameLift::Server::Model::ServerParameters(websocketUrl, authToken, fleetId,
    hostId, processId);

//Call InitSDK to establish a local connection with the Amazon GameLift Servers Agent
to enable further communication.
Aws::GameLift::Server::InitSDKOutcome initSdkOutcome =
    Aws::GameLift::Server::InitSDK(serverParameters);
```

ProcessReady()

Notifies Amazon GameLift Servers that the server process is ready to host game sessions. Call this method after invoking [InitSDK\(\)](#). This method should be called only once per process.

Syntax

```
GenericOutcome ProcessReady(const Aws::GameLift::Server::ProcessParameters
&processParameters);
```

Parameters

processParameters

A [ProcessParameters](#) object communicating the following information about the server process:

- Names of callback methods implemented in the game server code that the Amazon GameLift Servers service invokes to communicate with the server process.

- Port number that the server process is listening on.
- Path to any game session-specific files that you want Amazon GameLift Servers to capture and store.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example illustrates both the [ProcessReady\(\)](#) call and delegate function implementations.

```
// Set parameters and call ProcessReady
std::string serverLog("serverOut.log");           // Example of a log file written by the
game server
std::vector<std::string> logPaths;
logPaths.push_back(serverLog);
int listenPort = 9339;

Aws::GameLift::Server::ProcessParameters processReadyParameter =
    Aws::GameLift::Server::ProcessParameters(
        std::bind(&Server::onStartGameSession, this, std::placeholders::_1),
        std::bind(&Server::onProcessTerminate, this),
        std::bind(&Server::OnHealthCheck, this),
        std::bind(&Server::OnUpdateGameSession, this),
        listenPort,
        Aws::GameLift::Server::LogParameters(logPaths)
    );

Aws::GameLift::GenericOutcome outcome =
    Aws::GameLift::Server::ProcessReady(processReadyParameter);

// Implement callback functions
void Server::onStartGameSession(Aws::GameLift::Model::GameSession myGameSession)
{
    // game-specific tasks when starting a new game session, such as loading map
    GenericOutcome outcome =
        Aws::GameLift::Server::ActivateGameSession (maxPlayers);
}

void Server::onProcessTerminate()
{

```

```
// game-specific tasks required to gracefully shut down a game session,  
// such as notifying players, preserving game state data, and other cleanup  
GenericOutcome outcome = Aws::GameLift::Server::ProcessEnding();  
}  
  
bool Server::onHealthCheck()  
{  
    bool health;  
    // complete health evaluation within 60 seconds and set health  
    return health;  
}
```

ProcessReadyAsync()

Notifies the Amazon GameLift Servers service that the server process is ready to host game sessions. This method should be called after the server process is ready to host a game session. The parameters specify the callback function names for Amazon GameLift Servers to call in certain circumstances. Game server code must implement these functions.

This call is asynchronous. To make a synchronous call, use [ProcessReady\(\)](#). See [Initialize the server process](#) for more details.

Syntax

```
GenericOutcomeCallable ProcessReadyAsync(  
    const Aws::GameLift::Server::ProcessParameters &processParameters);
```

Parameters

processParameters

A [ProcessParameters](#) object communicating the following information about the server process:

- Names of callback methods implemented in the game server code that the Amazon GameLift Servers service invokes to communicate with the server process.
- Port number that the server process is listening on.
- Path to any game session-specific files that you want Amazon GameLift Servers to capture and store.

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
// Set parameters and call ProcessReady
std::string serverLog("serverOut.log");           // This is an example of a log file
                                                    written by the game server
std::vector<std::string> logPaths;
logPaths.push_back(serverLog);
int listenPort = 9339;

Aws::GameLift::Server::ProcessParameters processReadyParameter =
    Aws::GameLift::Server::ProcessParameters(std::bind(&Server::onStartGameSession, this,
std::placeholders::_1),
    std::bind(&Server::onProcessTerminate, this), std::bind(&Server::OnHealthCheck,
this),
    std::bind(&Server::OnUpdateGameSession, this), listenPort,
    Aws::GameLift::Server::LogParameters(logPaths));

Aws::GameLift::GenericOutcomeCallable outcome =
    Aws::GameLift::Server::ProcessReadyAsync(processReadyParameter);

// Implement callback functions
void onStartGameSession(Aws::GameLift::Model::GameSession myGameSession)
{
    // game-specific tasks when starting a new game session, such as loading map
    GenericOutcome outcome = Aws::GameLift::Server::ActivateGameSession (maxPlayers);
}

void onProcessTerminate()
{
    // game-specific tasks required to gracefully shut down a game session,
    // such as notifying players, preserving game state data, and other cleanup
    GenericOutcome outcome = Aws::GameLift::Server::ProcessEnding();
}

bool onHealthCheck()
{
    // perform health evaluation and complete within 60 seconds
    return health;
}
```

ProcessEnding()

Notifies Amazon GameLift Servers that the server process is terminating. Call this method after all other cleanup tasks (including shutting down the active game session) and before terminating the process. Depending on the result of `ProcessEnding()`, the process exits with success (0) or error (-1) and generates a fleet event. If the process terminates with an error, the fleet event generated is `SERVER_PROCESS_TERMINATED_UNHEALTHY`.

Syntax

```
Aws::GameLift::GenericOutcome processEndingOutcome =  
    Aws::GameLift::Server::ProcessEnding();
```

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example calls `ProcessEnding()` and `Destroy()` before terminating the server process with a success or error exit code.

```
Aws::GameLift::GenericOutcome processEndingOutcome =  
    Aws::GameLift::Server::ProcessEnding();  
Aws::GameLift::Server::Destroy();  
  
// Exit the process with success or failure  
if (processEndingOutcome.IsSuccess()) {  
    exit(0);  
}  
else {  
    cout << "ProcessEnding() failed. Error: " <<  
        processEndingOutcome.GetError().GetErrorMessage();  
    exit(-1);  
}
```

ActivateGameSession()

Notifies Amazon GameLift Servers that the server process has activated a game session and is now ready to receive player connections. This action should be called as part of the `onStartGameSession()` callback function, after all game session initialization.

Syntax

```
Aws::GameLift::GenericOutcome activateGameSessionOutcome =  
    Aws::GameLift::Server::ActivateGameSession();
```

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example shows `ActivateGameSession()` called as part of the `onStartGameSession()` delegate function.

```
void onStartGameSession(Aws::GameLift::Model::GameSession myGameSession)  
{  
    // game-specific tasks when starting a new game session, such as loading map  
    GenericOutcome outcome = Aws::GameLift::Server::ActivateGameSession();  
}
```

UpdatePlayerSessionCreationPolicy()

Updates the current game session's ability to accept new player sessions. A game session can be set to either accept or deny all new player sessions.

Syntax

```
GenericOutcome  
    UpdatePlayerSessionCreationPolicy(Aws::GameLift::Model::PlayerSessionCreationPolicy  
        newPlayerSessionPolicy);
```

Parameters

playerCreationSessionPolicy

Type: `PlayerSessionCreationPolicy` [enum](#) value.

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example sets the current game session's join policy to accept all players.

```
Aws::GameLift::GenericOutcome outcome =  
  
    Aws::GameLift::Server::UpdatePlayerSessionCreationPolicy(Aws::GameLift::Model::PlayerSessionCr
```

GetGameSessionId()

Retrieves the ID of the game session hosted by the active server process.

For idle processes that aren't activated with a game session, the call returns a [the section called "GameLiftError"](#).

Syntax

```
AwsStringOutcome GetGameSessionId()
```

Parameters

This action has no parameters.

Return value

If successful, returns the game session ID as an [the section called "AwsStringOutcome"](#) object. If not successful, returns an error message.

For idle processes that aren't activated with a game session, the call returns Success=True and GameSessionId="".

Example

```
Aws::GameLift::AwsStringOutcome sessionIdOutcome =  
    Aws::GameLift::Server::GetGameSessionId();
```

GetTerminationTime()

Returns the time that a server process is scheduled to be shut down, if a termination time is available. A server process takes action after receiving an onProcessTerminate() callback

from Amazon GameLift Servers. Amazon GameLift Servers calls `onProcessTerminate()` for the following reasons:

- When the server process has reported poor health or has not responded to Amazon GameLift Servers.
- When terminating the instance during a scale-down event.
- When an instance is terminated due to a [spot-instance interruption](#).

Syntax

```
AwsDateTimeOutcome GetTerminationTime()
```

Return value

If successful, returns the termination time as an `AwsDateTimeOutcome` object. The value is the termination time, expressed in elapsed ticks since `0001 00:00:00`. For example, the date time value `2020-09-13 12:26:40 -000Z` is equal to `637355968000000000` ticks. If no termination time is available, returns an error message.

If the process hasn't received a `ProcessParameters.OnProcessTerminate()` callback, an error message is returned. For more information about shutting down a server process, see [Respond to a server process shutdown notification](#).

Example

```
Aws::GameLift::AwsLongOutcome TermTimeOutcome =  
    Aws::GameLift::Server::GetTerminationTime();
```

AcceptPlayerSession()

Notifies Amazon GameLift Servers that a player with the specified player session ID has connected to the server process and needs validation. Amazon GameLift Servers verifies that the player session ID is valid. After the player session is validated, Amazon GameLift Servers changes the status of the player slot from `RESERVED` to `ACTIVE`.

Syntax

```
GenericOutcome AcceptPlayerSession(String playerId)
```

Parameters

playerSessionId

Unique ID issued by Amazon GameLift Servers when a new player session is created.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example handles a connection request that includes validating and rejecting non-valid player session IDs.

```
void ReceiveConnectingPlayerSessionID (Connection& connection, const std::string&
playerSessionId)
{
    Aws::GameLift::GenericOutcome connectOutcome =
    Aws::GameLift::Server::AcceptPlayerSession(playerSessionId);
    if(connectOutcome.IsSuccess())
    {
        connectionToSessionMap.emplace(connection, playerSessionId);
        connection.Accept();
    }
    else
    {
        connection.Reject(connectOutcome.GetError().GetMessage());
    }
}
```

RemovePlayerSession()

Notifies Amazon GameLift Servers that a player has disconnected from the server process. In response, Amazon GameLift Servers changes the player slot to available.

Syntax

```
GenericOutcome RemovePlayerSession(String playerSessionId)
```

Parameters

playerSessionId

Unique ID issued by Amazon GameLift Servers when a new player session is created.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
Aws::GameLift::GenericOutcome disconnectOutcome =  
    Aws::GameLift::Server::RemovePlayerSession(playerSessionId);
```

DescribePlayerSessions()

Retrieves player session data which includes settings, session metadata, and player data. Use this method to get information about the following:

- A single player session
- All player sessions in a game session
- All player sessions associated with a single player ID

Syntax

```
DescribePlayerSessionsOutcome DescribePlayerSessions(DescribePlayerSessionsRequest  
    describePlayerSessionsRequest)
```

Parameters

DescribePlayerSessionsRequest

A [the section called “DescribePlayerSessionsRequest”](#) object that describes which player sessions to retrieve.

Return value

If successful, returns a [the section called “DescribePlayerSessionsOutcome”](#) object containing a set of player session objects that fit the request parameters.

Example

This example requests all player sessions actively connected to a specified game session. By omitting *NextToken* and setting the *Limit* value to 10, Amazon GameLift Servers returns the first 10 player session records matching the request.

```
// Set request parameters
Aws::GameLift::Server::Model::DescribePlayerSessionsRequest request;
request.SetPlayerSessionStatusFilter(Aws::GameLift::Server::Model::PlayerSessionStatusMapper::G
request.SetLimit(10);
request.SetGameSessionId("the game session ID");    // can use GetGameSessionId()

// Call DescribePlayerSessions
Aws::GameLift::DescribePlayerSessionsOutcome playerSessionsOutcome =
    Aws::GameLift::Server::DescribePlayerSessions(request);
```

StartMatchBackfill()

Sends a request to find new players for open slots in a game session created with FlexMatch. For more information, see [FlexMatch backfill feature](#).

This action is asynchronous. If new players are matched, Amazon GameLift Servers delivers updated matchmaker data using the callback function `OnUpdateGameSession()`.

A server process can have only one active match backfill request at a time. To send a new request, first call [StopMatchBackfill\(\)](#) to cancel the original request.

Syntax

```
StartMatchBackfillOutcome StartMatchBackfill (StartMatchBackfillRequest
startBackfillRequest);
```

Parameters

[StartMatchBackfillRequest](#)

A `StartMatchBackfillRequest` object that communicates the following information:

- A ticket ID to assign to the backfill request. This information is optional; if no ID is provided, Amazon GameLift Servers will generate one.
- The matchmaker to send the request to. The full configuration ARN is required. This value is in the game session's matchmaker data.

- The ID of the game session to backfill.
- The available matchmaking data for the game session's current players.

Return value

Returns a [the section called “StartMatchBackfillOutcome”](#) object with the match backfill ticket ID, or failure with an error message.

Example

```
// Build a backfill request
std::vector<Player> players;
Aws::GameLift::Server::Model::StartMatchBackfillRequest startBackfillRequest;
startBackfillRequest.SetTicketId("1111aaaa-22bb-33cc-44dd-5555eeee66ff"); // optional,
    autogenerated if not provided
startBackfillRequest.SetMatchmakingConfigurationArn("arn:aws:gamelift:us-
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig"); //from the game
    session matchmaker data
startBackfillRequest.SetGameSessionArn("the game session ARN");           // can use
    GetGameSessionId()
startBackfillRequest.SetPlayers(players);                                 // from the
    game session matchmaker data

// Send backfill request
Aws::GameLift::StartMatchBackfillOutcome backfillOutcome =
    Aws::GameLift::Server::StartMatchBackfill(startBackfillRequest);

// Implement callback function for backfill
void Server::OnUpdateGameSession(Aws::GameLift::Server::Model::GameSession gameSession,
    Aws::GameLift::Server::Model::UpdateReason updateReason, std::string backfillTicketId)
{
    // handle status messages
    // perform game-specific tasks to prep for newly matched players
}
```

StopMatchBackfill()

Cancels an active match backfill request. For more information, see [FlexMatch backfill feature](#).

Syntax

```
GenericOutcome StopMatchBackfill (StopMatchBackfillRequest stopBackfillRequest);
```

Parameters

StopMatchBackfillRequest

A StopMatchBackfillRequest object identifying the matchmaking ticket to cancel:

- The ticket ID assigned to the backfill request.
- The matchmaker the backfill request was sent to.
- The game session associated with the backfill request.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
// Set backfill stop request parameters

Aws::GameLift::Server::Model::StopMatchBackfillRequest stopBackfillRequest;
stopBackfillRequest.SetTicketId("1111aaaa-22bb-33cc-44dd-5555eeee66ff");
stopBackfillRequest.SetGameSessionArn("the game session ARN"); // can use
    GetGameSessionId()
stopBackfillRequest.SetMatchmakingConfigurationArn("arn:aws:gamelift:us-
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig");
// from the game session matchmaker data

Aws::GameLift::GenericOutcome stopBackfillOutcome =
    Aws::GameLift::Server::StopMatchBackfill(stopBackfillRequest);
```

GetComputeCertificate()

Retrieves the path to the TLS certificate used to encrypt the network connection between your Amazon GameLift Servers Anywhere compute resource and Amazon GameLift Servers. You can use the certificate path when you register your compute device to a Amazon GameLift Servers Anywhere fleet. For more information see, [RegisterCompute](#).

Syntax

```
GetComputeCertificateOutcome Server::GetComputeCertificate()
```

Return value

Returns a [the section called "GetComputeCertificateOutcome"](#).

Example

```
Aws::GameLift::GetComputeCertificateOutcome certificate =  
    Aws::GameLift::Server::GetComputeCertificate();
```

GetFleetRoleCredentials()

Retrieves IAM role credentials that authorize Amazon GameLift Servers to interact with other AWS services. For more information, see [Connect your Amazon GameLift Servers hosted game server to other AWS resources](#).

Syntax

```
GetFleetRoleCredentialsOutcome GetFleetRoleCredentials(GetFleetRoleCredentialsRequest  
    request);
```

Parameters

[GetFleetRoleCredentialsRequest](#)

Return value

Returns a [the section called "GetFleetRoleCredentialsOutcome"](#) object.

Example

```
// form the fleet credentials request  
Aws::GameLift::Server::Model::GetFleetRoleCredentialsRequest  
    getFleetRoleCredentialsRequest;  
getFleetRoleCredentialsRequest.SetRoleArn("arn:aws:iam::123456789012:role/service-role/  
exampleGameLiftAction");  
  
Aws::GameLift::GetFleetRoleCredentialsOutcome credentials =  
    Aws::GameLift::Server::GetFleetRoleCredentials(getFleetRoleCredentialsRequest);
```

This example shows the use of the optional RoleSessionName value to assign a name to the credentials session for auditing purposes. If you don't provide a role session name, the default value "*[fleet-id]-[host-id]*" is used.

```
// form the fleet credentials request
Aws::GameLift::Server::Model::GetFleetRoleCredentialsRequest
    getFleetRoleCredentialsRequest;
getFleetRoleCredentialsRequest.SetRoleArn("arn:aws:iam::123456789012:role/service-role/
exampleGameLiftAction");
getFleetRoleCredentialsRequest.SetRoleSessionName("MyFleetRoleSession");

Aws::GameLift::GetFleetRoleCredentialsOutcome credentials =
    Aws::GameLift::Server::GetFleetRoleCredentials(getFleetRoleCredentialsRequest);
```

ListContainersNetworkInfo()

Retrieves network information for all containers running on the same instance, including each container's name, ID, local IP address, and container group type. Use this information to enable game server processes to discover and communicate with other containers that are running on the same instance.

This action is supported only on container fleets. When called from any other compute type, it returns an `UNSUPPORTED_COMPUTE_TYPE_EXCEPTION` error. Amazon GameLift Servers obtains the network information from a discovery server that runs locally on the instance.

Syntax

```
ListContainersNetworkInfoOutcome ListContainersNetworkInfo()
```

Return value

Returns a [the section called "ListContainersNetworkInfoOutcome"](#).

Example

```
Aws::GameLift::ListContainersNetworkInfoOutcome containersNetworkInfo =
    Aws::GameLift::Server::ListContainersNetworkInfo();
```

Destroy()

Frees the Amazon GameLift Servers game server SDK from memory. As a best practice, call this method after `ProcessEnding()` and before terminating the process. If you're using an Anywhere fleet and you're not terminating server processes after every game session, call `Destroy()` and then `InitSDK()` to reinitialize before notifying Amazon GameLift Servers that the process is ready to host a game session with `ProcessReady()`.

Syntax

```
GenericOutcome Aws::GameLift::Server::Destroy();
```

Parameters

There are no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
Aws::GameLift::GenericOutcome processEndingOutcome =
    Aws::GameLift::Server::ProcessEnding();
Aws::GameLift::Server::Destroy();

// Exit the process with success or failure
if (processEndingOutcome.IsSuccess()) {
    exit(0);
}
else {
    cout << "ProcessEnding() failed. Error: " <<
        processEndingOutcome.GetError().GetErrorMessage();
    exit(-1);
}
```

C# server SDK 5.x for Amazon GameLift Servers -- Actions

Use the server SDK 5.x reference to integrate your multiplayer game for hosting with Amazon GameLift Servers. For guidance about the integration process, see [Add Amazon GameLift Servers to your game server with the server SDK](#). If you're using the Amazon GameLift Servers plugin for Unity, see also [Amazon GameLift Servers plugin for Unity \(server SDK 5.x\)](#).

C# server SDK 5.x for Amazon GameLift Servers -- Data types

Use the Amazon GameLift Servers C# server SDK 5.x reference to integrate your multiplayer game for hosting with Amazon GameLift Servers. For guidance about the integration process, see [Add Amazon GameLift Servers to your game server with the server SDK](#). If you're using the Amazon GameLift Servers plugin for Unity, see also [Amazon GameLift Servers plugin for Unity \(server SDK 5.x\)](#).

C# server SDK 5.x for Amazon GameLift Servers -- Actions

Data types

- [LogParameters](#)
- [MetricsParameters](#)
- [ProcessParameters](#)
- [UpdateGameSession](#)
- [GameSession](#)
- [ServerParameters](#)
- [StartMatchBackfillRequest](#)
- [Player](#)
- [DescribePlayerSessionsRequest](#)
- [StopMatchBackfillRequest](#)
- [GetFleetRoleCredentialsRequest](#)
- [AttributeValue](#)
- [AwsStringOutcome](#)
- [GenericOutcome](#)
- [MetricsManagerOutcome](#)
- [DescribePlayerSessionsOutcome](#)
- [DescribePlayerSessionsResult](#)
- [PlayerSession](#)
- [StartMatchBackfillOutcome](#)
- [StartMatchBackfillResult](#)
- [GetComputeCertificateOutcome](#)
- [GetComputeCertificateResult](#)
- [GetFleetRoleCredentialsOutcome](#)
- [GetFleetRoleCredentialsResult](#)
- [ListContainersNetworkInfoOutcome](#)
- [ListContainersNetworkInfoResult](#)
- [ContainerNetworkInfo](#)
- [AwsDateTimeOutcome](#)

- [GameLiftError](#)
- [Enums](#)

LogParameters

Use this data type to identify which files generated during a game session that you want the game server to upload to Amazon GameLift Servers after the game session ends. The game server communicates `LogParameters` to Amazon GameLift Servers in a [ProcessReady\(\)](#) call.

Properties	Description
LogPaths	The list of directory paths to game server log files you want Amazon GameLift Servers to store for future access. The server process generates these files during each game session. You define file paths and names in your game server and store them in the root game build directory. The log paths must be absolute. For example, if your game build stores game session logs in a path like <code>MyGame\sessionLogs\</code>, then the path would be <code>c:\game\MyGame\sessionLogs</code> on a Windows instance. Type: <code>List<String></code> Required: No

MetricsParameters

Use this data type to configure metrics collection and crash reporting for the game server. The game server communicates `MetricsParameters` to Amazon GameLift Servers in an [InitMetrics\(\)](#) call.

Properties	Description

StatsdHost	The hostname or IP address of the StatsD server for metrics collection. Type: String Required: No
StatsdPort	The port number of the StatsD server for metrics collection. Type: Integer Required: No
CrashReporterHost	The hostname or IP address of the crash reporter server. Type: String Required: No
CrashReporterPort	The port number of the crash reporter server. Type: Integer Required: No
FlushIntervalMs	The interval in milliseconds for flushing metrics data to the server. Type: Integer Required: No
MaxPacketSize	The maximum size in bytes for metrics packets sent to the server. Type: Integer Required: No

ProcessParameters

This data type contains the set of parameters sent to Amazon GameLift Servers in a [ProcessReady\(\)](#) call.

Properties	Description
LogParameters	The object with a list of directory paths to game session log files. Type: <code>Aws::GameLift::Server::</code> LogParameters Required: Yes
OnHealthCheck	The name of callback function that Amazon GameLift Servers invokes to request a health status report from the server process. Amazon GameLift Servers calls this function every 60 seconds. After calling this function Amazon GameLift Servers waits 60 seconds for a response, if none is received, Amazon GameLift Servers records the server process as unhealthy. Type: <code>void OnHealthCheckDelegate()</code> Required: Yes
OnProcessTerminate	The name of callback function that Amazon GameLift Servers invokes to force the server process to shut down. After calling this function, Amazon GameLift Servers waits five minutes for the server process to shut down and respond with a ProcessEnding() call before it shuts down the server process. Type: <code>void OnProcessTerminateDelegate()</code>

	Required: Yes
OnStartGameSession	The name of callback function that Amazon GameLift Servers invokes to activate a new game session. Amazon GameLift Servers calls this function in response to the client request CreateGameSession. The callback function takes a GameSession object. Type: void OnStartGameSession Delegate(GameSession) Required: Yes
OnUpdateGameSession	The name of callback function that Amazon GameLift Servers invokes to pass an updated game session object to the server process. Amazon GameLift Servers calls this function when a match backfill request has been processed to provide updated matchmaker data. It passes a GameSession object, a status update (updateReason), and the match backfill ticket ID. Type: void OnUpdateGameSessionDelegate (UpdateGameSession) Required: No

Port

The port number that the server process listens on for new player connections. The value must fall into the port range configured for any fleet deploying this game server build. This port number is included in game session and player session objects, which game sessions use when connecting to a server process.

Type: Integer

Required: Yes

UpdateGameSession

Updated information for a game session object, includes the reason that the game session was updated. If the update is related to a match backfill action, this data type includes the backfill ticket ID.

Properties	Description
GameSession	A GameSession object. The GameSession object contains properties describing a game session. Type: GameSession GameSession() Required: Yes
UpdateReason	The reason that the game session is being updated. Type: UpdateReason UpdateReason() Required: Yes
BackfillTicketId	The ID of the backfill ticket attempting to update the game session.

Properties	Description
	Type: String
	Required: Yes

GameSession

Details of a game session.

Properties	Description
GameSessionId	A unique identifier for the game session. A game session ARN has the following format: <code>arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token> .</code> Type: String Required: No
Name	A descriptive label of the game session. Type: String Required: No
FleetId	A unique identifier for the fleet that the game session is running on. Type: String Required: No
MaximumPlayerSessionCount	The maximum number of player connections to the game session. Type: Integer

Properties	Description
	Required: No
Port	The port number for the game session. To connect to an Amazon GameLift Servers game server, an app needs both the IP address and port number. Type: Integer Required: No
IpAddress	The IP address of the game session. To connect to an Amazon GameLift Servers game server, an app needs both the IP address and port number. Type: String Required: No
GameSessionData	A set of custom game session properties, formatted as a single string value. Type: String Required: No
MatchmakerData	The information about the matchmaking process that was used to create the game session, in JSON syntax, formatted as a string. In addition the matchmaking configuration used, it contains data on all players assigned to the match, including player attributes and team assignments. Type: String Required: No

Properties	Description
GameProperties	A set of custom properties for a game session, formatted as key:value pairs. These properties are passed with a request to start a new game session. Type: Dictionary<string, string> Required: No
DnsName	The DNS identifier assigned to the instance that's running the game session. Values have the following format: • TLS-enabled fleets: <unique identifier>.<region identifier>.amazon.gamelift.com . • Non-TLS-enabled fleets: ec2-<unique identifier>.compute.amazonaws.com . When connecting to a game session that's running on a TLS-enabled fleet, you must use the DNS name, not the IP address. Type: String Required: No

ServerParameters

Information used to maintain the connection between an Amazon GameLift Servers Anywhere server and the Amazon GameLift Servers service. This information is used when launching new server processes with [InitSDK\(\)](#). For servers hosted on Amazon GameLift Servers managed EC2 instances, use an empty object.

Properties	Description
WebSocketUrl	The <code>GameLiftServerSdkEndpoint</code> returned when you <code>RegisterCompute</code> as part of Amazon GameLift Servers Anywhere. Type: String Required: Yes
ProcessId	A unique identifier registered to the server process hosting your game. Type: String Required: Yes
HostId	A unique identifier for the host with the server processes hosting your game. The <code>hostId</code> is the <code>ComputeName</code> used when you registered your compute. For more information see, RegisterCompute Type: String Required: Yes
FleetId	The fleet ID of the fleet that the compute is registered to. For more information see, RegisterCompute. Type: String Required: Yes
AuthToken	The authentication token generated by Amazon GameLift Servers that authenticates your server to Amazon GameLift Servers. For more information see, GetComputeAuthToken.

Properties	Description
	Type: String
	Required: Yes

StartMatchBackfillRequest

Information used to create a matchmaking backfill request. The game server communicates this information to Amazon GameLift Servers in a [StartMatchBackfill\(\)](#) call.

Properties	Description
GameSessionArn	The unique game session identifier. The API operation GetGameSessionId returns the identifier in ARN format. Type: String Required: Yes
MatchmakingConfigurationArn	The unique identifier, in the form of an ARN, for the matchmaker to use for this request. The matchmaker ARN for the original game session is in the game session object in the matchmaker data property. Learn more about matchmaker data in Work with matchmaker data. Type: String Required: Yes
Players	A set of data that represents all players who are currently in the game session. The matchmaker uses this information to search for new players who are good matches for the current players.

Properties	Description
	Type: List<Player> Required: Yes
TicketId	The unique identifier for a matchmaking or match backfill request ticket. If you don't provide a value, Amazon GameLift Servers generates one. Use this identifier to track the match backfill ticket status or cancel the request if needed. Type: String Required: No

Player

Represents a player in matchmaking. When a matchmaking request starts, a player has a player ID, attributes, and possibly latency data. Amazon GameLift Servers adds team information after a match is made.

Properties	Description
LatencyInMS	A set of values expressed in milliseconds, that indicate the amount of latency that a player experiences when connected to a location. If this property is used, the player is only matched for locations listed. If a matchmaker has a rule that evaluates player latency, players must report latency to be matched. Type: Dictionary<string, int> Required: No

Properties	Description
PlayerAttributes	A collection of key:value pairs that contain player information for use in matchmaking. Player attribute keys must match the PlayerAttributes used in a matchmaking rule set. For more information about player attributes, see AttributeValue. Type: Dictionary<string, Attribute Value Required: No
PlayerId	A unique identifier for a player. Type: String Required: No
Team	The name of the team that the player is assigned to in a match. You define team name in the matchmaking rule set. Type: String Required: No

DescribePlayerSessionsRequest

This data type is used to specify which player session(s) to retrieve. It can be used in several ways: (1) provide a PlayerSessionId to request a specific player session; (2) provide a GameSessionId to request all player sessions in the specified game session; or (3) provide a PlayerId to request all player sessions for the specified player. For large collections of player sessions, use the pagination parameters to retrieve results as sequential pages.

Properties	Description
GameSessionId	The unique game session identifier. Use this parameter to request all player sessions for the specified game session. Game session ID format is as follows: <code>arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string></code> . The value of <code><ID string></code> is either a custom ID string (if one was specified when the game session was created) a generated string. Type: String Required: No
PlayerSessionId	The unique identifier for a player session. Type: String Required: No
PlayerId	The unique identifier for a player. See Generate player IDs. Type: String Required: No
PlayerSessionStatusFilter	The player session status to filter results on. Possible player session statuses include the following: RESERVED – The player session request has been received, but the player has not yet connected to the server process and/or been validated.

Properties	Description
	• ACTIVE – The player has been validated by the server process and is currently connected. • COMPLETED – The player connection has been dropped. • TIMEDOUT – A player session request was received, but the player did not connect and/or was not validated within the time-out limit (60 seconds). Type: String Required: No
NextToken	The token indicating the start of the next page of results. To specify the start of the result set, don't provide a value. If you provide a player session ID, this parameter is ignored. Type: String Required: No
Limit	The maximum number of results to return. If you provide a player session ID, this parameter is ignored. Type: int Required: No

StopMatchBackfillRequest

Information used to cancel a matchmaking backfill request. The game server communicates this information to Amazon GameLift Servers service in a [StopMatchBackfill\(\)](#) call.

Properties	Description
GameSessionArn	The unique game session identifier of the request being canceled. Type: string Required: Yes
MatchmakingConfigurationArn	The unique identifier of the matchmaker this request was sent to. Type: string Required: Yes
TicketId	The unique identifier of the backfill request ticket to be canceled. Type: string Required: Yes

GetFleetRoleCredentialsRequest

This data type gives the game server limited access to your other AWS resources. For more information see, [Set up an IAM service role for Amazon GameLift Servers](#).

Properties	Description
RoleArn	The Amazon Resource Name (ARN) of the service role that extends limited access to your AWS resources. Type: string Required: Yes
RoleSessionName	The name of the session that describes the use of the role credentials.

Properties	Description
	Type: string
	Required: No

AttributeValue

Use these values in [Player](#) attribute key-value pairs. This object lets you specify an attribute value using any of the valid data types: string, number, string array, or data map. Each `AttributeValue` object can use only one of the available properties.

Properties	Description
attrType	Specifies the type of attribute value. Type: An <code>AttrType</code> enum value. Required: No
S	Represents a string attribute value. Type: string Required: Yes
N	Represents a numeric attribute value. Type: double Required: Yes
SL	Represents an array of string attribute values. Type: string[] Required: Yes
SDM	Represents a dictionary of string keys and double values. Type: Dictionary<string, double>

Properties	Description
	Required: Yes

AwsStringOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: string Required: No
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "GameLiftError" Required: No

GenericOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Success	Whether the action was successful or not. Type: bool Required: Yes

Properties	Description
Error	The error that occurred if the action was unsuccessful. Type: the section called “GameLiftError” Required: No

MetricsManagerOutcome

Represents the result of an [InitMetrics\(\)](#) call. Contains either a MetricsManager instance on success or error information on failure.

Result	The MetricsManager instance for collecting and reporting metrics. Type: MetricsManager Required: No
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called “GameLiftError” Required: No

DescribePlayerSessionsOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: the section called "DescribePlayerSessionsResult" Required: No
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "GameLiftError" Required: No

DescribePlayerSessionsResult

Properties	Description
NextToken	The token indicating the start of the next page of results. To specify the start of the result set, don't provide a value. If you provide a player session ID, this parameter is ignored. Type: string Required: Yes
PlayerSessions	A collection of objects containing properties for each player session that matches the request. Type: IList<the section called "PlayerSession"> Required:
Success	Whether the action was successful or not.

Properties	Description
	Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "GameLiftError" Required: No

PlayerSession

Properties	Description
CreationTime	Type: long Required: Yes
FleetId	Type: string Required: Yes
GameSessionId	Type: string Required: Yes
IpAddress	Type: string Required: Yes
PlayerData	Type: string Required: Yes
PlayerId	Type: string Required: Yes
PlayerSessionId	Type: string

Properties	Description
	Required: Yes
Port	Type: int Required: Yes
Status	Type: A PlayerSessionStatus enum . Required: Yes
TerminationTime	Type: long Required: Yes
DnsName	Type: string Required: Yes

StartMatchBackfillOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: the section called "StartMatchBackfillResult" Required: No
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "GameLiftError"

Properties	Description
	Required: No

StartMatchBackfillResult

Properties	Description
TicketId	Type: string Required: Yes

GetComputeCertificateOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: the section called "GetComputeCertificateResult" Required: No
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "GameLiftError" Required: No

GetComputeCertificateResult

The path to the TLS certificate on your compute and the compute's host name.

Properties	Description
CertificatePath	Type: string Required: Yes
ComputeName	Type: string Required: Yes

GetFleetRoleCredentialsOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: the section called "GetFleetRoleCredentialsResult" Required: No
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "GameLiftError" Required: No

GetFleetRoleCredentialsResult

Properties	Description
AccessKeyId	The access key ID to authenticate and provide access to your AWS resources. Type: string Required: No
AssumedRoleId	The ID of the user that the service role belongs to. Type: string Required: No
AssumedRoleUserArn	The Amazon Resource Name (ARN) of the user that the service role belongs to. Type: string Required: No
Expiration	The amount of time until your session credentials expire. Type: DateTime Required: No
SecretAccessKey	The secret access key ID for authentication. Type: string Required: No
SessionToken	A token to identify the current active session interacting with your AWS resources. Type: string Required: No

Properties	Description
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "GameLiftError" Required: No

ListContainersNetworkInfoOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: the section called "ListContainersNetworkInfoResult" Required: No
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "GameLiftError" Required: No

ListContainersNetworkInfoResult

Network information for all containers running on the same instance as the calling game server process.

Properties	Description
ContainersNetworkInfo	The list of network information for each container running on the instance. Type: A list of the section called “ContainerNetworkInfo” objects Required: No

ContainerNetworkInfo

Network information for a single container running on the instance.

Properties	Description
ContainerName	The name of the container, as defined in the container group definition. Type: string Required: No
ContainerId	The unique identifier of the container. Type: string Required: No
IpAddress	The container's local IPv4 address on the Docker bridge network. Type: string Required: No

Properties	Description
ContainerGroupType	The type of container group that the container belongs to. Type: A ContainerGroupType enum . Required: No

AwsDateTimeOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: DateTime Required: No
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "GameLiftError" Required: No

GameLiftError

Properties	Description
ErrorType	The type of error. Type: A GameLiftErrorType enum .

Properties	Description
	Required: No
ErrorMessage	The error message. Type: string Required: No
ErrorMessage	The error message. Type: string Required: No

Enums

Enums defined for the server SDK for Amazon GameLift Servers (C#) are defined as follows:

AttrType

- **NONE**
- **STRING**
- **DOUBLE**
- **STRING_LIST**
- **STRING_DOUBLE_MAP**

GameLiftErrorType

String value indicating the error type. Valid values include:

- **SERVICE_CALL_FAILED** – A call to an AWS service has failed.
- **LOCAL_CONNECTION_FAILED** – The local connection to Amazon GameLift Servers failed.
- **NETWORK_NOT_INITIALIZED** – The network has not been initialized.
- **GAMESESSION_ID_NOT_SET** – The game session ID has not been set.
- **BAD_REQUEST_EXCEPTION**
- **INTERNAL_SERVICE_EXCEPTION**
- **UNSUPPORTED_COMPUTE_TYPE_EXCEPTION** – The API that was called was unsupported on the compute type.

- **ALREADY_INITIALIZED** – The Amazon GameLift Servers Server or Client has already been initialized with `Initialize()`.
- **FLEET_MISMATCH** – The target fleet does not match the fleet of a `gameSession` or `playerSession`.
- **GAMELIFT_CLIENT_NOT_INITIALIZED** – The Amazon GameLift Servers client has not been initialized.
- **GAMELIFT_SERVER_NOT_INITIALIZED** – The Amazon GameLift Servers server has not been initialized.
- **GAME_SESSION_ENDED_FAILED** – the server SDK for Amazon GameLift Servers could not contact the service to report the game session ended.
- **GAME_SESSION_NOT_READY** – The Amazon GameLift Servers Server Game Session was not activated.
- **GAME_SESSION_READY_FAILED** – the server SDK for Amazon GameLift Servers could not contact the service to report the game session is ready.
- **INITIALIZATION_MISMATCH** – A client method was called after `Server::Initialize()`, or vice versa.
- **NOT_INITIALIZED** – The Amazon GameLift Servers Server or Client has not been initialized with `Initialize()`.
- **NO_TARGET_ALIASID_SET** – A target `aliasId` has not been set.
- **NO_TARGET_FLEET_SET** – A target fleet has not been set.
- **PROCESS_ENDING_FAILED** – the server SDK for Amazon GameLift Servers could not contact the service to report the process is ending.
- **PROCESS_NOT_ACTIVE** – The server process is not yet active, not bound to a `GameSession`, and cannot accept or process `PlayerSessions`.
- **PROCESS_NOT_READY** – The server process is not yet ready to be activated.
- **PROCESS_READY_FAILED** – the server SDK for Amazon GameLift Servers could not contact the service to report the process is ready.
- **SDK_VERSION_DETECTION_FAILED** – SDK version detection failed.
- **STX_CALL_FAILED** – A call to the XStx server backend component has failed.
- **STX_INITIALIZATION_FAILED** – The XStx server backend component has failed to initialize.
- **UNEXPECTED_PLAYER_SESSION** – An unregistered player session was encountered by the server.

- **WEBSOCKET_CONNECT_FAILURE**
- **WEBSOCKET_CONNECT_FAILURE_FORBIDDEN**
- **WEBSOCKET_CONNECT_FAILURE_INVALID_URL**
- **WEBSOCKET_CONNECT_FAILURE_TIMEOUT**
- **WEBSOCKET_RETRIABLE_SEND_MESSAGE_FAILURE** – Retriable failure to send a message to the GameLift Service WebSocket.
- **WEBSOCKET_SEND_MESSAGE_FAILURE** – Failure to send a message to the GameLift Service WebSocket.
- **MATCH_BACKFILL_REQUEST_VALIDATION** – Validation of the request failed.
- **PLAYER_SESSION_REQUEST_VALIDATION** – Validation of the request failed.

PlayerSessionCreationPolicy

String value indicating whether the game session accepts new players. Valid values include:

- **ACCEPT_ALL** – Accept all new player sessions.
- **DENY_ALL** – Deny all new player sessions.
- **NOT_SET** – The game session is not set to accept or deny new player sessions.

PlayerSessionStatus

- **ACTIVE**
- **COMPLETED**
- **NOT_SET**
- **RESERVED**
- **TIMEDOUT**

ContainerGroupType

The type of container group that a container belongs to. Valid values include:

- **GAME_SERVER** – A game server replica container group. An instance can have multiple game server container groups.
- **PER_INSTANCE** – A per-instance daemon container group. An instance has exactly one per-instance container group.

[C# server SDK 5.x for Amazon GameLift Servers -- Data types](#)

Topics

- [GetSdkVersion\(\)](#)
- [InitMetrics\(\)](#)
- [InitSDK\(\)](#)
- [InitSDK\(\)](#)
- [ProcessReady\(\)](#)
- [ProcessEnding\(\)](#)
- [ActivateGameSession\(\)](#)
- [UpdatePlayerSessionCreationPolicy\(\)](#)
- [GetGameSessionId\(\)](#)
- [GetTerminationTime\(\)](#)
- [AcceptPlayerSession\(\)](#)
- [RemovePlayerSession\(\)](#)
- [DescribePlayerSessions\(\)](#)
- [StartMatchBackfill\(\)](#)
- [StopMatchBackfill\(\)](#)
- [GetComputeCertificate\(\)](#)
- [GetFleetRoleCredentials\(\)](#)
- [ListContainersNetworkInfo\(\)](#)
- [Destroy\(\)](#)

GetSdkVersion()

Returns the current version number of the SDK built into the server process.

Syntax

```
AwsStringOutcome GetSdkVersion();
```

Return value

If successful, returns the current SDK version as an [the section called “AwsStringOutcome”](#) object. The returned string includes the version number (example 5.0.0). If not successful, returns an error message.

Example

```
var getSdkVersionOutcome = GameLiftServerAPI.GetSdkVersion();
```

InitMetrics()

Initializes the metrics system for collecting and reporting server performance data. For best results, call this method before [InitSDK\(\)](#) to enable metrics collection during SDK initialization.

Syntax

```
MetricsManagerOutcome InitMetrics();  
MetricsManagerOutcome InitMetrics(MetricsParameters metricsParameters);
```

Parameters

[MetricsParameters](#) (optional)

Configuration parameters for metrics collection. If not provided, uses default values that can be overridden by environment variables.

Return value

If successful, returns a [the section called “MetricsManagerOutcome”](#) object containing the MetricsManager instance. If not successful, returns an error message.

Example

```
// Initialize with default parameters (uses environment variables if available)  
var outcome = GameLiftServerAPI.InitMetrics();  
if (outcome.Success) {  
    var metricsManager = outcome.Result;  
} else {  
    Console.WriteLine("Failed to initialize metrics: " + outcome.Error.ErrorMessage);  
}  
  
// Initialize with custom parameters  
var metricsParams = new MetricsParameters("localhost", 8125, "crash-host", 9999, 1000,  
    1024);  
var customOutcome = GameLiftServerAPI.InitMetrics(metricsParams);
```

```
if (customOutcome.Success) {  
    var metricsManager = customOutcome.Result;  
}
```

InitSDK()

Initializes the Amazon GameLift Servers SDK for a managed EC2 fleet. Call this method on launch, before any other initialization related to Amazon GameLift Servers occurs. This method reads server parameters from the host environment to set up communication between the server and the Amazon GameLift Servers service. It uses an idempotency token, so you safely retry this call when it fails.

Syntax

```
GenericOutcome InitSDK();
```

Return value

If successful, returns an `InitSdkOutcome` object to indicate that the server process is ready to call [ProcessReady\(\)](#).

Example

```
//Call InitSDK to establish a local connection with the GameLift agent to enable  
further communication.  
GenericOutcome initSDKOutcome = GameLiftServerAPI.InitSDK();
```

InitSDK()

Initializes the Amazon GameLift Servers SDK for an Anywhere fleet. Call this method on launch, before any other initialization related to Amazon GameLift Servers occurs. This method requires explicit server parameters to set up communication between the server and the Amazon GameLift Servers service. It uses an idempotency token, so you safely retry this call when it fails.

Syntax

```
GenericOutcome InitSDK(ServerParameters serverParameters);
```

Parameters

[ServerParameters](#)

To initialize a game server on an Amazon GameLift Servers Anywhere fleet, construct a `ServerParameters` object with the following information:

- The URL of the WebSocket used to connect to your game server.
- The ID of the process used to host your game server.
- The ID of the compute hosting your game server processes.
- The ID of the Amazon GameLift Servers fleet containing your Amazon GameLift Servers Anywhere compute.
- The authorization token generated by the Amazon GameLift Servers operation.

Return value

If successful, returns an `InitSdkOutcome` object to indicate that the server process is ready to call [ProcessReady\(\)](#).

Note

If calls to `InitSDK()` are failing for game builds deployed to Anywhere fleets, check the `ServerSdkVersion` parameter used when creating the build resource. You must explicitly set this value to the server SDK version in use. The default value for this parameter is 4.x, which is not compatible. To resolve this issue, create a new build and deploy it to a new fleet.

Example

```
//Define the server parameters
string websocketUrl = "wss://us-west-1.api.amazongamelift.com";
string processId = "PID1234";
string fleetId = "aarn:aws:gamelift:us-west-1:111122223333:fleet/
fleet-9999ffff-88ee-77dd-66cc-5555bbbb44aa";
string hostId = "HardwareAnywhere";
string authToken = "1111aaaa-22bb-33cc-44dd-5555eeee66ff";
ServerParameters serverParameters =
    new ServerParameters(websocketUrl, processId, hostId, fleetId, authToken);
```



```
//Call InitSDK to establish a local connection with the GameLift agent to enable
further communication.
GenericOutcome initSDKOutcome = GameLiftServerAPI.InitSDK(serverParameters);
```

ProcessReady()

Notifies Amazon GameLift Servers that the server process is ready to host game sessions. Call this method after invoking [InitSDK\(\)](#). This method should be called only once per process.

Syntax

```
GenericOutcome ProcessReady(ProcessParameters processParameters)
```

Parameters

[ProcessParameters](#)

A ProcessParameters object holds information about the server process.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example illustrates both the method and delegate function implementations.

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

ProcessEnding()

Notifies Amazon GameLift Servers that the server process is terminating. Call this method after all other cleanup tasks (including shutting down the active game session) and before terminating the process. Depending on the result of `ProcessEnding()`, the process exits with success (0) or error (-1) and generates a fleet event. If the process terminates with an error, the fleet event generated is `SERVER_PROCESS_TERMINATED_UNHEALTHY`.

Syntax

```
GenericOutcome ProcessEnding()
```

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example calls `ProcessEnding()` and `Destroy()` before terminating the server process with a success or error exit code.

```
GenericOutcome processEndingOutcome = GameLiftServerAPI.ProcessEnding();
GameLiftServerAPI.Destroy();

if (processEndingOutcome.Success)
{
    Environment.Exit(0);
}
else
{
    Console.WriteLine("ProcessEnding() failed. Error: " +
        processEndingOutcome.Error.ToString());
    Environment.Exit(-1);
}
```

ActivateGameSession()

Notifies Amazon GameLift Servers that the server process has activated a game session and is now ready to receive player connections. This action should be called as part of the `onStartGameSession()` callback function, after all game session initialization.

Syntax

```
GenericOutcome ActivateGameSession()
```

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example shows `ActivateGameSession()` being called as part of the `onStartGameSession()` delegate function.

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

UpdatePlayerSessionCreationPolicy()

Updates the current game session's ability to accept new player sessions. A game session can be set to either accept or deny all new player sessions.

Syntax

```
GenericOutcome UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy
playerSessionPolicy)
```

Parameters

playerSessionPolicy

String value that indicates whether the game session accepts new players.

Valid values include:

- **ACCEPT_ALL** – Accept all new player sessions.
- **DENY_ALL** – Deny all new player sessions.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example sets the current game session's join policy to accept all players.

```
GenericOutcome updatePlayerSessionPolicyOutcome =  
    GameLiftServerAPI.UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy.ACCEPT_ALL);
```

GetGameSessionId()

Retrieves the ID of the game session hosted by the active server process.

For idle processes that aren't activated with a game session, the call returns a [the section called "GameLiftError"](#).

Syntax

```
AwsStringOutcome GetGameSessionId()
```

Return value

If successful, returns the game session ID as an [the section called "AwsStringOutcome"](#) object. If not successful, returns an error message."

Example

```
AwsStringOutcome getGameSessionIdOutcome = GameLiftServerAPI.GetGameSessionId();
```

GetTerminationTime()

Returns the time that a server process is scheduled to be shut down, if a termination time is available. A server process takes this action after receiving an `onProcessTerminate()` callback from Amazon GameLift Servers. Amazon GameLift Servers calls `onProcessTerminate()` for the following reasons:

- When the server process has reported poor health or has not responded to Amazon GameLift Servers.

- When terminating the instance during a scale-down event.
- When an instance is terminated due to a [spot-instance interruption](#).

Syntax

```
AwsDateTimeOutcome GetTerminationTime()
```

Return value

If successful, returns the termination time as an [the section called "AwsDateTimeOutcome"](#) object. The value is the termination time, expressed in elapsed ticks since 0001 00:00:00. For example, the date time value 2020-09-13 12:26:40 -0000Z is equal to 637355968000000000 ticks. If no termination time is available, returns an error message.

Example

```
AwsDateTimeOutcome getTerminationTimeOutcome = GameLiftServerAPI.GetTerminationTime();
```

AcceptPlayerSession()

Notifies Amazon GameLift Servers that a player with the specified player session ID has connected to the server process and needs validation. Amazon GameLift Servers verifies that the player session ID is valid. After the player session is validated, Amazon GameLift Servers changes the status of the player slot from RESERVED to ACTIVE.

Syntax

```
GenericOutcome AcceptPlayerSession(String playerSessionId)
```

Parameters

playerSessionId

Unique ID issued by GameLift when a new player session is created.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example illustrates a function for handling a connection request, including validating and rejecting invalid player session IDs.

```
void ReceiveConnectingPlayerSessionID (Connection connection, String playerId)
{
    GenericOutcome acceptPlayerSessionOutcome =
    GameLiftServerAPI.AcceptPlayerSession(playerSessionId);
    if(acceptPlayerSessionOutcome.Success)
    {
        connectionToSessionMap.emplace(connection, playerId);
        connection.Accept();
    }
    else
    {
        connection.Reject(acceptPlayerSessionOutcome.Error.ErrorMessage);
    }
}
```

RemovePlayerSession()

Notifies Amazon GameLift Servers that a player has disconnected from the server process. In response, Amazon GameLift Servers changes the player slot to available.

Syntax

```
GenericOutcome RemovePlayerSession(String playerId)
```

Parameters

playerSessionId

Unique ID issued by Amazon GameLift Servers when a new player session is created.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
GenericOutcome removePlayerSessionOutcome =  
    GameLiftServerAPI.RemovePlayerSession(playerSessionId);
```

DescribePlayerSessions()

Retrieves player session data which includes settings, session metadata, and player data. Use this action to get information for a single player session, for all player sessions in a game session, or for all player sessions associated with a single player ID.

Syntax

```
DescribePlayerSessionsOutcome DescribePlayerSessions(DescribePlayerSessionsRequest  
    describePlayerSessionsRequest)
```

Parameters

DescribePlayerSessionsRequest

A [the section called “DescribePlayerSessionsRequest”](#) object that describes which player sessions to retrieve.

Return value

If successful, returns a [the section called “DescribePlayerSessionsOutcome”](#) object that contains a set of player session objects that fit the request parameters.

Example

This example illustrates a request for all player sessions actively connected to a specified game session. By omitting *NextToken* and setting the *Limit* value to 10, Amazon GameLift Servers will return the first 10 player session records matching the request.

```
// Set request parameters  
DescribePlayerSessionsRequest describePlayerSessionsRequest = new  
    DescribePlayerSessionsRequest()  
{  
    GameSessionId = GameLiftServerAPI.GetGameSessionId().Result,    //gets the ID for the  
    current game session
```

```
Limit = 10,  
PlayerSessionStatusFilter =  
    PlayerSessionStatusMapper.GetNameForPlayerSessionStatus(PlayerSessionStatus.ACTIVE)  
};  
// Call DescribePlayerSessions  
DescribePlayerSessionsOutcome describePlayerSessionsOutcome =  
    GameLiftServerAPI.DescribePlayerSessions(describePlayerSessionsRequest);
```

StartMatchBackfill()

Sends a request to find new players for open slots in a game session created with FlexMatch. For more information, see [FlexMatch backfill feature](#).

This action is asynchronous. If new players are matched, Amazon GameLift Servers delivers updated matchmaker data using the callback function `OnUpdateGameSession()`.

A server process can have only one active match backfill request at a time. To send a new request, first call [StopMatchBackfill\(\)](#) to cancel the original request.

Syntax

```
StartMatchBackfillOutcome StartMatchBackfill (StartMatchBackfillRequest  
startBackfillRequest);
```

Parameters

[StartMatchBackfillRequest](#)

A `StartMatchBackfillRequest` object holds information about the backfill request.

Return value

Returns a [the section called "StartMatchBackfillOutcome"](#) object with the match backfill ticket ID, or failure with an error message.

Example

```
// Build a backfill request  
StartMatchBackfillRequest startBackfillRequest = new StartMatchBackfillRequest()  
{
```



```
TicketId = "1111aaaa-22bb-33cc-44dd-5555eeee66ff", //optional
MatchmakingConfigurationArn = "arn:aws:gamelift:us-
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig",
GameSessionId = GameLiftServerAPI.GetGameSessionId().Result,    // gets ID for
current game session
MatchmakerData matchmakerData =
    MatchmakerData.FromJson(gameSession.MatchmakerData), // gets matchmaker data for
current players
// get matchmakerData.Players
// remove data for players who are no longer connected
Players = ListOfPlayersRemainingInTheGame
};

// Send backfill request
StartMatchBackfillOutcome startBackfillOutcome =
    GameLiftServerAPI.StartMatchBackfill(startBackfillRequest);

// Implement callback function for backfill
void OnUpdateGameSession(GameSession myGameSession)
{
    // game-specific tasks to prepare for the newly matched players and update matchmaker
    data as needed
}
```

StopMatchBackfill()

Cancels an active match backfill request. For more information, see [FlexMatch backfill feature](#).

Syntax

```
GenericOutcome StopMatchBackfill (StopMatchBackfillRequest stopBackfillRequest);
```

Parameters

StopMatchBackfillRequest

A `StopMatchBackfillRequest` object that provides details about the matchmaking ticket you are stopping.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
// Set backfill stop request parameters
StopMatchBackfillRequest stopBackfillRequest = new StopMatchBackfillRequest(){
    TicketId = "1111aaaa-22bb-33cc-44dd-5555eeee66ff", //optional, if not provided one is
    autogenerated
    MatchmakingConfigurationArn = "arn:aws:gamelift:us-
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig",
    GameSessionId = GameLiftServerAPI.GetGameSessionId().Result //gets the ID for the
    current game session
};
GenericOutcome stopBackfillOutcome =
    GameLiftServerAPI.StopMatchBackfillRequest(stopBackfillRequest);
```

GetComputeCertificate()

Retrieves the path to the TLS certificate used to encrypt the network connection between the game server and your game client. You can use the certificate path when you register your compute device to a Amazon GameLift Servers Anywhere fleet. For more information see, [RegisterCompute](#).

Syntax

```
GetComputeCertificateOutcome GetComputeCertificate();
```

Return value

Returns a `GetComputeCertificateResponse` object that contains the following:

- **CertificatePath:** The path to the TLS certificate on your compute resource. When using an Amazon GameLift Servers managed fleet, this path contains:
 - `certificate.pem`: The end-user certificate. The full certificate chain is the combination of `certificateChain.pem` appended to this certificate.
 - `certificateChain.pem`: The certificate chain that contains the root certificate and intermediate certificates.
 - `rootCertificate.pem`: The root certificate.
 - `privateKey.pem`: The private key for the end-user certificate.
- **ComputeName:** The name of your compute resource.

Example

```
GetComputeCertificateOutcome getComputeCertificateOutcome =  
    GameLiftServerAPI.GetComputeCertificate();
```

GetFleetRoleCredentials()

Retrieves IAM role credentials that authorize Amazon GameLift Servers to interact with other AWS services. For more information, see [Connect your Amazon GameLift Servers hosted game server to other AWS resources](#).

Syntax

```
GetFleetRoleCredentialsOutcome GetFleetRoleCredentials(GetFleetRoleCredentialsRequest  
    request);
```

Parameters

[GetFleetRoleCredentialsRequest](#)

Role credentials that extend limited access to your AWS resources to the game server.

Return value

Returns a [the section called "GetFleetRoleCredentialsOutcome"](#) object.

Example

```
// form the fleet credentials request  
GetFleetRoleCredentialsRequest getFleetRoleCredentialsRequest = new  
    GetFleetRoleCredentialsRequest(){  
    RoleArn = "arn:aws:iam::123456789012:role/service-role/exampleGameLiftAction"  
};  
GetFleetRoleCredentialsOutcome GetFleetRoleCredentialsOutcome credentials =  
    GetFleetRoleCredentials(getFleetRoleCredentialsRequest);
```

ListContainersNetworkInfo()

Retrieves network information for all containers running on the same instance, including each container's name, ID, local IP address, and container group type. Use this information to enable

game server processes to discover and communicate with other containers that are running on the same instance.

This action is supported only on container fleets. When called from any other compute type, it returns an `UNSUPPORTED_COMPUTE_TYPE_EXCEPTION` error. Amazon GameLift Servers obtains the network information from a discovery server that runs locally on the instance.

Syntax

```
ListContainersNetworkInfoOutcome ListContainersNetworkInfo();
```

Return value

Returns a [the section called “ListContainersNetworkInfoOutcome”](#) object.

Example

```
ListContainersNetworkInfoOutcome listContainersNetworkInfoOutcome =  
    GameLiftServerAPI.ListContainersNetworkInfo();
```

Destroy()

Frees the Amazon GameLift Servers game server SDK from memory. As a best practice, call this method after `ProcessEnding()` and before terminating the process. If you're using an Anywhere fleet and you're not terminating server processes after every game session, call `Destroy()` and then `InitSDK()` to reinitialize before notifying Amazon GameLift Servers that the process is ready to host a game session with `ProcessReady()`.

Syntax

```
GenericOutcome Destroy()
```

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
// Operations to end game sessions and the server process  
GenericOutcome processEndingOutcome = GameLiftServerAPI.ProcessEnding();  
  
// Shut down and destroy the instance of the GameLift Game Server SDK
```

```
GenericOutcome destroyOutcome = GameLiftServerAPI.Destroy();

// Exit the process with success or failure
if (processEndingOutcome.Success)
{
    Environment.Exit(0);
}
else
{
    Console.WriteLine("ProcessEnding() failed. Error: " +
        processEndingOutcome.Error.ToString());
    Environment.Exit(-1);
}
```

Go server SDK for Amazon GameLift Servers -- Actions

Use the server SDK 5.x reference to integrate your multiplayer game for hosting with Amazon GameLift Servers. For guidance about the integration process, see [Add Amazon GameLift Servers to your game server with the server SDK](#).

`GameLiftServerAPI.go` defines the Go server SDK actions.

Go server SDK for Amazon GameLift Servers -- Data types

Use the server SDK reference to integrate your multiplayer game for hosting with Amazon GameLift Servers. For guidance about the integration process, see [Add Amazon GameLift Servers to your game server with the server SDK](#).

`GameLiftServerAPI.go` defines the Go server SDK actions.

[Go server SDK for Amazon GameLift Servers -- Actions](#)

Data types

- [LogParameters](#)
- [MetricsParameters](#)
- [ProcessParameters](#)
- [UpdateGameSession](#)
- [GameSession](#)
- [ServerParameters](#)
- [StartMatchBackfillRequest](#)

- [Player](#)
- [DescribePlayerSessionsRequest](#)
- [StopMatchBackfillRequest](#)
- [GetFleetRoleCredentialsRequest](#)

LogParameters

An object identifying files generated during a game session that you want Amazon GameLift Servers to upload and store after the game session ends. The game server provides `LogParameters` to Amazon GameLift Servers as part of a `ProcessParameters` object in a [ProcessReady\(\)](#) call.

Properties	Description
LogPaths	The list of directory paths to game server log files that you want Amazon GameLift Servers to store for future access. The server process generates these files during each game session. You define file paths and names in your game server and store them in the root game build directory. The log paths must be absolute. For example, if your game build stores game session logs in a path such as <code>MyGame\sessionLogs\</code>, then the path would be <code>c:\game\MyGame\sessionLogs</code> on a Windows instance. Type: <code>[]string</code> Required: No

MetricsParameters

An object communicating configuration parameters for initializing the metrics system. This configuration is used to set up StatsD reporting, crash reporting, and metrics processing behavior. The game server provides `MetricsParameters` to Amazon GameLift Servers as part of a [InitMetrics\(\)](#) call.

Properties	Description
StatsdHost	The StatsD server host for metrics reporting (e.g., "localhost").

	Type: string Required: Yes
StatsdPort	The StatsD server port for metrics reporting (e.g., 8125). Type: int Required: Yes
CrashReporterHost	The crash reporter host for crash tracking and process monitoring. Type: string Required: Yes
CrashReporterPort	The crash reporter port for crash tracking and process monitoring. Type: int Required: Yes
FlushIntervals	The metrics flush interval in milliseconds. Controls how frequently metrics are sent to StatsD. Type: int Required: Yes
MaxPacketSize	The maximum packet size for metrics in bytes. Limits the size of UDP packets sent to StatsD. Type: int Required: Yes

ProcessParameters

An object describing the communication between a server process and Amazon GameLift Servers. The server process provides this information to Amazon GameLift Servers with a call to [ProcessReady\(\)](#).

Properties	Description
LogParameters	An object with directory paths to files that are generated during a game session. Amazon GameLift Servers copies and stores the files for future access. Type: LogParameters Required: No
OnHealthCheck	The callback function that Amazon GameLift Servers invokes to request a health status report from the server process. Amazon GameLift Servers calls this function every 60 seconds and waits 60 seconds for a response. The server process returns TRUE if healthy, FALSE if not healthy. If no response is returned, Amazon GameLift Servers records the server process as not healthy. Type: OnHealthCheck func() bool Required: No
OnProcessTerminate	The callback function that Amazon GameLift Servers invokes to force the server process to shut down. After calling this function, Amazon GameLift Servers waits 5 minutes for the server process to shut down and respond with a ProcessEnding() call before it shuts down the server process. Type: OnProcessTerminate func() Required: Yes
OnStartGameSession	The callback function that Amazon GameLift Servers invokes to pass an updated game session object to the server process. Amazon GameLift Servers calls this function when a match backfill request has been processed to provide updated matchmaker data. It passes a GameSession object, a status update (updateReason), and the match backfill ticket ID. Type: OnStartGameSession func (model.GameSession) Required: Yes

OnUpdateGameSession	The callback function that Amazon GameLift Servers invokes to pass updated game session information to the server process. Amazon GameLift Servers calls this function after processing a match backfill request to provide updated matchmaker data. Type: OnUpdateGameSession func (model.UpdateGameSession) Required: No
Port	The port number that the server process listens on for new player connections. The value must fall into the port range configured for any fleet deploying this game server build. This port number is included in game session and player session objects, which game sessions use when connecting to a server process. Type: int Required: Yes

UpdateGameSession

The updates to a game session object, which includes the reason that the game session was updated, and the related backfill ticket ID if backfill is being used to fill player sessions in the game session.

Properties	Description
GameSession	A GameSession object. The GameSession object contains properties describing a game session. Type: GameSession GameSession() Required: Yes
UpdateReason	The reason that the game session is being updated. Type: UpdateReason UpdateReason()

Properties	Description
	Required: Yes
BackfillTicketId	The ID of the backfill ticket attempting to update the game session. Type: String Required: No

GameSession

The details of a game session.

Properties	Description
GameSessionId	A unique identifier for the game session. A game session Amazon Resource Name (ARN) has the following format: <code>arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token></code> . Type: String Required: No
Name	A descriptive label of the game session. Type: String Required: No
FleetId	A unique identifier for the fleet that the game session is running on. Type: String Required: No
MaximumPlayerSessionCount	The maximum number of player connections to the game session. Type: Integer

Properties	Description
	Required: No
Port	The port number for the game session. To connect to an Amazon GameLift Servers game server, an app needs both the IP address and port number. Type: Integer Required: No
IpAddress	The IP address of the game session. To connect to an Amazon GameLift Servers game server, an app needs both the IP address and port number. Type: String Required: No
GameSessionData	A set of custom game session properties, formatted as a single string value. Type: String Required: No
MatchmakerData	The information about the matchmaking process that was used to create the game session, in JSON syntax, formatted as a string. In addition to the matchmaking configuration used, it contains data on all players assigned to the match, including player attributes and team assignments. Type: String Required: No
GameProperties	A set of custom properties for a game session, formatted as key:value pairs. These properties are passed with a request to start a new game session. Type: map[string] string Required: No

Properties	Description
DnsName	The DNS identifier assigned to the instance that's running the game session. Values have the following format: • TLS-enabled fleets: <unique identifier>.<region identifier>.amazongamelift.com . • Non-TLS-enabled fleets: ec2-<unique identifier>.compute.amazonaws.com . When connecting to a game session that's running on a TLS-enabled fleet, you must use the DNS name, not the IP address. Type: String Required: No

ServerParameters

Information used to maintain the connection between an Amazon GameLift Servers Anywhere server and the Amazon GameLift Servers service. This information is used when launching new server processes with [InitSDK\(\)](#). For servers hosted on Amazon GameLift Servers managed EC2 instances, use an empty object.

Properties	Description
WebSocket URL	The <code>GameLiftServerSdkEndpoint</code> Amazon GameLift Servers returns when you RegisterCompute for an Amazon GameLift Servers Anywhere compute resource. Type: string Required: Yes
ProcessID	A unique identifier registered to the server process hosting your game. Type: string

Properties	Description
	Required: Yes
HostID	The unique identifier of the compute resource that's hosting the new server process. The HostID is the ComputeName used when you registered your compute. For more information, see RegisterCompute. Type: string Required: Yes
FleetID	The unique identifier of the fleet that the compute is registered to. For more information, see RegisterCompute. Type: string Required: Yes
AuthToken	The authentication token generated by Amazon GameLift Servers that authenticates your server to Amazon GameLift Servers. For more information, see GetComputeAuthToken. Type: string Required: Yes

StartMatchBackfillRequest

Information used to create a matchmaking backfill request. The game server communicates this information to Amazon GameLift Servers in a [StartMatchBackfill\(\)](#) call.

Properties	Description
GameSessionArn	The unique game session identifier. The API operation GetGameSessionId returns the identifier in ARN format. Type: String

Properties	Description
	Required: Yes
MatchmakingConfigurationArn	The unique identifier (in the form of an ARN) for the matchmaker to use for this request. The matchmaker ARN for the original game session is in the game session object in the matchmaker data property. For more information about matchmaker data, see Work with matchmaker data. Type: String Required: Yes
Players	A set of data that represents all players who are currently in the game session. The matchmaker uses this information to search for new players who are good matches for the current players. Type: []model.Player Required: Yes
TicketId	The unique identifier for a matchmaking or match backfill request ticket. If you don't provide a value, Amazon GameLift Servers generates one. Use this identifier to track the match backfill ticket status or cancel the request if needed. Type: String Required: No

Player

The object that represents a player in matchmaking. When a matchmaking request starts, a player has a player ID, attributes, and possibly latency data. Amazon GameLift Servers adds team information after a match is made.

Properties	Description
LatencyInMS	A set of values expressed in milliseconds that indicate the amount of latency that a player experiences when connected to a location.

Properties	Description
	If this property is used, the player is only matched for locations listed. If a matchmaker has a rule that evaluates player latency, players must report latency to be matched. Type: <code>map[string] int</code> Required: No
PlayerAttributes	A collection of key:value pairs that contain player information for use in matchmaking. Player attribute keys must match the PlayerAttributes used in a matchmaking rule set. For more information about player attributes, see AttributeValue. Type: <code>map[string] AttributeValue</code> Required: No
PlayerId	A unique identifier for a player. Type: <code>String</code> Required: No
Team	The name of the team that the player is assigned to in a match. You define the team name in the matchmaking rule set. Type: <code>String</code> Required: No

DescribePlayerSessionsRequest

An object that specifies which player sessions to retrieve. The server process provides this information with a [DescribePlayerSessions\(\)](#) call to Amazon GameLift Servers.

Properties	Description
GameSessionID	A unique game session identifier. Use this parameter to request all player sessions for the specified game session. Game session ID format is <code>arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string></code>. The <code>GameSessionID</code> is a custom ID string or a generated string. Type: String Required: No
PlayerSessionID	The unique identifier for a player session. Use this parameter to request a single specific player session. Type: String Required: No
PlayerID	The unique identifier for a player. Use this parameter to request all player sessions for a specific player. See Generate player IDs. Type: String Required: No
PlayerSessionStatusFilter	The player session status to filter results on. Possible player session statuses include: • RESERVED – The player session request was received, but the player hasn't connected to the server process or been validated. • ACTIVE – The player was validated by the server process and is connected. • COMPLETED – The player connection dropped. • TIMEDOUT – A player session request was received, but the player didn't connect or wasn't validated within the time-out limit (60 seconds). Type: String

Properties	Description
	Required: No
NextToken	The token indicating the start of the next page of results. To specify the start of the result set, don't provide a value. If you provide a player session ID, this parameter is ignored. Type: String Required: No
Limit	The maximum number of results to return. If you provide a player session ID, this parameter is ignored. Type: int Required: No

StopMatchBackfillRequest

Information used to cancel a matchmaking backfill request. The game server communicates this information to Amazon GameLift Servers service in a [StopMatchBackfill\(\)](#) call.

Properties	Description
GameSessionArn	The unique game session identifier of the request being canceled. Type: string Required: No
MatchmakingConfigurationArn	The unique identifier of the matchmaker this request was sent to. Type: string Required: No
TicketId	The unique identifier of the backfill request ticket to be canceled. Type: string

Properties	Description
	Required: No

GetFleetRoleCredentialsRequest

The role credentials that extend limited access to your AWS resources to the game server. For more information see, [Set up an IAM service role for Amazon GameLift Servers](#).

Properties	Description
RoleArn	The ARN of the service role that extends limited access to your AWS resources. Type: string Required: Yes
RoleSessionName	The name of the session that describes the use of the role credentials. Type: string Required: Yes

[Go server SDK for Amazon GameLift Servers -- Data types](#)

Actions

- [GetSdkVersion\(\)](#)
- [InitMetrics\(\)](#)
- [InitMetricsFromEnvironment\(\)](#)
- [InitSDK\(\)](#)
- [ProcessReady\(\)](#)
- [ProcessEnding\(\)](#)
- [ActivateGameSession\(\)](#)
- [UpdatePlayerSessionCreationPolicy\(\)](#)
- [GetGameSessionId\(\)](#)

- [GetTerminationTime\(\)](#)
- [AcceptPlayerSession\(\)](#)
- [RemovePlayerSession\(\)](#)
- [DescribePlayerSessions\(\)](#)
- [StartMatchBackfill\(\)](#)
- [StopMatchBackfill\(\)](#)
- [GetComputeCertificate\(\)](#)
- [GetFleetRoleCredentials\(\)](#)
- [ListContainersNetworkInfo\(\)](#)
- [Destroy\(\)](#)

GetSdkVersion()

Returns the current version number of the SDK built into the server process.

Syntax

```
func GetSdkVersion() (string, error)
```

Return value

If successful, returns the current SDK version as a string. The returned string includes the version number (example 5.0.0). If not successful, returns an error message such as `common.SdkVersionDetectionFailed`.

Example

```
version, err := server.GetSdkVersion()
```

InitMetrics()

Initializes metrics collection for the Amazon GameLift Servers SDK. This method sets up metrics reporting to help monitor server performance and health. Call this method after [InitSDK\(\)](#) but before [ProcessReady\(\)](#).

Syntax

```
func InitMetrics() error
```

```
func InitMetrics(metricsParameters MetricsParameters) error
```

Parameters

MetricsParameters (optional)

A `MetricsParameters` object that configures metrics collection. If not provided, default metrics configuration is used. The `MetricsParameters` structure contains the following fields:

- `StatsdHost` - The hostname or IP address of the StatsD server.
- `StatsdPort` - The port number for the StatsD server.
- `CrashReporterHost` - The hostname or IP address of the crash reporter service.
- `CrashReporterPort` - The port number for the crash reporter service.
- `FlushIntervalMs` - The interval in milliseconds for flushing metrics data.
- `MaxPacketSize` - The maximum size of metrics packets in bytes.

For more information about the `MetricsParameters` structure, see [Server SDK 5.x for C# data types](#).

Return value

If successful, returns `nil` error to indicate that metrics collection has been initialized successfully.

Example

Initialize metrics with default configuration:

```
err := server.InitMetrics()
```

Initialize metrics with custom configuration:

```
metricsParams := MetricsParameters{
    StatsdHost:      "localhost",
    StatsdPort:      8125,
    CrashReporterHost: "localhost",
    CrashReporterPort: 9125,
    FlushIntervalMs:  5000,
    MaxPacketSize:    1024,
}
```

```
err := server.InitMetrics(metricsParams)
```

InitMetricsFromEnvironment()

Initializes metrics collection for the Amazon GameLift Servers SDK using configuration from environment variables. This method sets up metrics reporting using default settings derived from the runtime environment.

Call this method after [InitSDK\(\)](#) but before [ProcessReady\(\)](#).

Syntax

```
func InitMetricsFromEnvironment() error
```

Return value

If successful, returns `nil` error to indicate that metrics collection has been initialized successfully using environment configuration.

Example

```
err := server.InitMetricsFromEnvironment()
```

InitSDK()

Initializes the Amazon GameLift Servers SDK. Call this method on launch before any other initialization related to Amazon GameLift Servers occurs. This method sets up communication between the server and the Amazon GameLift Servers service.

Syntax

```
func InitSDK(params ServerParameters) error
```

Parameters

[ServerParameters](#)

To initialize a game server on an Amazon GameLift Servers Anywhere fleet, construct a `ServerParameters` object with the following information:

- The URL of the WebSocket used to connect to your game server.
- The ID of the process used to host your game server.

- The ID of the compute hosting your game server processes.
- The ID of the Amazon GameLift Servers fleet containing your Amazon GameLift Servers Anywhere compute.
- The authorization token generated by the Amazon GameLift Servers operation.

To initialize a game server on an Amazon GameLift Servers managed EC2 fleet, construct a `ServerParameters` object with no parameters. With this call, the Amazon GameLift Servers agent sets up the compute environment and automatically connects to the Amazon GameLift Servers service for you.

Return value

If successful, returns `nil` error to indicate that the server process is ready to call [ProcessReady\(\)](#).

Note

If calls to `InitSDK()` are failing for game builds deployed to Anywhere fleets, check the `ServerSdkVersion` parameter used when creating the build resource. You must explicitly set this value to the server SDK version in use. The default value for this parameter is 4.x, which is not compatible. To resolve this issue, create a new build and deploy it to a new fleet.

Example

Amazon GameLift Servers Anywhere example

```
//Define the server parameters
serverParameters := ServerParameters {
    WebSocketURL: "wss://us-west-1.api.amazongamelift.com",
    ProcessID: "PID1234",
    HostID: "HardwareAnywhere",
    FleetID: "aarn:aws:gamelift:us-west-1:111122223333:fleet/
fleet-9999ffff-88ee-77dd-66cc-5555bbbb44aa",
    AuthToken: "1111aaaa-22bb-33cc-44dd-5555eeee66ff"
}

//Call InitSDK to establish a local connection with the Amazon GameLift Servers Agent
to enable further communication.
```

```
err := server.InitSDK(serverParameters)
```

Amazon GameLift Servers managed EC2 example

```
//Define the server parameters
serverParameters := ServerParameters {}

//Call InitSDK to establish a local connection with the Amazon GameLift Servers Agent
to enable further communication.
err := server.InitSDK(serverParameters)
```

ProcessReady()

Notifies Amazon GameLift Servers that the server process is ready to host game sessions. Call this method after invoking [InitSDK\(\)](#). This method should be called only one time per process.

Syntax

```
func ProcessReady(param ProcessParameters) error
```

Parameters

ProcessParameters

A [ProcessParameters](#) object communicates the following information about the server process:

- The names of the callback methods implemented in the game server code that the Amazon GameLift Servers service invokes to communicate with the server process.
- The port number that the server process is listening on.
- The [LogParameters](#) data type containing the path to any game session-specific files that you want Amazon GameLift Servers to capture and store.

Return value

Returns an error with an error message if the method fails. Returns `nil` if the method is successful.

Example

This example illustrates both the [ProcessReady\(\)](#) call and delegate function implementations.

```
// Define the process parameters
```

```
processParams := ProcessParameters {
    OnStartGameSession: gameProcess.OnStartGameSession,
    OnUpdateGameSession: gameProcess.OnGameSessionUpdate,
    OnProcessTerminate: gameProcess.OnProcessTerminate,
    OnHealthCheck: gameProcess.OnHealthCheck,
    Port: port,
    LogParameters: LogParameters {    // logging and error example
        []string {"C:\\game\\logs", "C:\\game\\error"}
    }
}

err := server.ProcessReady(processParams)
```

ProcessEnding()

Notifies Amazon GameLift Servers that the server process is terminating. Call this method after all other cleanup tasks (including shutting down the active game session) and before terminating the process. Depending on the result of `ProcessEnding()`, the process exits with success (0) or error (-1) and generates a fleet event. If the process terminates with an error, the fleet event generated is `SERVER_PROCESS_TERMINATED_UNHEALTHY`.

Syntax

```
func ProcessEnding() error
```

Return value

Returns a 0 error code or a defined error code.

Example

```
// operations to end game sessions and the server process
defer func() {
    err := server.ProcessEnding()
    server.Destroy()
    if err != nil {
        fmt.Println("ProcessEnding() failed. Error: ", err)
        os.Exit(-1)
    } else {
        os.Exit(0)
    }
}
```


ActivateGameSession()

Notifies Amazon GameLift Servers that the server process has activated a game session and is now ready to receive player connections. This action is called as part of the `onStartGameSession()` callback function, after all game session initialization.

Syntax

```
func ActivateGameSession() error
```

Return value

Returns an error with an error message if the method fails.

Example

This example shows `ActivateGameSession()` called as part of the `onStartGameSession()` delegate function.

```
func OnStartGameSession(GameSession gameSession) {  
    // game-specific tasks when starting a new game session, such as loading map  
    // Activate when ready to receive players  
    err := server.ActivateGameSession();  
}
```

UpdatePlayerSessionCreationPolicy()

Updates the current game session's ability to accept new player sessions. A game session can be set to either accept or deny all new player sessions.

Syntax

```
func UpdatePlayerSessionCreationPolicy(policy model.PlayerSessionCreationPolicy) error
```

Parameters

playerSessionCreationPolicy

String value that indicates whether the game session accepts new players.

Valid values include:

- **model.AcceptAll** – Accept all new player sessions.

- **model.DenyAll** – Deny all new player sessions.

Return value

Returns an error with an error message if failure occurs.

Example

This example sets the current game session's join policy to accept all players.

```
err := server.UpdatePlayerSessionCreationPolicy(model.AcceptAll)
```

GetGameSessionId()

Retrieves the ID of the game session hosted by the active server process.

Syntax

```
func GetGameSessionID() (string, error)
```

Parameters

This action has no parameters.

Return value

If successful, returns the game session ID and nil error. For idle processes that aren't yet activated with a game session, the call returns an empty string and nil error.

Example

```
gameSessionID, err := server.GetGameSessionID()
```

GetTerminationTime()

Returns the time that a server process is scheduled to be shut down if a termination time is available. A server process takes this action after receiving an `onProcessTerminate()` callback from Amazon GameLift Servers. Amazon GameLift Servers calls `onProcessTerminate()` for the following reasons:

- When the server process has reported poor health or hasn't responded to Amazon GameLift Servers.

- When terminating the instance during a scale-down event.
- When an instance is terminated due to a [spot-instance interruption](#).

Syntax

```
func GetTerminationTime() (int64, error)
```

Return value

If successful, returns the timestamp in epoch seconds that the server process is scheduled to shut down and a `nil` error termination. The value is the termination time, expressed in elapsed ticks from `0001 00:00:00`. For example, the date time value `2020-09-13 12:26:40 -000Z` is equal to `6373559680000000000` ticks. If no termination time is available, returns an error message.

Example

```
terminationTime, err := server.GetTerminationTime()
```

AcceptPlayerSession()

Notifies Amazon GameLift Servers that a player with the specified player session ID has connected to the server process and needs validation. Amazon GameLift Servers verifies that the player session ID is valid. After the player session is validated, Amazon GameLift Servers changes the status of the player slot from `RESERVED` to `ACTIVE`.

Syntax

```
func AcceptPlayerSession(playerSessionID string) error
```

Parameters

playerSessionId

Unique ID issued by Amazon GameLift Servers when a new player session is created.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example handles a connection request that includes validating and rejecting non-valid player session IDs.

```
func ReceiveConnectingPlayerSessionID(conn Connection, playerSessionID string) {  
    err := server.AcceptPlayerSession(playerSessionID)  
    if err != nil {  
        connection.Accept()  
    } else {  
        connection.Reject(err.Error())  
    }  
}
```

RemovePlayerSession()

Notifies Amazon GameLift Servers that a player has disconnected from the server process. In response, Amazon GameLift Servers changes the player slot to available.

Syntax

```
func RemovePlayerSession(playerSessionID string) error
```

Parameters

playerSessionId

Unique ID issued by Amazon GameLift Servers when a new player session is created.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
err := server.RemovePlayerSession(playerSessionID)
```

DescribePlayerSessions()

Retrieves player session data which includes settings, session metadata, and player data. Use this method to get information about the following:

- A single player session
- All player sessions in a game session
- All player sessions associated with a single player ID

Syntax

```
func DescribePlayerSessions(req request.DescribePlayerSessionsRequest)
    (result.DescribePlayerSessionsResult, error) {
    return srv.describePlayerSessions(&req)
}
```

Parameters

[DescribePlayerSessionsRequest](#)

A `DescribePlayerSessionsRequest` object describes which player sessions to retrieve.

Return value

If successful, returns a `DescribePlayerSessionsResult` object that contains a set of player session objects that fit the request parameters.

Example

This example requests all player sessions actively connected to a specified game session. By omitting *NextToken* and setting the *Limit* value to 10, Amazon GameLift Servers returns the first 10 player session records matching the request.

```
// create request
describePlayerSessionsRequest := request.NewDescribePlayerSessions()
describePlayerSessionsRequest.GameSessionID, _ = server.GetGameSessionID() // get ID
for the current game session
describePlayerSessionsRequest.Limit = 10 // return the
first 10 player sessions
describePlayerSessionsRequest.PlayerSessionStatusFilter = "ACTIVE" // Get all
player sessions actively connected to the game session

describePlayerSessionsResult, err :=
    server.DescribePlayerSessions(describePlayerSessionsRequest)
```

StartMatchBackfill()

Sends a request to find new players for open slots in a game session created with FlexMatch. For more information, see [FlexMatch backfill feature](#).

This action is asynchronous. If new players are matched, Amazon GameLift Servers delivers updated matchmaker data using the callback function `OnUpdateGameSession()`.

A server process can have only one active match backfill request at a time. To send a new request, first call [StopMatchBackfill\(\)](#) to cancel the original request.

Syntax

```
func StartMatchBackfill(req request.StartMatchBackfillRequest)
    (result.StartMatchBackfillResult, error)
```

Parameters

[StartMatchBackfillRequest](#)

A `StartMatchBackfillRequest` object communicates the following information:

- A ticket ID to assign to the backfill request. This information is optional; if no ID is provided, Amazon GameLift Servers generates one.
- The matchmaker to send the request to. The full configuration ARN is required. This value is in the game session's matchmaker data.
- The ID of the game session to backfill.
- The available matchmaking data for the game session's current players.

Return value

Returns a `StartMatchBackfillResult` object with the match backfill ticket ID, or failure with an error message.

Example

```
// form the request
startBackfillRequest := request.NewStartMatchBackfill()
startBackfillRequest.RequestID = "1111aaaa-22bb-33cc-44dd-5555eeee66ff" //
optional
```

```
startBackfillRequest.MatchmakingConfigurationArn = "arn:aws:gamelift:us-  
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig"  
var matchMaker model.MatchmakerData  
if err := matchMaker.UnmarshalJSON([]byte(gameSession.MatchmakerData)); err != nil {  
  
    return  
}  
startBackfillRequest.Players = matchMaker.Players  
res, err := server.StartMatchBackfill(startBackfillRequest)  
  
// Implement callback function for backfill  
func OnUpdateGameSession(myGameSession model.GameSession) {  
    // game-specific tasks to prepare for the newly matched players and update  
    matchmaker data as needed  
}
```

StopMatchBackfill()

Cancels an active match backfill request. For more information, see [FlexMatch backfill feature](#).

Syntax

```
func StopMatchBackfill(req request.StopMatchBackfillRequest) error
```

Parameters

StopMatchBackfillRequest

A StopMatchBackfillRequest object that identifies the matchmaking ticket to cancel:

- The ticket ID assigned to the backfill request.
- The matchmaker the backfill request was sent to.
- The game session associated with the backfill request.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
stopBackfillRequest := request.NewStopMatchBackfill() // Use this function to create
request
stopBackfillRequest.TicketID = "1111aaaa-22bb-33cc-44dd-5555eeee66ff"
stopBackfillRequest.MatchmakingConfigurationArn = "arn:aws:gamelift:us-
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig"

//error
err := server.StopMatchBackfill(stopBackfillRequest)
```

GetComputeCertificate()

Retrieves the path to the TLS certificate used to encrypt the network connection between the game server and your game client. You can use the certificate path when you register your compute device to a Amazon GameLift Servers Anywhere fleet. For more information, see [RegisterCompute](#).

Syntax

```
func GetComputeCertificate() (result.GetComputeCertificateResult, error)
```

Return value

Returns a `GetComputeCertificateResult` object that contains the following:

- **CertificatePath:** The path to the TLS certificate on your compute resource. When using an Amazon GameLift Servers managed fleet, this path contains:
 - `certificate.pem`: The end-user certificate. The full certificate chain is the combination of `certificateChain.pem` appended to this certificate.
 - `certificateChain.pem`: The certificate chain that contains the root certificate and intermediate certificates.
 - `rootCertificate.pem`: The root certificate.
 - `privateKey.pem`: The private key for the end-user certificate.
- **ComputeName:** The name of your compute resource.

Example

```
tlsCertificate, err := server.GetFleetRoleCredentials(getFleetRoleCredentialsRequest)
```


GetFleetRoleCredentials()

Retrieves the service role credentials that you create to extend permissions to your other AWS services to Amazon GameLift Servers. These credentials allow your game server to use your AWS resources. For more information, see [Set up an IAM service role for Amazon GameLift Servers](#).

Syntax

```
func GetFleetRoleCredentials(  
    req request.GetFleetRoleCredentialsRequest,  
) (result.GetFleetRoleCredentialsResult, error) {  
    return srv.getFleetRoleCredentials(&req)  
}
```

Parameters

[GetFleetRoleCredentialsRequest](#)

Role credentials that extend limited access to your AWS resources to the game server.

Return value

Returns a `GetFleetRoleCredentialsResult` object that contains the following:

- `AssumedRoleUserArn` - The Amazon Resource Name (ARN) of the user that the service role belongs to.
- `AssumedRoleId` - The ID of the user that the service role belongs to.
- `AccessKeyId` - The access key ID to authenticate and provide access to your AWS resources.
- `SecretAccessKey` - The secret access key ID for authentication.
- `SessionToken` - A token to identify the current active session interacting with your AWS resources.
- `Expiration` - The amount of time until your session credentials expire.

Example

```
// form the customer credentials request  
getFleetRoleCredentialsRequest := request.NewGetFleetRoleCredentials()  
getFleetRoleCredentialsRequest.RoleArn = "arn:aws:iam::123456789012:role/service-role/  
exampleGameLiftAction"
```

```
credentials, err := server.GetFleetRoleCredentials(getFleetRoleCredentialsRequest)
```

ListContainersNetworkInfo()

Retrieves network information for all containers running on the same instance, including each container's name, ID, local IP address, and container group type. Use this information to enable game server processes to discover and communicate with other containers that are running on the same instance.

This action is supported only on container fleets. When called from any other compute type, it returns an `UnsupportedComputeTypeException` error. Amazon GameLift Servers obtains the network information from a discovery server that runs locally on the instance.

Syntax

```
func ListContainersNetworkInfo() (result.ListContainersNetworkInfoResult, error)
```

Return value

If successful, returns a `ListContainersNetworkInfoResult` object that contains the following:

- `ContainersNetworkInfo` - A list of `ContainerNetworkInfo` entries, with one entry for each container running on the instance. Each entry contains the following:
 - `ContainerName` - The name of the container.
 - `ContainerID` - The unique identifier of the container.
 - `IPAddress` - The local IP address of the container on the instance.
 - `ContainerGroupType` - The type of container group that the container belongs to. Valid values are `GAME_SERVER` and `PER_INSTANCE`.

Example

```
networkInfo, err := server.ListContainersNetworkInfo()
if err != nil {
    fmt.Println("ListContainersNetworkInfo() failed. Error: ", err)
    return
}

for _, container := range networkInfo.ContainersNetworkInfo {
```

```
fmt.Printf("Container %s (%s) at %s, group type %s\n",
    container.ContainerName, container.ContainerID, container.IPAddress,
    container.ContainerGroupType)
}
```

Destroy()

Frees the Amazon GameLift Servers game server SDK from memory. As a best practice, call this method after `ProcessEnding()` and before terminating the process. If you're using an Anywhere fleet and you're not terminating server processes after every game session, call `Destroy()` and then `InitSDK()` to reinitialize before notifying Amazon GameLift Servers that the process is ready to host a game session with `ProcessReady()`.

Syntax

```
func Destroy() error {
    return srv.destroy()
}
```

Return value

Returns an error with an error message if the method fails.

Example

```
// operations to end game sessions and the server process
defer func() {
    err := server.ProcessEnding()
    server.Destroy()
    if err != nil {
        fmt.Println("ProcessEnding() failed. Error: ", err)
        os.Exit(-1)
    } else {
        os.Exit(0)
    }
}
```

C++ (Unreal) server SDK 5.x for Amazon GameLift Servers -- Actions

Use the Amazon GameLift Servers Unreal server SDK 5.x reference to help you prepare your multiplayer game for use with Amazon GameLift Servers. For details about the integration process,

see [Add Amazon GameLift Servers to your game server with the server SDK](#). If you're using the Amazon GameLift Servers plugin for Unreal, see also [Amazon GameLift Servers plugin for Unreal Engine](#).

Note

This topic describes the Amazon GameLift Servers C++ API that you can use when you build for the Unreal Engine. Specifically, this documentation applies to code that you compile with the `-DBUILD_FOR_UNREAL=1` option.

C++ (Unreal) server SDK 5.x for Amazon GameLift Servers -- Data types

Use the Amazon GameLift Servers Unreal server SDK 5.x reference to help you prepare your multiplayer game for use with Amazon GameLift Servers. For details about the integration process, see [Add Amazon GameLift Servers to your game server with the server SDK](#). If you're using the Amazon GameLift Servers plugin for Unreal, see also [Amazon GameLift Servers plugin for Unreal Engine](#).

Note

This topic describes the Amazon GameLift Servers C++ API that you can use when you build for the Unreal Engine. Specifically, this documentation applies to code that you compile with the `-DBUILD_FOR_UNREAL=1` option.

[C++ \(Unreal\) server SDK 5.x for Amazon GameLift Servers -- Actions](#)

Data types

- [FProcessParameters](#)
- [UpdateGameSession](#)
- [GameSession](#)
- [FServerParameters](#)
- [FStartMatchBackfillRequest](#)
- [FPlayer](#)
- [FGameLiftDescribePlayerSessionsRequest](#)

- [FStopMatchBackfillRequest](#)
- [FAttributeValue](#)
- [FGameLiftGetFleetRoleCredentialsRequest](#)
- [FGameLiftLongOutcome](#)
- [FGameLiftStringOutcome](#)
- [FGameLiftDescribePlayerSessionsOutcome](#)
- [FGameLiftDescribePlayerSessionsResult](#)
- [FGenericOutcome](#)
- [FGameLiftPlayerSession](#)
- [FGameLiftGetComputeCertificateOutcome](#)
- [FGameLiftGetComputeCertificateResult](#)
- [FGameLiftGetFleetRoleCredentialsOutcome](#)
- [FGetFleetRoleCredentialsResult](#)
- [FGameLiftListContainersNetworkInfoOutcome](#)
- [FGameLiftListContainersNetworkInfoResult](#)
- [FContainerNetworkInfo](#)
- [FGameLiftError](#)
- [Enums](#)

FProcessParameters

This data type contains the set of parameters sent to Amazon GameLift Servers in a [ProcessReady\(\)](#).

Properties	Description
LogParameters	An object with directory paths to files that are generated during a game session. Amazon GameLift Servers copies and stores the files for future access. Type: TArray<FString> Required: No

OnHealthCheck

The callback function that Amazon GameLift Servers invokes to request a health status report from the server process. Amazon GameLift Servers calls this function every 60 seconds and waits 60 seconds for a response. The server process returns TRUE if healthy, FALSE if not healthy. If no response is returned, Amazon GameLift Servers records the server process as not healthy.

This property is a delegate function defined as `DECLARE_DELEGATE_RetVal(bool, FOnHealthCheck)` ;

Type: FOnHealthCheck

Required: No

OnProcessTerminate

The callback function that Amazon GameLift Servers invokes to force the server process to shut down. After calling this function, Amazon GameLift Servers waits 5 minutes for the server process to shut down and respond with a [ProcessEnding\(\)](#) call before it shuts down the server process.

Type: FSimpleDelegate

Required: Yes

OnStartGameSession

The callback function that Amazon GameLift Servers invokes to activate a new game session. Amazon GameLift Servers calls this function in response to a client request [CreateGameSession](#). The callback function passes a [GameSession](#) object.

This property is a delegate function defined as `DECLARE_DELEGATE_OneParam(FOnStartGameSession, Aws::GameLift::Server::Model::GameSession);`

Type: FOnStartGameSession

Required: Yes

OnUpdateGameSession

The callback function that Amazon GameLift Servers invokes to pass an updated game session object to the server process. Amazon GameLift Servers calls this function when a match backfill request has been processed to provide updated matchmaker data. It passes a [GameSession](#) object, a status update (`updateReason`), and the match backfill ticket ID.

This property is a delegate function defined as `DECLARE_DELEGATE_OneParam(FOnUpdateGameSession, Aws::GameLift::Server::Model::UpdateGameSession);`

Type: FOnUpdateGameSession

Required: No

Port

The port number the server process listens on for new player connections. The value must fall into the port range configured for any fleet deploying this game server build. This port number is included in game session and player session objects, which game sessions use when connecting to a server process.

Type: `int`

Required: Yes

UpdateGameSession

This data type updates to a game session object, which includes the reason that the game session was updated and the related backfill ticket ID if backfill is used to fill player sessions in the game session.

Properties	Description
GameSession	A GameSession object. The <code>GameSession</code> object contains properties describing a game session. Type: <code>Aws::GameLift::Server::GameSession</code> Required: No
UpdateReason	The reason that the game session is being updated. Type: <code>enum class UpdateReason</code> • <code>MATCHMAKING_DATA_UPDATED</code> • <code>BACKFILL_FAILED</code> • <code>BACKFILL_TIMED_OUT</code> • <code>BACKFILL_CANCELLED</code>

Properties	Description
	Required: No
BackfillTicketId	The ID of the backfill ticket attempting to update the game session. Type: <code>char[]</code> Required: No

GameSession

This data type provides details of a game session.

Properties	Description
GameSessionId	A unique identifier for the game session. A game session ARN has the following format: <code>arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token></code> . Type: <code>char[]</code> Required: No
Name	A descriptive label of the game session. Type: <code>char[]</code> Required: No
FleetId	A unique identifier for the fleet that the game session is running on. Type: <code>char[]</code> Required: No

Properties	Description
MaximumPlayerSessionCount	The maximum number of player connections to the game session. Type: <code>int</code> Required: No
Port	The port number for the game session. To connect to an Amazon GameLift Servers game server, an app needs both the IP address and port number. Type: <code>int</code> Required: No
IpAddress	The IP address of the game session. To connect to an Amazon GameLift Servers game server, an app needs both the IP address and port number. Type: <code>char[]</code> Required: No
GameSessionData	A set of custom game session properties, formatted as a single string value. Type: <code>char[]</code> Required: No

Properties	Description
MatchmakerData	Information about the matchmaking process that was used to create the game session, in JSON syntax, formatted as a string. In addition to the matchmaking configuration used, it contains data on all players assigned to the match, including player attributes and team assignments. Type: <code>char[]</code> Required: No
GameProperties	A set of custom properties for a game session, formatted as key:value pairs. These properties are passed with a request to start a new game session. Type: <code>GameProperty[]</code> Required: No

Properties	Description
DnsName	The DNS identifier assigned to the instance that's running the game session. Values have the following format: • TLS-enabled fleets: <unique identifier>.<region identifier>.amazon.gamelift.com . • Non-TLS-enabled fleets: ec2-<unique identifier>.compute.amazonaws.com . When connecting to a game session that's running on a TLS-enabled fleet, you must use the DNS name, not the IP address. Type: char[] Required: No

FServerParameters

Information used to maintain the connection between an Amazon GameLift Servers Anywhere server and the Amazon GameLift Servers service. This information is used when launching new server processes with [InitSDK\(\)](#). For servers hosted on Amazon GameLift Servers managed EC2 instances, use an empty object.

Properties	Description
websocketUrl	The <code>GameLiftServerSdkEndpoint</code> Amazon GameLift Servers returns when you RegisterCompute for a Amazon GameLift Servers Anywhere compute resource. Type: char[]

Properties	Description
	Required: Yes
processId	A unique identifier registered to the server process hosting your game. Type: <code>char[]</code> Required: Yes
hostId	The HostID is the ComputeName used when you registered your compute. For more information see, RegisterCompute. Type: <code>char[]</code> Required: Yes
fleetId	The unique identifier of the fleet that the compute is registered to. For more information see, RegisterCompute. Type: <code>char[]</code> Required: Yes
authToken	The authentication token generated by Amazon GameLift Servers that authenticates your server to Amazon GameLift Servers. For more information see, GetComputeAuthToken. Type: <code>char[]</code> Required: Yes

FStartMatchBackfillRequest

Information used to create a matchmaking backfill request. The game server communicates this information to Amazon GameLift Servers in a [StartMatchBackfill\(\)](#) call.

Properties	Description
GameSessionArn	A unique game session identifier. The API operation GetGameSessionId returns the identifier in ARN format. Type: <code>char[]</code> Required: Yes
MatchmakingConfigurationArn	A unique identifier, in the form of an ARN, for the matchmaker to use for this request. The matchmaker ARN for the original game session is in the game session object in the matchmaker data property. Learn more about matchmaker data in Work with matchmaker data. Type: <code>char[]</code> Required: Yes
Players	A set of data representing all players who are in the game session. The matchmaker uses this information to search for new players who are good matches for the current players. Type: <code>TArray<FPlayer></code> Required: Yes
TicketId	A unique identifier for a matchmaking or match backfill request ticket. If you don't provide a value, Amazon GameLift Servers generates one. Use this identifier to track the match backfill ticket status or cancel the request if needed. Type: <code>char[]</code>

Properties	Description
	Required: No

FPlayer

This data type represents a player in matchmaking. When starting a matchmaking request, a player has a player ID, attributes, and possibly latency data. Amazon GameLift Servers adds team information after a match is made.

Properties	Description
LatencyInMS	A set of values expressed in milliseconds that indicate the amount of latency that a player experiences when connected to a location. If this property is used, the player is only matched for locations listed. If a matchmaker has a rule that evaluates player latency, players must report latency to be matched. Type: TMap>FString, int32< Required: No
PlayerAttributes	A collection of key:value pairs containing player information for use in matchmaking. Player attribute keys must match the PlayerAttributes used in a matchmaking rule set. For more information about player attributes, see AttributeValue. Type: TMap>FString, FAttributeValue< Required: No
PlayerId	A unique identifier for a player.

Properties	Description
	Type: <code>std::string</code> Required: No
Team	The name of the team that the player is assigned to in a match. You define team name in the matchmaking rule set. Type: <code>FString</code> Required: No

FGameLiftDescribePlayerSessionsRequest

An object that specifies which player sessions to retrieve. The server process provides this information with a [DescribePlayerSessions\(\)](#) call to Amazon GameLift Servers.

Properties	Description
GameSessionId	A unique game session identifier. Use this parameter to request all player sessions for the specified game session. Game session ID format is <code>FString</code> . The <code>GameSessionID</code> is a custom ID string or a Type: <code>std::string</code> Required: No
PlayerSessionId	The unique identifier for a player session. Use this parameter to request a single specific player session. Type: <code>FString</code> Required: No

Properties	Description
PlayerId	The unique identifier for a player. Use this parameter to request all player sessions for a specific player. See Generate player IDs. Type: FString Required: No
PlayerSessionStatusFilter	The player session status to filter results on. Possible player session statuses include: • RESERVED – The player session request was received, but the player hasn't connected to the server process or been validated. • ACTIVE – The player was validated by the server process and is connected. • COMPLETED – The player connection dropped. • TIMEDOUT – A player session request was received, but the player didn't connect or wasn't validated within the time-out limit (60 seconds). Type: FString Required: No
NextToken	The token indicating the start of the next page of results. To specify the start of the result set, don't provide a value. If you provide a player session ID, this parameter is ignored. Type: FString Required: No

Properties	Description
Limit	The maximum number of results to return. If you provide a player session ID, this parameter is ignored. Type: int Required: No

FStopMatchBackfillRequest

Information used to cancel a matchmaking backfill request. The game server communicates this information to Amazon GameLift Servers service in a [StopMatchBackfill\(\)](#) call.

Properties	Description
GameSessionArn	A unique game session identifier of the request being canceled. Type: FString Required: Yes
MatchmakingConfigurationArn	A unique identifier of the matchmaker this request was sent to. Type: FString Required: Yes
TicketId	A unique identifier of the backfill request ticket to be canceled. Type: FString Required: Yes

FAttributeValue

Use these values in [FPlayer](#) attribute key-value pairs. This object lets you specify an attribute value using any of the valid data types: string, number, string array, or data map. Each `FAttributeValue` object can use only one of the available properties.

Properties	Description
attrType	Specifies the type of attribute value. Type: An <code>FAttributeType</code> enum value. Required: No
S	Represents a string attribute value. Type: <code>FString</code> Required: No
N	Represents a numeric attribute value. Type: <code>double</code> Required: No
SL	Represents an array of string attribute values. Type: <code>TArray<FString></code> Required: No
SDM	Represents a dictionary of string keys and double values. Type: <code>TMap<FString, double></code> Required: No

FGameLiftGetFleetRoleCredentialsRequest

This data type provides role credentials that extend limited access to your AWS resources to the game server. For more information see, [Set up an IAM service role for Amazon GameLift Servers](#).

Properties	Description
RoleArn	The Amazon Resource Name (ARN) of the service role that extends limited access to your AWS resources. Type: FString Required: No
RoleSessionName	The name of the session describing the use of the role credentials. Type: FString Required: No

FGameLiftLongOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: long Required: No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. Type: long&& Required: No

Properties	Description
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "FGameLiftError" Required: No

FGameLiftStringOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: FString Required: No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. Type: FString&& Required: No
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful.

Properties	Description
	Type: the section called “FGameLiftError” Required: No

FGameLiftDescribePlayerSessionsOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: the section called “FGameLiftDescribePlayerSessionsResult” Required: No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. Type: FGameLiftDescribePlayerSessionsResult&& Required: No
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called “FGameLiftError” Required: No

FGameLiftDescribePlayerSessionsResult

Properties	Description
PlayerSessions	Type: TArray<FGameLiftPlayerSession> Required: Yes
NextToken	The token indicating the start of the next page of results. To specify the start of the result set, don't provide a value. If you provide a player session ID, this parameter is ignored. Type: FString Required: No
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "FGameLiftError" Required: No

FGenericOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Success	Whether the action was successful or not. Type: bool Required: Yes

Properties	Description
Error	The error that occurred if the action was unsuccessful. Type: the section called “FGameLiftError” Required: No

FGameLiftPlayerSession

Properties	Description
CreationTime	Type: long Required: Yes
FleetId	Type: FString Required: Yes
GameSessionId	Type: FString Required: Yes
IpAddress	Type: FString Required: Yes
PlayerData	Type: FString Required: Yes
PlayerId	Type: FString Required: Yes
PlayerSessionId	Type: FString Required: Yes
Port	Type: int

Properties	Description
	Required: Yes
Status	Type: A PlayerSessionStatus enum . Required: Yes
TerminationTime	Type: long Required: Yes
DnsName	Type: FString Required: Yes

FGameLiftGetComputeCertificateOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: the section called "FGameLiftGetComputeCertificateResult" Required: No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. Type: FGameLiftGetComputeCertificateResult&& Required: No
Success	Whether the action was successful or not. Type: bool Required: Yes

Properties	Description
Error	The error that occurred if the action was unsuccessful. Type: the section called “FGameLiftError” Required: No

FGameLiftGetComputeCertificateResult

The path to the TLS certificate on your compute and the compute's host name.

Properties	Description
CertificatePath	Type: FString Required: Yes
ComputeName	Type: FString Required: Yes

FGameLiftGetFleetRoleCredentialsOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: the section called “FGetFleetRoleCredentialsResult” Required: No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. Type: FGameLiftGetFleetRoleCredentialsResult&&

Properties	Description
	Required: No
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "FGameLiftError" Required: No

FGetFleetRoleCredentialsResult

Properties	Description
AccessKeyId	The access key ID to authenticate and provide access to your AWS resources. Type: FString Required: No
AssumedRoleId	The ID of the user that the service role belongs to. Type: FString Required: No
AssumedRoleUserArn	The Amazon Resource Name (ARN) of the user that the service role belongs to. Type: FString Required: No
Expiration	The amount of time until your session credentials expire.

Properties	Description
	Type: FDateTime Required: No
SecretAccessKey	The secret access key ID for authentication. Type: FString Required: No
SessionToken	A token to identify the current active session interacting with your AWS resources. Type: FString Required: No
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called "GameLiftError" Required: No

FGameLiftListContainersNetworkInfoOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. Type: the section called "FGameLiftListContainersNetworkInfoResult"

Properties	Description
	Required: No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. Type: FGameLiftListContainersNetworkInfoResult&& Required: No
Success	Whether the action was successful or not. Type: bool Required: Yes
Error	The error that occurred if the action was unsuccessful. Type: the section called “FGameLiftError” Required: No

FGameLiftListContainersNetworkInfoResult

Network information for all containers running on the same instance as the calling game server process.

Properties	Description
ContainersNetworkInfo	The list of network information for each container running on the instance. Type: TArray<the section called “FContainerNetworkInfo”> Required: No

FContainerNetworkInfo

Network information for a single container running on the instance.

Properties	Description
ContainerName	The name of the container, as defined in the container group definition. Type: FString Required: No
ContainerId	The unique identifier of the container. Type: FString Required: No
IpAddress	The container's local IPv4 address on the Docker bridge network. Type: FString Required: No
ContainerGroupType	The type of container group that the container belongs to. Type: An EContainerGroupType enum. Required: No

FGameLiftError

Properties	Description
ErrorType	The type of error. Type: A GameLiftErrorType enum.

Properties	Description
	Required: No
ErrorMessage	The error message. Type: <code>std::string</code> Required: No
ErrorMessage	The error message. Type: <code>std::string</code> Required: No

Enums

Enums defined for the server SDK for Amazon GameLift Servers (Unreal) are defined as follows:

FAttributeType

- **NONE**
- **STRING**
- **DOUBLE**
- **STRING_LIST**
- **STRING_DOUBLE_MAP**

GameLiftErrorType

String value indicating the error type. Valid values include:

- **SERVICE_CALL_FAILED** – A call to an AWS service has failed.
- **LOCAL_CONNECTION_FAILED** – The local connection to Amazon GameLift Servers failed.
- **NETWORK_NOT_INITIALIZED** – The network has not been initialized.
- **GAMESESSION_ID_NOT_SET** – The game session ID has not been set.
- **BAD_REQUEST_EXCEPTION**
- **INTERNAL_SERVICE_EXCEPTION**
- **ALREADY_INITIALIZED** – The Amazon GameLift Servers Server or Client has already been initialized with `Initialize()`.

- **FLEET_MISMATCH** – The target fleet does not match the fleet of a gameSession or playerSession.
- **GAMELIFT_CLIENT_NOT_INITIALIZED** – The Amazon GameLift Servers client has not been initialized.
- **GAMELIFT_SERVER_NOT_INITIALIZED** – The Amazon GameLift Servers server has not been initialized.
- **GAME_SESSION_ENDED_FAILED** – the server SDK for Amazon GameLift Servers could not contact the service to report the game session ended.
- **GAME_SESSION_NOT_READY** – The Amazon GameLift Servers Server Game Session was not activated.
- **GAME_SESSION_READY_FAILED** – the server SDK for Amazon GameLift Servers could not contact the service to report the game session is ready.
- **INITIALIZATION_MISMATCH** – A client method was called after Server::Initialize(), or vice versa.
- **NOT_INITIALIZED** – The Amazon GameLift Servers Server or Client has not been initialized with Initialize().
- **NO_TARGET_ALIASID_SET** – A target aliasId has not been set.
- **NO_TARGET_FLEET_SET** – A target fleet has not been set.
- **PROCESS_ENDING_FAILED** – the server SDK for Amazon GameLift Servers could not contact the service to report the process is ending.
- **PROCESS_NOT_ACTIVE** – The server process is not yet active, not bound to a GameSession, and cannot accept or process PlayerSessions.
- **PROCESS_NOT_READY** – The server process is not yet ready to be activated.
- **PROCESS_READY_FAILED** – the server SDK for Amazon GameLift Servers could not contact the service to report the process is ready.
- **SDK_VERSION_DETECTION_FAILED** – SDK version detection failed.
- **STX_CALL_FAILED** – A call to the XStx server backend component has failed.
- **STX_INITIALIZATION_FAILED** – The XStx server backend component has failed to initialize.
- **UNEXPECTED_PLAYER_SESSION** – An unregistered player session was encountered by the server.
- **WEBSOCKET_CONNECT_FAILURE**
- **WEBSOCKET_CONNECT_FAILURE_FORBIDDEN**

- **WEBSOCKET_CONNECT_FAILURE_INVALID_URL**
- **WEBSOCKET_CONNECT_FAILURE_TIMEOUT**
- **WEBSOCKET_RETRIABLE_SEND_MESSAGE_FAILURE** – Retriable failure to send a message to the GameLift Service WebSocket.
- **WEBSOCKET_SEND_MESSAGE_FAILURE** – Failure to send a message to the GameLift Service WebSocket.
- **MATCH_BACKFILL_REQUEST_VALIDATION** – Validation of the request failed.
- **PLAYER_SESSION_REQUEST_VALIDATION** – Validation of the request failed.
- **UNSUPPORTED_COMPUTE_TYPE_EXCEPTION** – The API that was called was unsupported on the compute type.

EPlayerSessionCreationPolicy

String value indicating whether the game session accepts new players. Valid values include:

- **ACCEPT_ALL** – Accept all new player sessions.
- **DENY_ALL** – Deny all new player sessions.
- **NOT_SET** – The game session is not set to accept or deny new player sessions.

EPlayerSessionStatus

- **ACTIVE**
- **COMPLETED**
- **NOT_SET**
- **RESERVED**
- **TIMEDOUT**

EContainerGroupType

The type of container group that a container belongs to. Valid values include:

- **GAME_SERVER** – A game server replica container group. An instance can run multiple game server container groups.
- **PER_INSTANCE** – A per-instance daemon container group. An instance runs exactly one per-instance container group.

[C++ \(Unreal\) server SDK 5.x for Amazon GameLift Servers -- Data types](#)

Topics

- [GetSdkVersion\(\)](#)
- [InitSDK\(\)](#)
- [InitSDK\(\)](#)
- [ProcessReady\(\)](#)
- [ProcessEnding\(\)](#)
- [ActivateGameSession\(\)](#)
- [UpdatePlayerSessionCreationPolicy\(\)](#)
- [GetGameSessionId\(\)](#)
- [GetTerminationTime\(\)](#)
- [AcceptPlayerSession\(\)](#)
- [RemovePlayerSession\(\)](#)
- [DescribePlayerSessions\(\)](#)
- [StartMatchBackfill\(\)](#)
- [StopMatchBackfill\(\)](#)
- [GetComputeCertificate\(\)](#)
- [GetFleetRoleCredentials\(\)](#)
- [ListContainersNetworkInfo\(\)](#)
- [Destroy\(\)](#)

GetSdkVersion()

Returns the current version number of the SDK built into the server process.

Syntax

```
FGameLiftStringOutcome GetSdkVersion();
```

Return value

If successful, returns the current SDK version as an [the section called “FGameLiftStringOutcome”](#) object. The returned object includes the version number (example 5.0.0). If not successful, returns an error message.

Example

```
Aws::GameLift::AwsStringOutcome SdkVersionOutcome =  
    Aws::GameLift::Server::GetSdkVersion();
```

InitSDK()

Initializes the Amazon GameLift Servers SDK for a managed EC2 fleet. Call this method on launch, before any other initialization related to Amazon GameLift Servers occurs. This method reads server parameters from the host environment to set up communication between the server and the Amazon GameLift Servers service. It uses an idempotency token, so you safely retry this call when it fails.

Syntax

```
FGameLiftGenericOutcome InitSDK()
```

Return value

If successful, returns an `InitSdkOutcome` object indicating that the server process is ready to call [ProcessReady\(\)](#).

Example

```
//Call InitSDK to establish a local connection with the Amazon GameLift Servers Agent  
to enable further communication.  
FGameLiftGenericOutcome initSdkOutcome = GameLiftSdkModule->InitSDK();
```

InitSDK()

Initializes the Amazon GameLift Servers SDK for an Anywhere fleet or managed container fleet. Call this method on launch, before any other initialization related to Amazon GameLift Servers occurs. This method requires explicit server parameters to set up communication between the server and the Amazon GameLift Servers service. It uses an idempotency token, so you safely retry this call when it fails.

Syntax

```
FGameLiftGenericOutcome InitSDK(serverParameters)
```

Parameters

[FServerParameters](#)

To initialize a game server on an Amazon GameLift Servers Anywhere fleet, construct a `ServerParameters` object with the following information:

- The URL of the WebSocket used to connect to your game server.
- The ID of the process used to host your game server.
- The ID of the compute hosting your game server processes.
- The ID of the Amazon GameLift Servers fleet containing your Amazon GameLift Servers Anywhere compute.
- The authorization token generated by the Amazon GameLift Servers operation.

Return value

If successful, returns an `InitSdkOutcome` object indicating that the server process is ready to call [ProcessReady\(\)](#).

Note

If calls to `InitSDK()` are failing for game builds deployed to Anywhere fleets, check the `ServerSdkVersion` parameter used when creating the build resource. You must explicitly set this value to the server SDK version in use. The default value for this parameter is 4.x, which is not compatible. To resolve this issue, create a new build and deploy it to a new fleet.

Example

```
//Define the server parameters
FServerParameters serverParameters;
serverParameters.m_authToken = "1111aaaa-22bb-33cc-44dd-5555eeee66ff";
serverParameters.m_fleetId = "arn:aws:gamelift:us-west-1:111122223333:fleet/
fleet-9999ffff-88ee-77dd-66cc-5555bbbb44aa";
serverParameters.m_hostId = "HardwareAnywhere";
serverParameters.m_processId = "PID1234";
serverParameters.m_webSocketUrl = "wss://us-west-1.api.amazongamelift.com";
```

```
//Call InitSDK to establish a local connection with the Amazon GameLift Servers Agent
to enable further communication.
FGameLiftGenericOutcome initSdkOutcome = GameLiftSdkModule->InitSDK(serverParameters);
```

ProcessReady()

Notifies Amazon GameLift Servers that the server process is ready to host game sessions. Call this method after invoking [InitSDK\(\)](#). This method should be called only once per process.

Syntax

```
GenericOutcome ProcessReady(const Aws::GameLift::Server::ProcessParameters
&processParameters);
```

Parameters

processParameters

An [FProcessParameters](#) object communicating the following information about the server process:

- Names of callback methods implemented in the game server code that the Amazon GameLift Servers service invokes to communicate with the server process.
- Port number that the server process is listening on.
- Path to any game session-specific files that you want Amazon GameLift Servers to capture and store.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example illustrates both the [ProcessReady\(\)](#) call and delegate function implementations.

```
//Calling ProcessReady tells Amazon GameLift Servers this game server is ready to
receive incoming game sessions!
UE_LOG(GameServerLog, Log, TEXT("Calling Process Ready"));
FGameLiftGenericOutcome processReadyOutcome = GameLiftSdkModule-
>ProcessReady(*params);
```

ProcessEnding()

Notifies Amazon GameLift Servers that the server process is terminating. Call this method after all other cleanup tasks (including shutting down the active game session) and before terminating the process. Depending on the result of `ProcessEnding()`, the process exits with success (0) or error (-1) and generates a fleet event. If the process terminates with an error, the fleet event generated is `SERVER_PROCESS_TERMINATED_UNHEALTHY`).

Syntax

```
FGameLiftGenericOutcome ProcessEnding()
```

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
//OnProcessTerminate callback. Amazon GameLift Servers will invoke this callback before
//shutting down an instance hosting this game server.
//It gives this game server a chance to save its state, communicate with services,
//etc., before being shut down.
//In this case, we simply tell Amazon GameLift Servers we are indeed going to shutdown.
params->OnTerminate.BindLambda( [=]() {
    UE_LOG(GameServerLog, Log, TEXT("Game Server Process is terminating"));
    GameLiftSdkModule->ProcessEnding();
});
```

ActivateGameSession()

Notifies Amazon GameLift Servers that the server process has activated a game session and is now ready to receive player connections. This action should be called as part of the `onStartGameSession()` callback function, after all game session initialization.

Syntax

```
FGameLiftGenericOutcome ActivateGameSession()
```

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example shows `ActivateGameSession()` called as part of the `onStartGameSession()` delegate function.

```
//When a game session is created, Amazon GameLift Servers sends an activation request
//to the game server and passes along the game session object containing game properties
//and other settings.
//Here is where a game server should take action based on the game session object.
//Once the game server is ready to receive incoming player connections, it should
//invoke GameLiftServerAPI.ActivateGameSession()
auto onGameSession = [=](Aws::GameLift::Server::Model::GameSession gameSession)
{
    FString gameSessionId = FString(gameSession.GetGameSessionId());
    UE_LOG(GameServerLog, Log, TEXT("GameSession Initializing: %s"), *gameSessionId);
    GameLiftSdkModule->ActivateGameSession();
};
```

UpdatePlayerSessionCreationPolicy()

Updates the current game session's ability to accept new player sessions. A game session can be set to either accept or deny all new player sessions.

Syntax

```
FGameLiftGenericOutcome UpdatePlayerSessionCreationPolicy(EPlayerSessionCreationPolicy
policy)
```

Parameters

playerCreationSessionPolicy

String value indicating whether the game session accepts new players.

Valid values include:

- **ACCEPT_ALL** – Accept all new player sessions.
- **DENY_ALL** – Deny all new player sessions.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example sets the current game session's join policy to accept all players.

```
FGameLiftGenericOutcome outcome = GameLiftSdkModule-  
>UpdatePlayerSessionCreationPolicy(Aws::GameLift::Model::EPlayerSessionCreationPolicy::ACCEPT_A
```

GetGameSessionId()

Retrieves the ID of the game session hosted by the active server process.

For idle processes that aren't activated with a game session, the call returns a [the section called "FGameLiftError"](#).

Syntax

```
FGameLiftStringOutcome GetGameSessionId()
```

Parameters

This action has no parameters.

Return value

If successful, returns the game session ID as an [the section called "FGameLiftStringOutcome"](#) object. If not successful, returns an error message."

For idle processes that aren't activated with a game session, the call returns Success=True and GameSessionId="".

Example

```
//When a game session is created, Amazon GameLift Servers sends an activation request  
to the game server and passes along the game session object containing game properties  
and other settings.  
//Here is where a game server should take action based on the game session object.  
//Once the game server is ready to receive incoming player connections, it should  
invoke GameLiftServerAPI.ActivateGameSession()  
auto onGameSession = [=](Aws::GameLift::Server::Model::GameSession gameSession)  
{  
    FString gameSessionId = FString(gameSession.GetGameSessionId());  
    UE_LOG(GameServerLog, Log, TEXT("GameSession Initializing: %s"), *gameSessionId);
```



```
GameLiftSdkModule->ActivateGameSession();  
};
```

GetTerminationTime()

Returns the time that a server process is scheduled to be shut down, if a termination time is available. A server process takes action after receiving an `onProcessTerminate()` callback from Amazon GameLift Servers. Amazon GameLift Servers calls `onProcessTerminate()` for the following reasons:

- When the server process has reported poor health or has not responded to Amazon GameLift Servers.
- When terminating the instance during a scale-down event.
- When an instance is terminated due to a [spot-instance interruption](#).

Syntax

```
AwsDateTimeOutcome GetTerminationTime()
```

Return value

If successful, returns the termination time as an `AwsDateTimeOutcome` object. The value is the termination time, expressed in elapsed ticks since 0001 00:00:00. For example, the date time value 2020-09-13 12:26:40 -000Z is equal to 637355968000000000 ticks. If no termination time is available, returns an error message.

If the process hasn't received a `ProcessParameters.OnProcessTerminate()` callback, an error message is returned. For more information about shutting down a server process, see [Respond to a server process shutdown notification](#).

Example

```
AwsDateTimeOutcome TermTimeOutcome = GameLiftSdkModule->GetTerminationTime();
```

AcceptPlayerSession()

Notifies Amazon GameLift Servers that a player with the specified player session ID has connected to the server process and needs validation. Amazon GameLift Servers verifies that the player

session ID is valid. After the player session is validated, Amazon GameLift Servers changes the status of the player slot from RESERVED to ACTIVE.

Syntax

```
FGameLiftGenericOutcome AcceptPlayerSession(const FString& playerId)
```

Parameters

`playerSessionId`

Unique ID issued by Amazon GameLift Servers when a new player session is created.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example handles a connection request that includes validating and rejecting non-valid player session IDs.

```
bool GameLiftManager::AcceptPlayerSession(const FString& playerId, const
    FString& gameId)
{
    #if WITH_GAMELIFT
        UE_LOG(GameServerLog, Log, TEXT("Accepting GameLift PlayerSession: %s . PlayerId:
        %s"), *playerSessionId, *playerId);
        FString gsId = GetCurrentGameSessionId();
        if (gsId.IsEmpty()) {
            UE_LOG(GameServerLog, Log, TEXT("No GameLift GameSessionId. Returning early!"));
            return false;
        }

        if (!GameLiftSdkModule->AcceptPlayerSession(playerSessionId).IsSuccess()) {
            UE_LOG(GameServerLog, Log, TEXT("PlayerSession not Accepted.));
            return false;
        }

        // Add PlayerSession from internal data structures keeping track of connected players
        connectedPlayerSessionIds.Add(playerSessionId);
```

```
idToPlayerSessionMap.Add(playerSessionId, PlayerSession{ playerId,  
playerSessionId });  
return true;  
#else  
return false;  
#endif  
}
```

RemovePlayerSession()

Notifies Amazon GameLift Servers that a player has disconnected from the server process. In response, Amazon GameLift Servers changes the player slot to available.

Syntax

```
FGameLiftGenericOutcome RemovePlayerSession(const FString& playerSessionId)
```

Parameters

playerSessionId

Unique ID issued by Amazon GameLift Servers when a new player session is created.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
bool GameLiftManager::RemovePlayerSession(const FString& playerSessionId)  
{  
    #if WITH_GAMELIFT  
    UE_LOG(GameServerLog, Log, TEXT("Removing GameLift PlayerSession: %s"),  
*playerSessionId);  
  
    if (!GameLiftSdkModule->RemovePlayerSession(playerSessionId).IsSuccess()) {  
        UE_LOG(GameServerLog, Log, TEXT("PlayerSession Removal Failed"));  
        return false;  
    }  
  
    // Remove PlayerSession from internal data structures that are keeping track of  
    connected players
```

```
connectedPlayerSessionIds.Remove(playerSessionId);
idToPlayerSessionMap.Remove(playerSessionId);

// end the session if there are no more players connected
if (connectedPlayerSessionIds.Num() == 0) {
    EndSession();
}

return true;
#else
return false;
#endif
}
```

DescribePlayerSessions()

Retrieves player session data which includes settings, session metadata, and player data. Use this method to get information about the following:

- A single player session
- All player sessions in a game session
- All player sessions associated with a single player ID

Syntax

```
FGameLiftDescribePlayerSessionsOutcome DescribePlayerSessions(const
FGameLiftDescribePlayerSessionsRequest &describePlayerSessionsRequest)
```

Parameters

[FGameLiftDescribePlayerSessionsRequest](#)

A [the section called “FGameLiftDescribePlayerSessionsRequest”](#) object that describes which player sessions to retrieve.

Return value

If successful, returns a [the section called “FGameLiftDescribePlayerSessionsOutcome”](#) object containing a set of player session objects that fit the request parameters.

Example

This example requests all player sessions actively connected to a specified game session. By omitting *NextToken* and setting the *Limit* value to 10, Amazon GameLift Servers returns the first 10 player session records matching the request.

```
void GameLiftManager::DescribePlayerSessions()
{
    #if WITH_GAMELIFT
        FString localPlayerSessions;
        for (auto& psId : connectedPlayerSessionIds)
        {
            PlayerSession ps = idToPlayerSessionMap[psId];
            localPlayerSessions += FString::Printf(TEXT("%s : %s ; "), *(ps.playerSessionId),
*(ps.playerId));
        }
        UE_LOG(GameServerLog, Log, TEXT("LocalPlayerSessions: %s"), *localPlayerSessions);

        UE_LOG(GameServerLog, Log, TEXT("Describing PlayerSessions in this GameSession"));
        FGameLiftDescribePlayerSessionsRequest request;
        request.m_gameSessionId = GetCurrentGameSessionId();

        FGameLiftDescribePlayerSessionsOutcome outcome = GameLiftSdkModule-
>DescribePlayerSessions(request);
        LogDescribePlayerSessionsOutcome(outcome);
    #endif
}
```

StartMatchBackfill()

Sends a request to find new players for open slots in a game session created with FlexMatch. For more information, see [FlexMatch backfill feature](#).

This action is asynchronous. If new players are matched, Amazon GameLift Servers delivers updated matchmaker data using the callback function `OnUpdateGameSession()`.

A server process can have only one active match backfill request at a time. To send a new request, first call [StopMatchBackfill\(\)](#) to cancel the original request.

Syntax

```
FGameLiftStringOutcome StartMatchBackfill (FStartMatchBackfillRequest
&startBackfillRequest);
```

Parameters

FStartMatchBackfillRequest

A StartMatchBackfillRequest object that communicates the following information:

- A ticket ID to assign to the backfill request. This information is optional; if no ID is provided, Amazon GameLift Servers will generate one.
- The matchmaker to send the request to. The full configuration ARN is required. This value is in the game session's matchmaker data.
- The ID of the game session to backfill.
- The available matchmaking data for the game session's current players.

Return value

Returns a StartMatchBackfillOutcome object with the match backfill ticket ID, or failure with an error message.

Example

```
FGameLiftStringOutcome FGameLiftServerSDKModule::StartMatchBackfill(const
FStartMatchBackfillRequest& request)
{
    #if WITH_GAMELIFT
    Aws::GameLift::Server::Model::StartMatchBackfillRequest sdkRequest;
    sdkRequest.SetTicketId(TCHAR_TO_UTF8(*request.m_ticketId));
    sdkRequest.SetGameSessionArn(TCHAR_TO_UTF8(*request.m_gameSessionArn));

    sdkRequest.SetMatchmakingConfigurationArn(TCHAR_TO_UTF8(*request.m_matchmakingConfigurationArn));
    for (auto player : request.m_players) {
        Aws::GameLift::Server::Model::Player sdkPlayer;
        sdkPlayer.SetPlayerId(TCHAR_TO_UTF8(*player.m_playerId));
        sdkPlayer.SetTeam(TCHAR_TO_UTF8(*player.m_team));
        for (auto entry : player.m_latencyInMs) {
            sdkPlayer.WithLatencyMs(TCHAR_TO_UTF8(*entry.Key), entry.Value);
        }
    }
}
```

```

    std::map<std::string, Aws::GameLift::Server::Model::AttributeValue>
    sdkAttributeMap;
    for (auto attributeEntry : player.m_playerAttributes) {
        FAttributeValue value = attributeEntry.Value;
        Aws::GameLift::Server::Model::AttributeValue attribute;
        switch (value.m_type) {
            case FAttributeType::STRING:
                attribute =
                Aws::GameLift::Server::Model::AttributeValue(TCHAR_TO_UTF8(*value.m_S));
                break;
            case FAttributeType::DOUBLE:
                attribute = Aws::GameLift::Server::Model::AttributeValue(value.m_N);
                break;
            case FAttributeType::STRING_LIST:
                attribute =
                Aws::GameLift::Server::Model::AttributeValue::ConstructStringList();
                for (auto sl : value.m_SL) {
                    attribute.AddString(TCHAR_TO_UTF8(*sl));
                };
                break;
            case FAttributeType::STRING_DOUBLE_MAP:
                attribute =
                Aws::GameLift::Server::Model::AttributeValue::ConstructStringDoubleMap();
                for (auto sdm : value.m_SDM) {
                    attribute.AddStringAndDouble(TCHAR_TO_UTF8(*sdm.Key), sdm.Value);
                };
                break;
        }
        sdkPlayer.WithPlayerAttribute((TCHAR_TO_UTF8(*attributeEntry.Key)), attribute);
    }
    sdkRequest.AddPlayer(sdkPlayer);
}
auto outcome = Aws::GameLift::Server::StartMatchBackfill(sdkRequest);
if (outcome.IsSuccess()) {
    return FGameLiftStringOutcome(outcome.GetResult().GetTicketId());
}
else {
    return FGameLiftStringOutcome(FGameLiftError(outcome.GetError()));
}
#else
return FGameLiftStringOutcome("");
#endif

```

```
}
```

StopMatchBackfill()

Cancels an active match backfill request. For more information, see [FlexMatch backfill feature](#).

Syntax

```
FGameLiftGenericOutcome StopMatchBackfill (FStopMatchBackfillRequest  
&stopBackfillRequest);
```

Parameters

FStopMatchBackfillRequest

A StopMatchBackfillRequest object identifying the matchmaking ticket to cancel:

- The ticket ID assigned to the backfill request.
- The matchmaker the backfill request was sent to.
- The game session associated with the backfill request.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
FGameLiftGenericOutcome FGameLiftServerSDKModule::StopMatchBackfill(const  
    FStopMatchBackfillRequest& request)  
{  
    #if WITH_GAMELIFT  
        Aws::GameLift::Server::Model::StopMatchBackfillRequest sdkRequest;  
        sdkRequest.SetTicketId(TCHAR_TO_UTF8(*request.m_ticketId));  
        sdkRequest.SetGameSessionArn(TCHAR_TO_UTF8(*request.m_gameSessionArn));  
  
        sdkRequest.SetMatchmakingConfigurationArn(TCHAR_TO_UTF8(*request.m_matchmakingConfigurationArn));  
        auto outcome = Aws::GameLift::Server::StopMatchBackfill(sdkRequest);  
        if (outcome.IsSuccess()) {  
            return FGameLiftGenericOutcome(nullptr);  
        }  
        else {  
            return FGameLiftGenericOutcome(FGameLiftError(outcome.GetError()));  
        }  
    }  
}
```



```

}
#else
return FGameLiftGenericOutcome(nullptr);
#endif
}

```

GetComputeCertificate()

Retrieves the path to the TLS certificate used to encrypt the network connection between your Amazon GameLift Servers Anywhere compute resource and Amazon GameLift Servers. You can use the certificate path when you register your compute device to a Amazon GameLift Servers Anywhere fleet. For more information see, [RegisterCompute](#).

Syntax

```
FGameLiftGetComputeCertificateOutcome FGameLiftServerSDKModule::GetComputeCertificate()
```

Return value

Returns a `GetComputeCertificateResponse` object containing the following:

- **CertificatePath:** The path to the TLS certificate on your compute resource.
- **HostName:** The host name of your compute resource.

Example

```

FGameLiftGetComputeCertificateOutcome FGameLiftServerSDKModule::GetComputeCertificate()
{
    #if WITH_GAMELIFT
    auto outcome = Aws::GameLift::Server::GetComputeCertificate();
    if (outcome.IsSuccess()) {
        auto& outres = outcome.GetResult();
        FGameLiftGetComputeCertificateResult result;
        result.m_certificate_path = UTF8_TO_TCHAR(outres.GetCertificatePath());
        result.m_computeName = UTF8_TO_TCHAR(outres.GetComputeName());
        return FGameLiftGetComputeCertificateOutcome(result);
    }
    else {
        return FGameLiftGetComputeCertificateOutcome(FGameLiftError(outcome.GetError()));
    }
    #else

```

```

    return FGameLiftGetComputeCertificateOutcome(FGameLiftGetComputeCertificateResult());
#endif
}

```

GetFleetRoleCredentials()

Retrieves IAM role credentials that authorize Amazon GameLift Servers to interact with other AWS services. For more information, see [Connect your Amazon GameLift Servers hosted game server to other AWS resources](#).

Syntax

```

FGameLiftGetFleetRoleCredentialsOutcome
FGameLiftServerSDKModule::GetFleetRoleCredentials(const
FGameLiftGetFleetRoleCredentialsRequest &request)

```

Parameters

[FGameLiftGetFleetRoleCredentialsRequest](#)

Return value

Returns a [the section called “FGameLiftGetFleetRoleCredentialsOutcome”](#) object.

Example

```

FGameLiftGetFleetRoleCredentialsOutcome
FGameLiftServerSDKModule::GetFleetRoleCredentials(const
FGameLiftGetFleetRoleCredentialsRequest &request)
{
    #if WITH_GAMELIFT
    Aws::GameLift::Server::Model::GetFleetRoleCredentialsRequest sdkRequest;
    sdkRequest.SetRoleArn(TCHAR_TO_UTF8(*request.m_roleArn));
    sdkRequest.SetRoleSessionName(TCHAR_TO_UTF8(*request.m_roleSessionName));

    auto outcome = Aws::GameLift::Server::GetFleetRoleCredentials(sdkRequest);

    if (outcome.IsSuccess()) {
        auto& outres = outcome.GetResult();
        FGameLiftGetFleetRoleCredentialsResult result;
        result.m_assumedUserRoleArn = UTF8_TO_TCHAR(outres.GetAssumedUserRoleArn());
        result.m_assumedRoleId = UTF8_TO_TCHAR(outres.GetAssumedRoleId());
        result.m_accessKeyId = UTF8_TO_TCHAR(outres.GetAccessKeyId());
    }
    }

```

```

    result.m_secretAccessKey = UTF8_T0_TCHAR(outres.GetSecretAccessKey());
    result.m_sessionToken = UTF8_T0_TCHAR(outres.GetSessionToken());
    result.m_expiration = FDateTime::FromUnixTimestamp(outres.GetExpiration());
    return FGameLiftGetFleetRoleCredentialsOutcome(result);
}
else {
    return FGameLiftGetFleetRoleCredentialsOutcome(FGameLiftError(outcome.GetError()));
}
#else
return
FGameLiftGetFleetRoleCredentialsOutcome(FGameLiftGetFleetRoleCredentialsResult());
#endif
}

```

ListContainersNetworkInfo()

Retrieves network information for all containers running on the same instance, including each container's name, ID, local IP address, and container group type. Use this information to enable game server processes to discover and communicate with other containers that are running on the same instance.

This action is supported only on container fleets. When called from any other compute type, it returns an `UNSUPPORTED_COMPUTE_TYPE_EXCEPTION` error. Amazon GameLift Servers obtains the network information from a discovery server that runs locally on the instance.

Syntax

```

FGameLiftListContainersNetworkInfoOutcome
FGameLiftServerSDKModule::ListContainersNetworkInfo()

```

Return value

Returns a [the section called "FGameLiftListContainersNetworkInfoOutcome"](#) object.

Example

```

FGameLiftListContainersNetworkInfoOutcome
FGameLiftServerSDKModule::ListContainersNetworkInfo()
{
    #if WITH_GAMELIFT
        auto outcome = Aws::GameLift::Server::ListContainersNetworkInfo();
        if (outcome.IsSuccess()) {

```

```

    auto& outres = outcome.GetResult();
    FGameLiftListContainersNetworkInfoResult result;

    const int count = outres.GetContainersNetworkInfoCount();
    if (count > 0) {
        const auto* containersNetworkInfo = outres.GetContainersNetworkInfo();
        result.m_containersNetworkInfo.Reserve(count);

        for (int i = 0; i < count; ++i) {
            const auto& info = containersNetworkInfo[i];
            FContainerNetworkInfo& dst =
result.m_containersNetworkInfo.AddDefaulted_GetRef();
            dst.m_containerName = UTF8_T0_TCHAR(info.GetContainerName());
            dst.m_containerId   = UTF8_T0_TCHAR(info.GetContainerId());
            dst.m_ipAddress     = UTF8_T0_TCHAR(info.GetIpAddress());
            switch (info.GetContainerGroupType()) {
                case Aws::GameLift::Server::Model::ContainerGroupType::GAME_SERVER:
                    dst.m_containerGroupType = EContainerGroupType::GAME_SERVER;
                    break;
                case
Aws::GameLift::Server::Model::ContainerGroupType::PER_INSTANCE:
                    dst.m_containerGroupType = EContainerGroupType::PER_INSTANCE;
                    break;
                default:
                    dst.m_containerGroupType = EContainerGroupType::GAME_SERVER;
                    break;
            }
        }
    }

    return FGameLiftListContainersNetworkInfoOutcome(result);
}
else {
    return
FGameLiftListContainersNetworkInfoOutcome(FGameLiftError(outcome.GetError()));
}
#else
    return
FGameLiftListContainersNetworkInfoOutcome(FGameLiftListContainersNetworkInfoResult());
#endif
}

```

Destroy()

Frees the Amazon GameLift Servers game server SDK from memory. As a best practice, call this method after `ProcessEnding()` and before terminating the process. If you're using an Anywhere fleet and you're not terminating server processes after every game session, call `Destroy()` and then `InitSDK()` to reinitialize before notifying Amazon GameLift Servers that the process is ready to host a game session with `ProcessReady()`.

Syntax

```
FGameLiftGenericOutcome Destroy();
```

Parameters

There are no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
// First call ProcessEnding()
FGameLiftGenericOutcome processEndingOutcome = GameLiftSdkModule->ProcessEnding();

// Then call Destroy() to free the SDK from memory
FGameLiftGenericOutcome destroyOutcome = GameLiftSdkModule->Destroy();

// Exit the process with success or failure
if (processEndingOutcome.IsSuccess() && destroyOutcome.IsSuccess())
{
    UE_LOG(GameServerLog, Log, TEXT("Server process ending successfully"));
}
else {
    if (!processEndingOutcome.IsSuccess()) {
        const FGameLiftError& error = processEndingOutcome.GetError();
        UE_LOG(GameServerLog, Error, TEXT("ProcessEnding() failed. Error: %s"),
            error.m_errorMessage.IsEmpty() ? TEXT("Unknown error") :
            *error.m_errorMessage);
    }
    if (!destroyOutcome.IsSuccess()) {
```

```
const FGameLiftError& error = destroyOutcome.GetError();
UE_LOG(GameServerLog, Error, TEXT("Destroy() failed. Error: %s"),
error.m_errorMessage.IsEmpty() ? TEXT("Unknown error") :
*error.m_errorMessage);
}
```

Server SDK for Amazon GameLift Servers version 4 and earlier

This reference documents the server SDK for Amazon GameLift Servers, version 4.x and earlier. The server SDK provides core functionality that your game servers use to communicate with the Amazon GameLift Servers service. For example, your game server receives prompts from the service to start and stop game sessions and it provides regular game session status updates to the service. Integrate your game servers with the server SDK before you deploy them for hosting.

Use this server SDK reference to integrate your custom multiplayer game servers for hosting with Amazon GameLift Servers. For guidance about the integration process, see [Add Amazon GameLift Servers to your game server with the server SDK](#).

For SDK version 4.0.2, you can download it from the [official GitHub releases](#). The complete SDK package including GameLiftLocal.jar is available in the GameLift-CSharp-ServerSDK-4.0.2.zip artifact.

The latest major version of the server SDK for Amazon GameLift Servers is 5.x. The following hosting features require updates to version 5.x:

- Amazon GameLift Servers Anywhere
- Amazon GameLift Servers managed containers
- Amazon GameLift Servers plugin for Unreal Engine and Unity

Topics

- [C++ server SDK for Amazon GameLift Servers 4.x -- Actions](#)
- [C# server SDK for Amazon GameLift Servers 4.x -- Actions](#)
- [Server SDK \(Unreal\) for Amazon GameLift Servers -- Actions](#)

C++ server SDK for Amazon GameLift Servers 4.x -- Actions

Use the server SDK reference to integrate your multiplayer game for hosting with Amazon GameLift Servers. For guidance about the integration process, see [Add Amazon GameLift Servers to your game server with the server SDK](#).

Note

This reference is for an earlier version of the server SDK for Amazon GameLift Servers. For the latest version, see [C++ server SDK 5.x for Amazon GameLift Servers -- Actions](#).

C++ server SDK for Amazon GameLift Servers 4.x -- Data types

Use the server SDK reference to integrate your multiplayer game for hosting with Amazon GameLift Servers. For guidance about the integration process, see [Add Amazon GameLift Servers to your game server with the server SDK](#).

Note

This reference is for an earlier version of the server SDK for Amazon GameLift Servers. For the latest version, see [C++ server SDK 5.x for Amazon GameLift Servers -- Data types](#).

This API is defined in `GameLiftServerAPI.h`, `LogParameters.h`, and `ProcessParameters.h`.

[C++ server SDK for Amazon GameLift Servers 4.x -- Actions](#)

DescribePlayerSessionsRequest

This data type is used to specify which player session(s) to retrieve. You can use it as follows:

- Provide a `PlayerSessionId` to request a specific player session.
- Provide a `GameSessionId` to request all player sessions in the specified game session.
- Provide a `PlayerId` to request all player sessions for the specified player.

For large collections of player sessions, use the pagination parameters to retrieve results in sequential blocks.

Contents

GameSessionId

Unique game session identifier. Use this parameter to request all player sessions for the specified game session. Game session ID format is as follows: `arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>`. The value of `<ID string>` is either a custom ID string or (if one was specified when the game session was created) a generated string.

Type: String

Required: No

Limit

Maximum number of results to return. Use this parameter with *NextToken* to get results as a set of sequential pages. If a player session ID is specified, this parameter is ignored.

Type: Integer

Required: No

NextToken

Token indicating the start of the next sequential page of results. Use the token that is returned with a previous call to this action. To specify the start of the result set, do not specify a value. If a player session ID is specified, this parameter is ignored.

Type: String

Required: No

PlayerId

Unique identifier for a player. Player IDs are defined by the developer. See [Generate player IDs](#).

Type: String

Required: No

PlayerSessionId

Unique identifier for a player session.

Type: String

Required: No

PlayerSessionStatusFilter

Player session status to filter results on. Possible player session statuses include the following:

- **RESERVED** – The player session request has been received, but the player has not yet connected to the server process and/or been validated.
- **ACTIVE** – The player has been validated by the server process and is currently connected.
- **COMPLETED** – The player connection has been dropped.
- **TIMEDOUT** – A player session request was received, but the player did not connect and/or was not validated within the time-out limit (60 seconds).

Type: String

Required: No

LogParameters

This data type is used to identify which files generated during a game session that you want Amazon GameLift Servers to upload and store once the game session ends. This information is communicated to the Amazon GameLift Servers service in a [ProcessReady\(\)](#) call.

Contents

logPaths

Directory paths to game server log files that you want Amazon GameLift Servers to store for future access. These files are generated during each game session. File paths and names are defined in your game server and stored in the root game build directory. The log paths must be absolute. For example, if your game build stores game session logs in a path like `MyGame\sessionlogs\`, then the log path would be `c:\game\MyGame\sessionLogs` (on a Windows instance) or `/local/game/MyGame/sessionLogs` (on a Linux instance).

Type: `std::vector<std::string>`

Required: No

ProcessParameters

This data type contains the set of parameters sent to the Amazon GameLift Servers service in a [ProcessReady\(\)](#) call.

Contents

port

Port number the server process listens on for new player connections. The value must fall into the port range configured for any fleet deploying this game server build. This port number is included in game session and player session objects, which game sessions use when connecting to a server process.

Type: Integer

Required: Yes

logParameters

Object with a list of directory paths to game session log files.

Type: `Aws::GameLift::Server::`[LogParameters](#)

Required: No

onStartGameSession

Name of callback function that the Amazon GameLift Servers service invokes to activate a new game session. Amazon GameLift Servers calls this function in response to the client request [CreateGameSession](#). The callback function passes a [GameSession](#) object (defined in the *Amazon GameLift Servers Service API Reference*).

Type: `const std::function<void(Aws::GameLift::Model::GameSession)>`
`onStartGameSession`

Required: Yes

onProcessTerminate

Name of callback function that the Amazon GameLift Servers service invokes to force the server process to shut down. After calling this function, Amazon GameLift Servers waits five minutes

for the server process to shut down and respond with a [ProcessEnding\(\)](#) call. If no response is received, it shuts down the server process.

Type: `std::function<void()> onProcessTerminate`

Required: No

onHealthCheck

Name of callback function that the Amazon GameLift Servers service invokes to request a health status report from the server process. Amazon GameLift Servers calls this function every 60 seconds. After calling this function Amazon GameLift Servers waits 60 seconds for a response, and if none is received, records the server process as unhealthy.

Type: `std::function<bool()> onHealthCheck`

Required: No

onUpdateGameSession

Name of callback function that the Amazon GameLift Servers service invokes to pass an updated game session object to the server process. Amazon GameLift Servers calls this function when a [match backfill](#) request has been processed in order to provide updated matchmaker data. It passes a [GameSession](#) object, a status update (`updateReason`), and the match backfill ticket ID.

Type: `std::function<void(Aws::GameLift::Server::Model::UpdateGameSession)> onUpdateGameSession`

Required: No

StartMatchBackfillRequest

This data type is used to send a matchmaking backfill request. The information is communicated to the Amazon GameLift Servers service in a [StartMatchBackfill\(\)](#) call.

Contents

GameSessionArn

Unique game session identifier. The API action [GetGameSessionId\(\)](#) returns the identifier in ARN format.

Type: String

Required: Yes

MatchmakingConfigurationArn

Unique identifier, in the form of an ARN, for the matchmaker to use for this request. To find the matchmaker that was used to create the original game session, look in the game session object, in the matchmaker data property. Learn more about matchmaker data in [Word with matchmaker data](#).

Type: String

Required: Yes

Players

A set of data representing all players who are currently in the game session. The matchmaker uses this information to search for new players who are good matches for the current players. See the *Amazon GameLift Servers API Reference Guide* for a description of the Player object format. To find player attributes, IDs, and team assignments, look in the game session object, in the matchmaker data property. If latency is used by the matchmaker, gather updated latency for the current region and include it in each player's data.

Type: std:vector[<player>](#)

Required: Yes

TicketId

Unique identifier for a matchmaking or match backfill request ticket. If no value is provided here, Amazon GameLift Servers will generate one in the form of a UUID. Use this identifier to track the match backfill ticket status or cancel the request if needed.

Type: String

Required: No

StopMatchBackfillRequest

This data type is used to cancel a matchmaking backfill request. The information is communicated to the Amazon GameLift Servers service in a [StopMatchBackfill\(\)](#) call.

Contents

GameSessionArn

Unique game session identifier associated with the request being canceled.

Type: String

Required: Yes

MatchmakingConfigurationArn

Unique identifier of the matchmaker this request was sent to.

Type: String

Required: Yes

TicketId

Unique identifier of the backfill request ticket to be canceled.

Type: String

Required: Yes

[C++ server SDK for Amazon GameLift Servers 4.x -- Data types](#)

Topics

- [AcceptPlayerSession\(\)](#)
- [ActivateGameSession\(\)](#)
- [DescribePlayerSessions\(\)](#)
- [GetGameSessionId\(\)](#)
- [GetInstanceCertificate\(\)](#)
- [GetSdkVersion\(\)](#)
- [GetTerminationTime\(\)](#)
- [InitSDK\(\)](#)
- [ProcessEnding\(\)](#)
- [ProcessReady\(\)](#)
- [ProcessReadyAsync\(\)](#)

- [RemovePlayerSession\(\)](#)
- [StartMatchBackfill\(\)](#)
- [StopMatchBackfill\(\)](#)
- [TerminateGameSession\(\)](#)
- [UpdatePlayerSessionCreationPolicy\(\)](#)
- [Destroy\(\)](#)

AcceptPlayerSession()

Notifies the Amazon GameLift Servers service that a player with the specified player session ID has connected to the server process and needs validation. Amazon GameLift Servers verifies that the player session ID is valid—that is, that the player ID has reserved a player slot in the game session. Once validated, Amazon GameLift Servers changes the status of the player slot from RESERVED to ACTIVE.

Syntax

```
GenericOutcome AcceptPlayerSession(const std::string& playerSessionId);
```

Parameters

playerSessionId

Unique ID issued by the Amazon GameLift Servers service in response to a call to the AWS SDK Amazon GameLift Servers API action [CreatePlayerSession](#). The game client references this ID when connecting to the server process.

Type: std::string

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example illustrates a function for handling a connection request, including validating and rejecting invalid player session IDs.

```
void ReceiveConnectingPlayerSessionID (Connection& connection, const std::string&
playerSessionId){
    Aws::GameLift::GenericOutcome connectOutcome =
        Aws::GameLift::Server::AcceptPlayerSession(playerSessionId);
    if(connectOutcome.IsSuccess())
    {
        connectionToSessionMap.emplace(connection, playerSessionId);
        connection.Accept();
    }
    else
    {
        connection.Reject(connectOutcome.GetError().GetMessage());
    }
}
```

ActivateGameSession()

Notifies the Amazon GameLift Servers service that the server process has started a game session and is now ready to receive player connections. This action should be called as part of the `onStartGameSession()` callback function, after all game session initialization has been completed.

Syntax

```
GenericOutcome ActivateGameSession();
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example shows `ActivateGameSession()` being called as part of the `onStartGameSession()` callback function.

```
void onStartGameSession(Aws::GameLift::Model::GameSession myGameSession)
{
    // game-specific tasks when starting a new game session, such as loading map
```

```
GenericOutcome outcome = Aws::GameLift::Server::ActivateGameSession();  
}
```

DescribePlayerSessions()

Retrieves player session data, including settings, session metadata, and player data. Use this action to get information for a single player session, for all player sessions in a game session, or for all player sessions associated with a single player ID.

Syntax

```
DescribePlayerSessionsOutcome DescribePlayerSessions (  
    const Aws::GameLift::Server::Model::DescribePlayerSessionsRequest  
    &describePlayerSessionsRequest);
```

Parameters

describePlayerSessionsRequest

A [DescribePlayerSessionsRequest](#) object describing which player sessions to retrieve.

Required: Yes

Return value

If successful, returns a `DescribePlayerSessionsOutcome` object containing a set of player session objects that fit the request parameters. Player session objects have a structure identical to the AWS SDK Amazon GameLift Servers API [PlayerSession](#) data type.

Example

This example illustrates a request for all player sessions actively connected to a specified game session. By omitting `NextToken` and setting the `Limit` value to 10, Amazon GameLift Servers returns the first 10 player sessions records matching the request.

```
// Set request parameters  
Aws::GameLift::Server::Model::DescribePlayerSessionsRequest request;  
request.SetPlayerSessionStatusFilter(Aws::GameLift::Server::Model::PlayerSessionStatusMapper::G  
request.SetLimit(10);  
request.SetGameSessionId("the game session ID");    // can use GetGameSessionId()
```



```
// Call DescribePlayerSessions
Aws::GameLift::DescribePlayerSessionsOutcome playerSessionsOutcome =
    Aws::GameLift::Server::DescribePlayerSessions(request);
```

GetGameSessionId()

Retrieves a unique identifier for the game session currently being hosted by the server process, if the server process is active. The identifier is returned in ARN format: `arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>`.

For idle process that are not yet activated with a game session, the call returns `Success=True` and `GameSessionId=""` (an empty string).

Syntax

```
AwsStringOutcome GetGameSessionId();
```

Parameters

This action has no parameters.

Return value

If successful, returns the game session ID as an `AwsStringOutcome` object. If not successful, returns an error message.

Example

```
Aws::GameLift::AwsStringOutcome sessionIdOutcome =
    Aws::GameLift::Server::GetGameSessionId();
```

GetInstanceCertificate()

Retrieves the file location of a pem-encoded TLS certificate that is associated with the fleet and its instances. AWS Certificate Manager generates this certificate when you create a new fleet with the certificate configuration set to `GENERATED`. Use this certificate to establish a secure connection with a game client and to encrypt client/server communication.

Syntax

```
GetInstanceCertificateOutcome GetInstanceCertificate();
```

Parameters

This action has no parameters.

Return value

If successful, returns a `GetInstanceCertificateOutcome` object containing the location of the fleet's TLS certificate file and certificate chain, which are stored on the instance. A root certificate file, extracted from the certificate chain, is also stored on the instance. If not successful, returns an error message.

For more information about the certificate and certificate chain data, see [GetCertificate Response Elements](#) in the AWS Certificate Manager API Reference.

Example

```
Aws::GameLift::GetInstanceCertificateOutcome certificateOutcome =  
    Aws::GameLift::Server::GetInstanceCertificate();
```

GetSdkVersion()

Returns the current version number of the SDK in use.

Syntax

```
AwsStringOutcome GetSdkVersion();
```

Parameters

This action has no parameters.

Return value

If successful, returns the current SDK version as an `AwsStringOutcome` object. The returned string includes the version number only (ex. "3.1.5"). If not successful, returns an error message.

Example

```
Aws::GameLift::AwsStringOutcome SdkVersionOutcome =
```

```
Aws::GameLift::Server::GetSdkVersion();
```

GetTerminationTime()

Returns the time that a server process is scheduled to be shut down, if a termination time is available. A server process takes this action after receiving an `onProcessTerminate()` callback from the Amazon GameLift Servers service. Amazon GameLift Servers may call `onProcessTerminate()` for the following reasons: (1) when the server process has reported poor health or has not responded to Amazon GameLift Servers, (2) when terminating the instance during a scale-down event, or (3) when an instance is being terminated due to a [Spot interruption](#).

If the process has received an `onProcessTerminate()` callback, the value returned is the estimated termination time. If the process has not received an `onProcessTerminate()` callback, an error message is returned. Learn more about [shutting down a server process](#).

Syntax

```
AwsLongOutcome GetTerminationTime();
```

Parameters

This action has no parameters.

Return value

If successful, returns the termination time as an `AwsLongOutcome` object. The value is the termination time, expressed in elapsed ticks since 0001 00:00:00. For example, the date time value 2020-09-13 12:26:40 -000Z is equal to 637355968000000000 ticks. If no termination time is available, returns an error message.

Example

```
Aws::GameLift::AwsLongOutcome TermTimeOutcome =  
    Aws::GameLift::Server::GetTerminationTime();
```

InitSDK()

Initializes the Amazon GameLift Servers SDK. This method should be called on launch, before any other Amazon GameLift Servers-related initialization occurs.

Syntax

```
InitSDKOutcome InitSDK();
```

Parameters

This action has no parameters.

Return value

If successful, returns an `InitSdkOutcome` object indicating that the server process is ready to call [ProcessReady\(\)](#).

Example

```
Aws::GameLift::Server::InitSDKOutcome initOutcome =  
    Aws::GameLift::Server::InitSDK();
```

ProcessEnding()

Notifies the Amazon GameLift Servers service that the server process is shutting down. This method should be called after all other cleanup tasks, including shutting down all active game sessions. This method should exit with an exit code of 0; a non-zero exit code results in an event message that the process did not exit cleanly.

Once the method exits with a code of 0, you can terminate the process with a successful exit code. You can also exit the process with an error code. If you exit with an error code, the fleet event will indicate the process terminated abnormally (`SERVER_PROCESS_TERMINATED_UNHEALTHY`).

Syntax

```
GenericOutcome ProcessEnding();
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
Aws::GameLift::GenericOutcome outcome = Aws::GameLift::Server::ProcessEnding();
if (outcome.Success)
    exit(0); // exit with success
// otherwise, exit with error code
exit(errorCode);
```

ProcessReady()

Notifies the Amazon GameLift Servers service that the server process is ready to host game sessions. Call this method after successfully invoking [InitSDK\(\)](#) and completing setup tasks that are required before the server process can host a game session. This method should be called only once per process.

This call is synchronous. To make an asynchronous call, use [ProcessReadyAsync\(\)](#). See [Initialize the server process](#) for more details.

Syntax

```
GenericOutcome ProcessReady(
    const Aws::GameLift::Server::ProcessParameters &processParameters);
```

Parameters

processParameters

A [ProcessParameters](#) object communicating the following information about the server process:

- Names of callback methods, implemented in the game server code, that the Amazon GameLift Servers service invokes to communicate with the server process.
- Port number that the server process is listening on.
- Path to any game session-specific files that you want Amazon GameLift Servers to capture and store.

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example illustrates both the [ProcessReady\(\)](#) call and callback function implementations.

```
// Set parameters and call ProcessReady
std::string serverLog("serverOut.log");           // Example of a log file written by the
game server
std::vector<std::string> logPaths;
logPaths.push_back(serverLog);

int listenPort = 9339;

Aws::GameLift::Server::ProcessParameters processReadyParameter =
    Aws::GameLift::Server::ProcessParameters(
        std::bind(&Server::onStartGameSession, this, std::placeholders::_1),
        std::bind(&Server::onProcessTerminate, this),
        std::bind(&Server::OnHealthCheck, this),
        std::bind(&Server::OnUpdateGameSession, this),
        listenPort,
        Aws::GameLift::Server::LogParameters(logPaths));

Aws::GameLift::GenericOutcome outcome =
    Aws::GameLift::Server::ProcessReady(processReadyParameter);

// Implement callback functions
void Server::onStartGameSession(Aws::GameLift::Model::GameSession myGameSession)
{
    // game-specific tasks when starting a new game session, such as loading map
    GenericOutcome outcome =
        Aws::GameLift::Server::ActivateGameSession (maxPlayers);
}

void Server::onProcessTerminate()
{
    // game-specific tasks required to gracefully shut down a game session,
    // such as notifying players, preserving game state data, and other cleanup
    GenericOutcome outcome = Aws::GameLift::Server::ProcessEnding();
}

bool Server::onHealthCheck()
{
    bool health;
    // complete health evaluation within 60 seconds and set health
    return health;
}
```

```
}
```

ProcessReadyAsync()

Notifies the Amazon GameLift Servers service that the server process is ready to host game sessions. This method should be called once the server process is ready to host a game session. The parameters specify the names of callback functions for Amazon GameLift Servers to call in certain circumstances. Game server code must implement these functions.

This call is asynchronous. To make a synchronous call, use [ProcessReady\(\)](#). See [Initialize the server process](#) for more details.

Syntax

```
GenericOutcomeCallable ProcessReadyAsync(  
    const Aws::GameLift::Server::ProcessParameters &processParameters);
```

Parameters

processParameters

A [ProcessParameters](#) object communicating the following information about the server process:

- Names of callback methods, implemented in the game server code, that the Amazon GameLift Servers service invokes to communicate with the server process.
- Port number that the server process is listening on.
- Path to any game session-specific files that you want Amazon GameLift Servers to capture and store.

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
// Set parameters and call ProcessReady  
std::string serverLog("serverOut.log");           // This is an example of a log file  
written by the game server
```

```

std::vector<std::string> logPaths;
logPaths.push_back(serverLog);

int listenPort = 9339;

Aws::GameLift::Server::ProcessParameters processReadyParameter =
    Aws::GameLift::Server::ProcessParameters(
        std::bind(&Server::onStartGameSession, this, std::placeholders::_1),
        std::bind(&Server::onProcessTerminate, this),
        std::bind(&Server::OnHealthCheck, this),
        std::bind(&Server::OnUpdateGameSession, this),
        listenPort,
        Aws::GameLift::Server::LogParameters(logPaths));

Aws::GameLift::GenericOutcomeCallable outcome =
    Aws::GameLift::Server::ProcessReadyAsync(processReadyParameter);

// Implement callback functions
void onStartGameSession(Aws::GameLift::Model::GameSession myGameSession)
{
    // game-specific tasks when starting a new game session, such as loading map
    GenericOutcome outcome = Aws::GameLift::Server::ActivateGameSession (maxPlayers);
}

void onProcessTerminate()
{
    // game-specific tasks required to gracefully shut down a game session,
    // such as notifying players, preserving game state data, and other cleanup
    GenericOutcome outcome = Aws::GameLift::Server::ProcessEnding();
}

bool onHealthCheck()
{
    // perform health evaluation and complete within 60 seconds
    return health;
}

```

RemovePlayerSession()

Notifies the Amazon GameLift Servers service that a player with the specified player session ID has disconnected from the server process. In response, Amazon GameLift Servers changes the player slot to available, which allows it to be assigned to a new player.

Syntax

```
GenericOutcome RemovePlayerSession(  
    const std::string& playerId);
```

Parameters

playerSessionId

Unique ID issued by the Amazon GameLift Servers service in response to a call to the AWS SDK Amazon GameLift Servers API action [CreatePlayerSession](#). The game client references this ID when connecting to the server process.

Type: `std::string`

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
Aws::GameLift::GenericOutcome disconnectOutcome =  
    Aws::GameLift::Server::RemovePlayerSession(playerSessionId);
```

StartMatchBackfill()

Sends a request to find new players for open slots in a game session created with FlexMatch. See also the AWS SDK action [StartMatchBackfill\(\)](#). With this action, match backfill requests can be initiated by a game server process that is hosting the game session. Learn more about the [FlexMatch backfill feature](#).

This action is asynchronous. If new players are successfully matched, the Amazon GameLift Servers service delivers updated matchmaker data by invoking the callback function `OnUpdateGameSession()`.

A server process can have only one active match backfill request at a time. To send a new request, first call [StopMatchBackfill\(\)](#) to cancel the original request.

Syntax

```
StartMatchBackfillOutcome StartMatchBackfill (  
    const Aws::GameLift::Server::Model::StartMatchBackfillRequest  
    &startBackfillRequest);
```

Parameters

StartMatchBackfillRequest

A [StartMatchBackfillRequest](#) object that communicates the following information:

- A ticket ID to assign to the backfill request. This information is optional; if no ID is provided, Amazon GameLift Servers will autogenerate one.
- The matchmaker to send the request to. The full configuration ARN is required. This value can be acquired from the game session's matchmaker data.
- The ID of the game session that is being backfilled.
- Available matchmaking data for the game session's current players.

Required: Yes

Return value

Returns a `StartMatchBackfillOutcome` object with the match backfill ticket or failure with an error message. Ticket status can be tracked using the AWS SDK action [DescribeMatchmaking\(\)](#).

Example

```
// Build a backfill request  
std::vector<Player> players;  
Aws::GameLift::Server::Model::StartMatchBackfillRequest startBackfillRequest;  
startBackfillRequest.SetTicketId("a ticket ID");  
    //optional, autogenerated if not provided  
startBackfillRequest.SetMatchmakingConfigurationArn("the matchmaker configuration  
    ARN"); //from the game session matchmaker data  
startBackfillRequest.SetGameSessionArn("the game session ARN");  
    // can use GetGameSessionId()  
startBackfillRequest.SetPlayers(players);  
    //from the game session matchmaker data  
  
// Send backfill request
```

```
Aws::GameLift::StartMatchBackfillOutcome backfillOutcome =  
    Aws::GameLift::Server::StartMatchBackfill(startBackfillRequest);  
  
// Implement callback function for backfill  
void Server::OnUpdateGameSession(Aws::GameLift::Server::Model::GameSession gameSession,  
    Aws::GameLift::Server::Model::UpdateReason updateReason, std::string backfillTicketId)  
{  
    // handle status messages  
    // perform game-specific tasks to prep for newly matched players  
}
```

StopMatchBackfill()

Cancels an active match backfill request that was created with [StartMatchBackfill\(\)](#). See also the AWS SDK action [StopMatchmaking\(\)](#). Learn more about the [FlexMatch backfill feature](#).

Syntax

```
GenericOutcome StopMatchBackfill (  
    const Aws::GameLift::Server::Model::StopMatchBackfillRequest &stopBackfillRequest);
```

Parameters

StopMatchBackfillRequest

A [StopMatchBackfillRequest](#) object identifying the matchmaking ticket to cancel:

- ticket ID assigned to the backfill request being canceled
- matchmaker the backfill request was sent to
- game session associated with the backfill request

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
// Set backfill stop request parameters  
  
Aws::GameLift::Server::Model::StopMatchBackfillRequest stopBackfillRequest;
```

```
stopBackfillRequest.SetTicketId("the ticket ID");
stopBackfillRequest.SetGameSessionArn("the game session ARN");
// can use GetGameSessionId()
stopBackfillRequest.SetMatchmakingConfigurationArn("the matchmaker configuration ARN");
// from the game session matchmaker data

Aws::GameLift::GenericOutcome stopBackfillOutcome =
    Aws::GameLift::Server::StopMatchBackfillRequest(stopBackfillRequest);
```

TerminateGameSession()

This method is deprecated with version 4.0.1. Instead, the server process should call [ProcessEnding\(\)](#) after a game session has ended.

Notifies the Amazon GameLift Servers service that the server process has ended the current game session. This action is called when the server process will remain active and ready to host a new game session. It should be called only after your game session termination procedure is complete, because it signals to Amazon GameLift Servers that the server process is immediately available to host a new game session.

This action is not called if the server process will be shut down after the game session stops. Instead, call [ProcessEnding\(\)](#) to signal that both the game session and the server process are ending.

Syntax

```
GenericOutcome TerminateGameSession();
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

UpdatePlayerSessionCreationPolicy()

Updates the current game session's ability to accept new player sessions. A game session can be set to either accept or deny all new player sessions. See also the AWS SDK action [UpdateGameSession\(\)](#).

Syntax

```
GenericOutcome UpdatePlayerSessionCreationPolicy(  
    Aws::GameLift::Model::PlayerSessionCreationPolicy newPlayerSessionPolicy);
```

Parameters

newPlayerSessionPolicy

String value indicating whether the game session accepts new players.

Type: `Aws::GameLift::Model::PlayerSessionCreationPolicy` enum. Valid values include:

- **ACCEPT_ALL** – Accept all new player sessions.
- **DENY_ALL** – Deny all new player sessions.

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example sets the current game session's join policy to accept all players.

```
Aws::GameLift::GenericOutcome outcome =  
    Aws::GameLift::Server::UpdatePlayerSessionCreationPolicy(Aws::GameLift::Model::PlayerSessionCr
```

Destroy()

Cleans up memory allocated by `initSDK()` during game server initialization. Use this method after you end a game server process to avoid wasting server memory.

Syntax

```
GenericOutcome Aws::GameLift::Server::Destroy();
```

Parameters

There are no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example cleans up the memory allocated by initSDK after a game server process has ended.

```
if (Aws::GameLift::Server::ProcessEnding().IsSuccess()) {  
    Aws::GameLift::Server::Destroy();  
    exit(0);  
}
```

C# server SDK for Amazon GameLift Servers 4.x -- Actions

Use the server SDK reference to integrate your multiplayer game for hosting with Amazon GameLift Servers. For guidance about the integration process, see [Add Amazon GameLift Servers to your game server with the server SDK](#).

Note

This reference is for an earlier version of the server SDK for Amazon GameLift Servers. For the latest version, see [C# server SDK 5.x for Amazon GameLift Servers -- Actions](#).

C# server SDK for Amazon GameLift Servers 4.x -- Data types

Use the server SDK reference to integrate your multiplayer game for hosting with Amazon GameLift Servers. For guidance about the integration process, see [Add Amazon GameLift Servers to your game server with the server SDK](#).

Note

This reference is for an earlier version of the server SDK for Amazon GameLift Servers. For the latest version, see [C# server SDK 5.x for Amazon GameLift Servers -- Data types](#).

[C# server SDK for Amazon GameLift Servers 4.x -- Actions](#)

Topics

- [LogParameters](#)
- [DescribePlayerSessionsRequest](#)
- [ProcessParameters](#)
- [StopMatchBackfillRequest](#)

LogParameters

This data type is used to identify which files generated during a game session that you want Amazon GameLift Servers to upload and store once the game session ends. This information is communicated to the Amazon GameLift Servers service in a [ProcessReady\(\)](#) call.

Contents

logPaths

List of directory paths to game server log files you want Amazon GameLift Servers to store for future access. These files are generated by a server process during each game session; file paths and names are defined in your game server and stored in the root game build directory. The log paths must be absolute. For example, if your game build stores game session logs in a path like `MyGame\sessionlogs\`, then the log path would be `c:\game\MyGame\sessionLogs` (on a Windows instance) or `/local/game/MyGame/sessionLogs` (on a Linux instance).

Type: List<String>

Required: No

DescribePlayerSessionsRequest

This data type is used to specify which player session(s) to retrieve. It can be used in several ways: (1) provide a `PlayerSessionId` to request a specific player session; (2) provide a `GameSessionId` to request all player sessions in the specified game session; or (3) provide a `PlayerId` to request all player sessions for the specified player. For large collections of player sessions, use the pagination parameters to retrieve results as sequential pages.

Contents

GameSessionId

Unique game session identifier. Use this parameter to request all player sessions for the specified game session. Game session ID format is as follows:

`arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>`. The value of `<ID string>` is either a custom ID string (if one was specified when the game session was created) a generated string.

Type: String

Required: No

Limit

Maximum number of results to return. Use this parameter with *NextToken* to get results as a set of sequential pages. If a player session ID is specified, this parameter is ignored.

Type: Integer

Required: No

NextToken

Token indicating the start of the next sequential page of results. Use the token that is returned with a previous call to this action. To specify the start of the result set, do not specify a value. If a player session ID is specified, this parameter is ignored.

Type: String

Required: No

PlayerId

Unique identifier for a player. Player IDs are defined by the developer. See [Generate player IDs](#).

Type: String

Required: No

PlayerSessionId

Unique identifier for a player session.

Type: String

Required: No

PlayerSessionStatusFilter

Player session status to filter results on. Possible player session statuses include the following:

- **RESERVED** – The player session request has been received, but the player has not yet connected to the server process and/or been validated.
- **ACTIVE** – The player has been validated by the server process and is currently connected.
- **COMPLETED** – The player connection has been dropped.
- **TIMEDOUT** – A player session request was received, but the player did not connect and/or was not validated within the time-out limit (60 seconds).

Type: String

Required: No

ProcessParameters

This data type contains the set of parameters sent to the Amazon GameLift Servers service in a [ProcessReady\(\)](#) call.

Contents

port

Port number the server process will listen on for new player connections. The value must fall into the port range configured for any fleet deploying this game server build. This port number is included in game session and player session objects, which game sessions use when connecting to a server process.

Type: Integer

Required: Yes

logParameters

Object with a list of directory paths to game session log files.

Type: `Aws::GameLift::Server::LogParameters`

Required: Yes

onStartGameSession

Name of callback function that the Amazon GameLift Servers service invokes to activate a new game session. Amazon GameLift Servers calls this function in response to the client request [CreateGameSession](#). The callback function takes a [GameSession](#) object (defined in the *Amazon GameLift Servers Service API Reference*).

Type: void OnStartGameSessionDelegate(GameSession gameSession)

Required: Yes

onProcessTerminate

Name of callback function that the Amazon GameLift Servers service invokes to force the server process to shut down. After calling this function, Amazon GameLift Servers waits five minutes for the server process to shut down and respond with a [ProcessEnding\(\)](#) call before it shuts down the server process.

Type: void OnProcessTerminateDelegate()

Required: Yes

onHealthCheck

Name of callback function that the Amazon GameLift Servers service invokes to request a health status report from the server process. Amazon GameLift Servers calls this function every 60 seconds. After calling this function Amazon GameLift Servers waits 60 seconds for a response, and if none is received, records the server process as unhealthy.

Type: bool OnHealthCheckDelegate()

Required: Yes

onUpdateGameSession

Name of callback function that the Amazon GameLift Servers service invokes to pass an updated game session object to the server process. Amazon GameLift Servers calls this function when a [match backfill](#) request has been processed in order to provide updated matchmaker data. It passes a [GameSession](#) object, a status update (updateReason), and the match backfill ticket ID.

Type: void OnUpdateGameSessionDelegate (UpdateGameSession
updateGameSession)

Required: No

StartMatchBackfillRequest

This data type is used to send a matchmaking backfill request. The information is communicated to the Amazon GameLift Servers service in a [StartMatchBackfill\(\)](#) call.

Contents

GameSessionArn

Unique game session identifier. The SDK method [GetGameSessionId\(\)](#) returns the identifier in ARN format.

Type: String

Required: Yes

MatchmakingConfigurationArn

Unique identifier, in the form of an ARN, for the matchmaker to use for this request. To find the matchmaker that was used to create the original game session, look in the game session object, in the matchmaker data property. Learn more about matchmaker data in [Work with matchmaker data](#).

Type: String

Required: Yes

Players

A set of data representing all players who are currently in the game session. The matchmaker uses this information to search for new players who are good matches for the current players. See the *Amazon GameLift Servers API Reference Guide* for a description of the Player object format. To find player attributes, IDs, and team assignments, look in the game session object, in the matchmaker data property. If latency is used by the matchmaker, gather updated latency for the current region and include it in each player's data.

Type: [Player](#)[]

Required: Yes

TicketId

Unique identifier for a matchmaking or match backfill request ticket. If no value is provided here, Amazon GameLift Servers will generate one in the form of a UUID. Use this identifier to track the match backfill ticket status or cancel the request if needed.

Type: String

Required: No

StopMatchBackfillRequest

This data type is used to cancel a matchmaking backfill request. The information is communicated to the Amazon GameLift Servers service in a [StopMatchBackfill\(\)](#) call.

Contents

GameSessionArn

Unique game session identifier associated with the request being canceled.

Type: String

Required: Yes

MatchmakingConfigurationArn

Unique identifier of the matchmaker this request was sent to.

Type: String

Required: Yes

TicketId

Unique identifier of the backfill request ticket to be canceled.

Type: String

Required: Yes

[C# server SDK for Amazon GameLift Servers 4.x -- Data types](#)

Topics

- [AcceptPlayerSession\(\)](#)
- [ActivateGameSession\(\)](#)
- [DescribePlayerSessions\(\)](#)
- [GetGameSessionId\(\)](#)
- [GetInstanceCertificate\(\)](#)
- [GetSdkVersion\(\)](#)
- [GetTerminationTime\(\)](#)

- [InitSDK\(\)](#)
- [ProcessEnding\(\)](#)
- [ProcessReady\(\)](#)
- [RemovePlayerSession\(\)](#)
- [StartMatchBackfill\(\)](#)
- [StopMatchBackfill\(\)](#)
- [TerminateGameSession\(\)](#)
- [UpdatePlayerSessionCreationPolicy\(\)](#)

AcceptPlayerSession()

Notifies the Amazon GameLift Servers service that a player with the specified player session ID has connected to the server process and needs validation. Amazon GameLift Servers verifies that the player session ID is valid—that is, that the player ID has reserved a player slot in the game session. Once validated, Amazon GameLift Servers changes the status of the player slot from RESERVED to ACTIVE.

Syntax

```
GenericOutcome AcceptPlayerSession(String playerId)
```

Parameters

playerSessionId

Unique ID issued by Amazon GameLift Servers when a new player session is created. A player session ID is specified in a `PlayerSession` object, which is returned in response to a client call to the *GameLift API* actions [StartGameSessionPlacement](#), [CreateGameSession](#), [DescribeGameSessionPlacement](#), or [DescribePlayerSessions](#).

Type: String

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example illustrates a function for handling a connection request, including validating and rejecting invalid player session IDs.

```
void ReceiveConnectingPlayerSessionID (Connection connection, String playerId){
    var acceptPlayerSessionOutcome =
    GameLiftServerAPI.AcceptPlayerSession(playerSessionId);
    if(acceptPlayerSessionOutcome.Success)
    {
        connectionToSessionMap.emplace(connection, playerId);
        connection.Accept();
    }
    else
    {
        connection.Reject(acceptPlayerSessionOutcome.Error.ErrorMessage);    }
}
```

ActivateGameSession()

Notifies the Amazon GameLift Servers service that the server process has activated a game session and is now ready to receive player connections. This action should be called as part of the `onStartGameSession()` callback function, after all game session initialization has been completed.

Syntax

```
GenericOutcome ActivateGameSession()
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example shows `ActivateGameSession()` being called as part of the `onStartGameSession()` delegate function.

```
void OnStartGameSession(GameSession gameSession)
```

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

DescribePlayerSessions()

Retrieves player session data, including settings, session metadata, and player data. Use this action to get information for a single player session, for all player sessions in a game session, or for all player sessions associated with a single player ID.

Syntax

```
DescribePlayerSessionsOutcome DescribePlayerSessions(DescribePlayerSessionsRequest  
    describePlayerSessionsRequest)
```

Parameters

describePlayerSessionsRequest

A [DescribePlayerSessionsRequest](#) object describing which player sessions to retrieve.

Required: Yes

Return value

If successful, returns a `DescribePlayerSessionsOutcome` object containing a set of player session objects that fit the request parameters. Player session objects have a structure identical to the AWS SDK Amazon GameLift Servers API [PlayerSession](#) data type.

Example

This example illustrates a request for all player sessions actively connected to a specified game session. By omitting *NextToken* and setting the *Limit* value to 10, Amazon GameLift Servers will return the first 10 player sessions records matching the request.

```
// Set request parameters  
var describePlayerSessionsRequest = new  
    Aws.GameLift.Server.Model.DescribePlayerSessionsRequest()
```

```
{
    GameSessionId = GameLiftServerAPI.GetGameSessionId().Result,    //gets the ID for
    the current game session
    Limit = 10,
    PlayerSessionStatusFilter =
    PlayerSessionStatusMapper.GetNameForPlayerSessionStatus(PlayerSessionStatus.ACTIVE)
};
// Call DescribePlayerSessions
Aws::GameLift::DescribePlayerSessionsOutcome playerSessionsOutcome =
    Aws::GameLift::Server::Model::DescribePlayerSessions(describePlayerSessionRequest);
```

GetGameSessionId()

Retrieves the ID of the game session currently being hosted by the server process, if the server process is active.

For idle process that are not yet activated with a game session, the call returns `Success=True` and `GameSessionId=""` (an empty string).

Syntax

```
AwsStringOutcome GetGameSessionId()
```

Parameters

This action has no parameters.

Return value

If successful, returns the game session ID as an `AwsStringOutcome` object. If not successful, returns an error message.

Example

```
var getGameSessionIdOutcome = GameLiftServerAPI.GetGameSessionId();
```

GetInstanceCertificate()

Retrieves the file location of a pem-encoded TLS certificate that is associated with the fleet and its instances. AWS Certificate Manager generates this certificate when you create a new fleet with the certificate configuration set to `GENERATED`. Use this certificate to establish a secure connection with a game client and to encrypt client/server communication.

Syntax

```
GetInstanceCertificateOutcome GetInstanceCertificate();
```

Parameters

This action has no parameters.

Return value

If successful, returns a `GetInstanceCertificateOutcome` object containing the location of the fleet's TLS certificate file and certificate chain, which are stored on the instance. A root certificate file, extracted from the certificate chain, is also stored on the instance. If not successful, returns an error message.

For more information about the certificate and certificate chain data, see [GetCertificate Response Elements](#) in the AWS Certificate Manager API Reference.

Example

```
var getInstanceCertificateOutcome = GameLiftServerAPI.GetInstanceCertificate();
```

GetSdkVersion()

Returns the current version number of the SDK built into the server process.

Syntax

```
AwsStringOutcome GetSdkVersion()
```

Parameters

This action has no parameters.

Return value

If successful, returns the current SDK version as an `AwsStringOutcome` object. The returned string includes the version number only (ex. "3.1.5"). If not successful, returns an error message.

Example

```
var getSdkVersionOutcome = GameLiftServerAPI.GetSdkVersion();
```

GetTerminationTime()

Returns the time that a server process is scheduled to be shut down, if a termination time is available. A server process takes this action after receiving an `onProcessTerminate()` callback from the Amazon GameLift Servers service. Amazon GameLift Servers may call `onProcessTerminate()` for the following reasons: (1) for poor health (the server process has reported port health or has not responded to Amazon GameLift Servers), (2) when terminating the instance during a scale-down event, or (3) when an instance is being terminated due to a [spot-instance interruption](#).

If the process has received an `onProcessTerminate()` callback, the value returned is the estimated termination time. If the process has not received an `onProcessTerminate()` callback, an error message is returned. Learn more about [shutting down a server process](#).

Syntax

```
AwsDateTimeOutcome GetTerminationTime()
```

Parameters

This action has no parameters.

Return value

If successful, returns the termination time as an `AwsDateTimeOutcome` object. The value is the termination time, expressed in elapsed ticks since 0001 00:00:00. For example, the date time value 2020-09-13 12:26:40 -000Z is equal to 637355968000000000 ticks. If no termination time is available, returns an error message.

Example

```
var getTerminationTimeOutcome = GameLiftServerAPI.GetTerminationTime();
```

InitSDK()

Initializes the Amazon GameLift Servers SDK. This method should be called on launch, before any other Amazon GameLift Servers-related initialization occurs.

Syntax

```
InitSDKOutcome InitSDK()
```

Parameters

This action has no parameters.

Return value

If successful, returns an `InitSdkOutcome` object indicating that the server process is ready to call [ProcessReady\(\)](#).

Example

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

ProcessEnding()

Notifies the Amazon GameLift Servers service that the server process is shutting down. This method should be called after all other cleanup tasks, including shutting down all active game sessions. This method should exit with an exit code of 0; a non-zero exit code results in an event message that the process did not exit cleanly.

Once the method exits with a code of 0, you can terminate the process with a successful exit code. You can also exit the process with an error code. If you exit with an error code, the fleet event will indicate the process terminated abnormally (`SERVER_PROCESS_TERMINATED_UNHEALTHY`).

Syntax

```
GenericOutcome ProcessEnding()
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
var processEndingOutcome = GameLiftServerAPI.ProcessEnding();  
if (processReadyOutcome.Success)  
    Environment.Exit(0);  
// otherwise, exit with error code  
Environment.Exit(errorCode);
```

ProcessReady()

Notifies the Amazon GameLift Servers service that the server process is ready to host game sessions. Call this method after successfully invoking [InitSDK\(\)](#) and completing setup tasks that are required before the server process can host a game session. This method should be called only once per process.

Syntax

```
GenericOutcome ProcessReady(ProcessParameters processParameters)
```

Parameters

processParameters

A [ProcessParameters](#) object communicating the following information about the server process:

- Names of callback methods, implemented in the game server code, that the Amazon GameLift Servers service invokes to communicate with the server process.
- Port number that the server process is listening on.
- Path to any game session-specific files that you want Amazon GameLift Servers to capture and store.

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example illustrates both the [ProcessReady\(\)](#) call and delegate function implementations.

```
// Set parameters and call ProcessReady
var processParams = new ProcessParameters(
    this.OnGameSession,
    this.OnProcessTerminate,
    this.OnHealthCheck,
    this.OnGameSessionUpdate,
```

```
port,
new LogParameters(new List<string>()           // Examples of log and error files
written by the game server
{
    "C:\\game\\logs",
    "C:\\game\\error"
}))
);

var processReadyOutcome = GameLiftServerAPI.ProcessReady(processParams);

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

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

bool OnHealthCheck()
{
    bool isHealthy;
    // complete health evaluation within 60 seconds and set health
    return isHealthy;
}
```

RemovePlayerSession()

Notifies the Amazon GameLift Servers service that a player with the specified player session ID has disconnected from the server process. In response, Amazon GameLift Servers changes the player slot to available, which allows it to be assigned to a new player.

Syntax

```
GenericOutcome RemovePlayerSession(String playerId)
```

Parameters

playerSessionId

Unique ID issued by Amazon GameLift Servers when a new player session is created. A player session ID is specified in a `PlayerSession` object, which is returned in response to a client call to the *GameLift API* actions [StartGameSessionPlacement](#), [CreateGameSession](#), [DescribeGameSessionPlacement](#), or [DescribePlayerSessions](#).

Type: String

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
Aws::GameLift::GenericOutcome disconnectOutcome =  
    Aws::GameLift::Server::RemovePlayerSession(playerSessionId);
```

StartMatchBackfill()

Sends a request to find new players for open slots in a game session created with FlexMatch. See also the AWS SDK action [StartMatchBackfill\(\)](#). With this action, match backfill requests can be initiated by a game server process that is hosting the game session. Learn more about the [FlexMatch backfill feature](#).

This action is asynchronous. If new players are successfully matched, the Amazon GameLift Servers service delivers updated matchmaker data using the callback function `OnUpdateGameSession()`.

A server process can have only one active match backfill request at a time. To send a new request, first call [StopMatchBackfill\(\)](#) to cancel the original request.

Syntax

```
StartMatchBackfillOutcome StartMatchBackfill (StartMatchBackfillRequest  
    startBackfillRequest);
```

Parameters

StartMatchBackfillRequest

A [StartMatchBackfillRequest](#) object that communicates the following information:

- A ticket ID to assign to the backfill request. This information is optional; if no ID is provided, Amazon GameLift Servers will autogenerate one.
- The matchmaker to send the request to. The full configuration ARN is required. This value can be acquired from the game session's matchmaker data.
- The ID of the game session that is being backfilled.
- Available matchmaking data for the game session's current players.

Required: Yes

Return value

Returns a `StartMatchBackfillOutcome` object with the match backfill ticket ID or failure with an error message.

Example

```
// Build a backfill request
var startBackfillRequest = new AWS.GameLift.Server.Model.StartMatchBackfillRequest()
{
    TicketId = "a ticket ID", //optional
    MatchmakingConfigurationArn = "the matchmaker configuration ARN",
    GameSessionId = GameLiftServerAPI.GetGameSessionId().Result, // gets ID for
current game session
    //get player data for all currently connected players
    MatchmakerData matchmakerData =
        MatchmakerData.FromJson(gameSession.MatchmakerData); // gets matchmaker
data for current players
    // get matchmakerData.Players
    // remove data for players who are no longer connected
    Players = ListOfPlayersRemainingInTheGame
};

// Send backfill request
var startBackfillOutcome = GameLiftServerAPI.StartMatchBackfill(startBackfillRequest);
```

```
// Implement callback function for backfill
void OnUpdateGameSession(GameSession myGameSession)
{
    // game-specific tasks to prepare for the newly matched players and update
    matchmaker data as needed
}
```

StopMatchBackfill()

Cancels an active match backfill request that was created with [StartMatchBackfill\(\)](#). See also the AWS SDK action [StopMatchmaking\(\)](#). Learn more about the [FlexMatch backfill feature](#).

Syntax

```
GenericOutcome StopMatchBackfill (StopMatchBackfillRequest stopBackfillRequest);
```

Parameters

StopMatchBackfillRequest

A [StopMatchBackfillRequest](#) object identifying the matchmaking ticket to cancel:

- ticket ID assigned to the backfill request being canceled
- matchmaker the backfill request was sent to
- game session associated with the backfill request

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
// Set backfill stop request parameters

var stopBackfillRequest = new AWS.GameLift.Server.Model.StopMatchBackfillRequest()
{
    TicketId = "a ticket ID", //optional, if not provided one is autogenerated
    MatchmakingConfigurationArn = "the matchmaker configuration ARN", //from the game
    session matchmaker data
}
```



```
GameSessionId = GameLiftServerAPI.GetGameSessionId().Result //gets the ID for
the current game session
};

var stopBackfillOutcome =
    GameLiftServerAPI.StopMatchBackfillRequest(stopBackfillRequest);
```

TerminateGameSession()

This method is deprecated with version 4.0.1. Instead, the server process should call [ProcessEnding\(\)](#) after a game session has ended.

Notifies the Amazon GameLift Servers service that the server process has ended the current game session. This action is called when the server process will remain active and ready to host a new game session. It should be called only after your game session termination procedure is complete, because it signals to Amazon GameLift Servers that the server process is immediately available to host a new game session.

This action is not called if the server process will be shut down after the game session stops. Instead, call [ProcessEnding\(\)](#) to signal that both the game session and the server process are ending.

Syntax

```
GenericOutcome TerminateGameSession()
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example illustrates a server process at the end of a game session.

```
// game-specific tasks required to gracefully shut down a game session,
// such as notifying players, preserving game state data, and other cleanup
```

```
var terminateGameSessionOutcome = GameLiftServerAPI.TerminateGameSession();  
var processReadyOutcome = GameLiftServerAPI.ProcessReady(processParams);
```

UpdatePlayerSessionCreationPolicy()

Updates the current game session's ability to accept new player sessions. A game session can be set to either accept or deny all new player sessions. (See also the [UpdateGameSession\(\)](#) action in the *Amazon GameLift Servers Service API Reference*).

Syntax

```
GenericOutcome UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy  
    playerSessionPolicy)
```

Parameters

newPlayerSessionPolicy

String value indicating whether the game session accepts new players.

Type: [PlayerSessionCreationPolicy](#) enum. Valid values include:

- **ACCEPT_ALL** – Accept all new player sessions.
- **DENY_ALL** – Deny all new player sessions.

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example sets the current game session's join policy to accept all players.

```
var updatePlayerSessionCreationPolicyOutcomex =  
  
    GameLiftServerAPI.UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy.ACCEPT_ALL);
```

Server SDK (Unreal) for Amazon GameLift Servers -- Actions

Use the server SDK for Unreal to integrate your multiplayer game for hosting with Amazon GameLift Servers. For guidance about the integration process, see [Add Amazon GameLift Servers to your game server with the server SDK](#).

Note

This reference is for an earlier version of the server SDK for Amazon GameLift Servers. For the latest version, see [C++ \(Unreal\) server SDK 5.x for Amazon GameLift Servers -- Actions](#).

This API is defined in `GameLiftServerSDK.h` and `GameLiftServerSDKModels.h`.

To set up the Unreal Engine plugin and see code examples [Integrate Amazon GameLift Servers into an Unreal Engine project](#).

Server SDK (Unreal) for Amazon GameLift Servers -- Data types

Use the Amazon GameLift Servers server SDK for Unreal reference to integrate your multiplayer game for hosting with Amazon GameLift Servers. For guidance about the integration process, see [Add Amazon GameLift Servers to your game server with the server SDK](#).

Note

This reference is for an earlier version of the server SDK for Amazon GameLift Servers. For the latest version, see [C++ \(Unreal\) server SDK 5.x for Amazon GameLift Servers -- Data types](#).

This API is defined in `GameLiftServerSDK.h` and `GameLiftServerSDKModels.h`.

To set up the Unreal Engine plugin and see code examples [Integrate Amazon GameLift Servers into an Unreal Engine project](#).

[Server SDK \(Unreal\) for Amazon GameLift Servers -- Actions](#)

Topics

- [FDescribePlayerSessionsRequest](#)
- [FProcessParameters](#)

- [FStartMatchBackfillRequest](#)
- [FStopMatchBackfillRequest](#)

FDescribePlayerSessionsRequest

This data type is used to specify which player session(s) to retrieve. You can use it as follows:

- Provide a `PlayerSessionId` to request a specific player session.
- Provide a `GameSessionId` to request all player sessions in the specified game session.
- Provide a `PlayerId` to request all player sessions for the specified player.

For large collections of player sessions, use the pagination parameters to retrieve results in sequential blocks.

Contents

GameSessionId

Unique game session identifier. Use this parameter to request all player sessions for the specified game session. Game session ID format is as follows: `arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>`. The value of `<ID string>` is either a custom ID string or (if one was specified when the game session was created) a generated string.

Type: String

Required: No

Limit

Maximum number of results to return. Use this parameter with *NextToken* to get results as a set of sequential pages. If a player session ID is specified, this parameter is ignored.

Type: Integer

Required: No

NextToken

Token indicating the start of the next sequential page of results. Use the token that is returned with a previous call to this action. To specify the start of the result set, do not specify a value. If a player session ID is specified, this parameter is ignored.

Type: String

Required: No

PlayerId

Unique identifier for a player. Player IDs are defined by the developer. See [Generate player IDs](#).

Type: String

Required: No

PlayerSessionId

Unique identifier for a player session.

Type: String

Required: No

PlayerSessionStatusFilter

Player session status to filter results on. Possible player session statuses include the following:

- RESERVED – The player session request has been received, but the player has not yet connected to the server process and/or been validated.
- ACTIVE – The player has been validated by the server process and is currently connected.
- COMPLETED – The player connection has been dropped.
- TIMEDOUT – A player session request was received, but the player did not connect and/or was not validated within the time-out limit (60 seconds).

Type: String

Required: No

FProcessParameters

This data type contains the set of parameters sent to the Amazon GameLift Servers service in a [ProcessReady\(\)](#) call.

Contents

port

Port number the server process will listen on for new player connections. The value must fall into the port range configured for any fleet deploying this game server build. This port number is included in game session and player session objects, which game sessions use when connecting to a server process.

Type: Integer

Required: Yes

logParameters

Object with a list of directory paths to game session log files.

Type: TArray<FString>

Required: No

onStartGameSession

Name of callback function that the Amazon GameLift Servers service invokes to activate a new game session. Amazon GameLift Servers calls this function in response to the client request [CreateGameSession](#). The callback function takes a [GameSession](#) object (defined in the *Amazon GameLift Servers Service API Reference*).

Type: FOnStartGameSession

Required: Yes

onProcessTerminate

Name of callback function that the Amazon GameLift Servers service invokes to force the server process to shut down. After calling this function, Amazon GameLift Servers waits five minutes for the server process to shut down and respond with a [ProcessEnding\(\)](#) call before it shuts down the server process.

Type: FSimpleDelegate

Required: No

onHealthCheck

Name of callback function that the Amazon GameLift Servers service invokes to request a health status report from the server process. Amazon GameLift Servers calls this function every 60 seconds. After calling this function Amazon GameLift Servers waits 60 seconds for a response, and if none is received, records the server process as unhealthy.

Type: FOnHealthCheck

Required: No

onUpdateGameSession

Name of callback function that the Amazon GameLift Servers service invokes to pass an updated game session object to the server process. Amazon GameLift Servers calls this function when a [match backfill](#) request has been processed in order to provide updated matchmaker data. It passes a [GameSession](#) object, a status update (updateReason), and the match backfill ticket ID.

Type: FOnUpdateGameSession

Required: No

FStartMatchBackfillRequest

This data type is used to send a matchmaking backfill request. The information is communicated to the Amazon GameLift Servers service in a [StartMatchBackfill\(\)](#) call.

Contents

GameSessionArn

Unique game session identifier. The API action [GetGameSessionId\(\)](#) returns the identifier in ARN format.

Type: FString

Required: Yes

MatchmakingConfigurationArn

Unique identifier, in the form of an ARN, for the matchmaker to use for this request. To find the matchmaker that was used to create the original game session, look in the game session

object, in the matchmaker data property. Learn more about matchmaker data in [Work with matchmaker data](#).

Type: FString

Required: Yes

Players

A set of data representing all players who are currently in the game session. The matchmaker uses this information to search for new players who are good matches for the current players. See the *Amazon GameLift Servers API Reference Guide* for a description of the Player object format. To find player attributes, IDs, and team assignments, look in the game session object, in the matchmaker data property. If latency is used by the matchmaker, gather updated latency for the current region and include it in each player's data.

Type: TArray<[FPlayer](#)>

Required: Yes

TicketId

Unique identifier for a matchmaking or match backfill request ticket. If no value is provided here, Amazon GameLift Servers will generate one in the form of a UUID. Use this identifier to track the match backfill ticket status or cancel the request if needed.

Type: FString

Required: No

FStopMatchBackfillRequest

This data type is used to cancel a matchmaking backfill request. The information is communicated to the Amazon GameLift Servers service in a [StopMatchBackfill\(\)](#) call.

Contents

GameSessionArn

Unique game session identifier associated with the request being canceled.

Type: FString

Required: Yes

MatchmakingConfigurationArn

Unique identifier of the matchmaker this request was sent to.

Type: FString

Required: Yes

TicketId

Unique identifier of the backfill request ticket to be canceled.

Type: FString

Required: Yes

[Server SDK \(Unreal\) for Amazon GameLift Servers -- Data types](#)

Topics

- [AcceptPlayerSession\(\)](#)
- [ActivateGameSession\(\)](#)
- [DescribePlayerSessions\(\)](#)
- [GetGameSessionId\(\)](#)
- [GetInstanceCertificate\(\)](#)
- [GetSdkVersion\(\)](#)
- [InitSDK\(\)](#)
- [ProcessEnding\(\)](#)
- [ProcessReady\(\)](#)
- [RemovePlayerSession\(\)](#)
- [StartMatchBackfill\(\)](#)
- [StopMatchBackfill\(\)](#)
- [TerminateGameSession\(\)](#)
- [UpdatePlayerSessionCreationPolicy\(\)](#)

AcceptPlayerSession()

Notifies the Amazon GameLift Servers service that a player with the specified player session ID has connected to the server process and needs validation. Amazon GameLift Servers verifies that the player session ID is valid—that is, that the player ID has reserved a player slot in the game session. Once validated, Amazon GameLift Servers changes the status of the player slot from RESERVED to ACTIVE.

Syntax

```
FGameLiftGenericOutcome AcceptPlayerSession(const FString& playerId)
```

Parameters

playerSessionId

Unique ID issued by the Amazon GameLift Servers service in response to a call to the AWS SDK Amazon GameLift Servers API action [CreatePlayerSession](#). The game client references this ID when connecting to the server process.

Type: FString

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

ActivateGameSession()

Notifies the Amazon GameLift Servers service that the server process has activated a game session and is now ready to receive player connections. This action should be called as part of the `onStartGameSession()` callback function, after all game session initialization has been completed.

Syntax

```
FGameLiftGenericOutcome ActivateGameSession()
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

DescribePlayerSessions()

Retrieves player session data, including settings, session metadata, and player data. Use this action to get information for a single player session, for all player sessions in a game session, or for all player sessions associated with a single player ID.

Syntax

```
FGameLiftDescribePlayerSessionsOutcome DescribePlayerSessions(const  
    FGameLiftDescribePlayerSessionsRequest &describePlayerSessionsRequest)
```

Parameters

describePlayerSessionsRequest

A [FDescribePlayerSessionsRequest](#) object describing which player sessions to retrieve.

Required: Yes

Return value

If successful, returns a [FDescribePlayerSessionsRequest](#) object containing a set of player session objects that fit the request parameters. Player session objects have a structure identical to the AWS SDK Amazon GameLift Servers API [PlayerSession](#) data type.

GetGameSessionId()

Retrieves the ID of the game session currently being hosted by the server process, if the server process is active.

Syntax

```
FGameLiftStringOutcome GetGameSessionId()
```

Parameters

This action has no parameters.

Return value

If successful, returns the game session ID as an `FGameLiftStringOutcome` object. If not successful, returns an error message.

GetInstanceCertificate()

Retrieves the file location of a pem-encoded TLS certificate that is associated with the fleet and its instances. AWS Certificate Manager generates this certificate when you create a new fleet with the certificate configuration set to `GENERATED`. Use this certificate to establish a secure connection with a game client and to encrypt client/server communication.

Syntax

```
FGameLiftGetInstanceCertificateOutcome GetInstanceCertificate()
```

Parameters

This action has no parameters.

Return value

If successful, returns a `GetInstanceCertificateOutcome` object containing the location of the fleet's TLS certificate file and certificate chain, which are stored on the instance. A root certificate file, extracted from the certificate chain, is also stored on the instance. If not successful, returns an error message.

For more information about the certificate and certificate chain data, see [GetCertificate Response Elements](#) in the AWS Certificate Manager API Reference.

GetSdkVersion()

Returns the current version number of the SDK built into the server process.

Syntax

```
FGameLiftStringOutcome GetSdkVersion();
```

Parameters

This action has no parameters.

Return value

If successful, returns the current SDK version as an `FGameLiftStringOutcome` object. The returned string includes the version number only (ex. "3.1.5"). If not successful, returns an error message.

Example

```
Aws::GameLift::AwsStringOutcome SdkVersionOutcome =  
    Aws::GameLift::Server::GetSdkVersion();
```

InitSDK()

Initializes the Amazon GameLift Servers SDK. This method should be called on launch, before any other Amazon GameLift Servers-related initialization occurs.

Syntax

```
FGameLiftGenericOutcome InitSDK()
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

ProcessEnding()

Notifies the Amazon GameLift Servers service that the server process is shutting down. This method should be called after all other cleanup tasks, including shutting down all active game sessions. This method should exit with an exit code of 0; a non-zero exit code results in an event message that the process did not exit cleanly.

Syntax

```
FGameLiftGenericOutcome ProcessEnding()
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

ProcessReady()

Notifies the Amazon GameLift Servers service that the server process is ready to host game sessions. Call this method after successfully invoking [InitSDK\(\)](#) and completing setup tasks that are required before the server process can host a game session. This method should be called only once per process.

Syntax

```
FGameLiftGenericOutcome ProcessReady(FProcessParameters &processParameters)
```

Parameters

FProcessParameters

A [FProcessParameters](#) object communicating the following information about the server process:

- Names of callback methods, implemented in the game server code, that the Amazon GameLift Servers service invokes to communicate with the server process.
- Port number that the server process is listening on.
- Path to any game session-specific files that you want Amazon GameLift Servers to capture and store.

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

See the sample code in [Using the Unreal Engine Plugin](#).

RemovePlayerSession()

Notifies the Amazon GameLift Servers service that a player with the specified player session ID has disconnected from the server process. In response, Amazon GameLift Servers changes the player slot to available, which allows it to be assigned to a new player.

Syntax

```
FGameLiftGenericOutcome RemovePlayerSession(const FString& playerId)
```

Parameters

playerSessionId

Unique ID issued by the Amazon GameLift Servers service in response to a call to the AWS SDK Amazon GameLift Servers API action [CreatePlayerSession](#). The game client references this ID when connecting to the server process.

Type: FString

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

StartMatchBackfill()

Sends a request to find new players for open slots in a game session created with FlexMatch. See also the AWS SDK action [StartMatchBackfill\(\)](#). With this action, match backfill requests can be initiated by a game server process that is hosting the game session. Learn more about the [FlexMatch backfill feature](#).

This action is asynchronous. If new players are successfully matched, the Amazon GameLift Servers service delivers updated matchmaker data using the callback function `OnUpdateGameSession()`.

A server process can have only one active match backfill request at a time. To send a new request, first call [StopMatchBackfill\(\)](#) to cancel the original request.

Syntax

```
FGameLiftStringOutcome StartMatchBackfill (FStartMatchBackfillRequest  
&startBackfillRequest);
```

Parameters

FStartMatchBackfillRequest

A [FStartMatchBackfillRequest](#) object that communicates the following information:

- A ticket ID to assign to the backfill request. This information is optional; if no ID is provided, Amazon GameLift Servers will autogenerate one.
- The matchmaker to send the request to. The full configuration ARN is required. This value can be acquired from the game session's matchmaker data.
- The ID of the game session that is being backfilled.
- Available matchmaking data for the game session's current players.

Required: Yes

Return value

If successful, returns the match backfill ticket as a `FGameLiftStringOutcome` object. If not successful, returns an error message. Ticket status can be tracked using the AWS SDK action [DescribeMatchmaking\(\)](#).

StopMatchBackfill()

Cancels an active match backfill request that was created with [StartMatchBackfill\(\)](#). See also the AWS SDK action [StopMatchmaking\(\)](#). Learn more about the [FlexMatch backfill feature](#).

Syntax

```
FGameLiftGenericOutcome StopMatchBackfill (FStopMatchBackfillRequest  
&stopBackfillRequest);
```

Parameters

StopMatchBackfillRequest

A [FStopMatchBackfillRequest](#) object identifying the matchmaking ticket to cancel:

- ticket ID assigned to the backfill request being canceled
- matchmaker the backfill request was sent to
- game session associated with the backfill request

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

TerminateGameSession()

This method is deprecated with version 4.0.1. Instead, the server process should call [ProcessEnding\(\)](#) after a game session has ended.

Notifies the Amazon GameLift Servers service that the server process has ended the current game session. This action is called when the server process will remain active and ready to host a new game session. It should be called only after your game session termination procedure is complete, because it signals to Amazon GameLift Servers that the server process is immediately available to host a new game session.

This action is not called if the server process will be shut down after the game session stops. Instead, call [ProcessEnding\(\)](#) to signal that both the game session and the server process are ending.

Syntax

```
FGameLiftGenericOutcome TerminateGameSession()
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

UpdatePlayerSessionCreationPolicy()

Updates the current game session's ability to accept new player sessions. A game session can be set to either accept or deny all new player sessions. (See also the [UpdateGameSession\(\)](#) action in the *Amazon GameLift Servers Service API Reference*).

Syntax

```
FGameLiftGenericOutcome UpdatePlayerSessionCreationPolicy(EPlayerSessionCreationPolicy policy)
```

Parameters

Policy

Value indicating whether the game session accepts new players.

Type: `EPlayerSessionCreationPolicy` enum. Valid values include:

- **ACCEPT_ALL** – Accept all new player sessions.
- **DENY_ALL** – Deny all new player sessions.

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Game session placement events

Amazon GameLift Servers emits events for each game session placement request as it is processed. You can publish these events to an Amazon SNS topic, as described in [Set up event notification for game session placement](#). These events are also emitted to Amazon CloudWatch Events in near real time and on a best-effort basis.

This topic describes the structure of game session placement events and provides an example for each event type. For more information on the status of game session placement requests, see [GameSessionPlacement](#) in the *Amazon GameLift Servers API Reference*.

Placement event syntax

Events are represented as JSON objects. Event structure conforms to the CloudWatch Events pattern, with similar top-level fields and service-specific details.

Top-level fields include the following (see [event pattern](#) for more detail):

version

This field is always set to 0 (zero).

id

Unique tracking identifier for the event.

detail-type

Value is always `GameLift Queue Placement Event`.

source

Value is always `aws.gamelift`.

account

The AWS account that is being used to manage Amazon GameLift Servers.

time

Event timestamp in ISO 8601 format.

region

The AWS Region where the placement request is being processed. This is the Region where the game session queue in use resides.

resources

ARN value of the game session queue that is processing the placement request.

PlacementFulfilled

The placement request has been successfully fulfilled. A new game session has been started and new player sessions have been created for each player listed in the game session placement request. Player connection information is available.

Detail syntax:

placementId

A unique identifier assigned to the game session placement request.

port

The port number for the new game session.

gameSessionArn

The ARN identifier for the new game session.

ipAddress

The IP address of the game session.

dnsName

The DNS identifier assigned to the instance that is running the new game session. The value format is different depending on whether the instance running the game session is TLS-enabled. When connecting to a game session on a TLS-enabled fleet, players must use the DNS name, not the IP address.

TLS-enabled fleets: `<unique identifier>.<region identifier>.amazongamelift.com`.

Non-TLS-enabled fleets: `ec2-<unique identifier>.compute.amazonaws.com`.

startTime

Time stamp indicating when this request was placed in the queue, in ISO 8601 format.

endTime

Time stamp indicating when this request was fulfilled, in ISO 8601 format.

gameSessionRegion

AWS Home region of the fleet that is hosting the game session.

gameSessionLocation

The fleet location where the game session is running. This value might specify the fleet's home region or a remote location.

playerGatewayStatus

The state of the player gateway at the fleet location running this game session.

Possible values include:

- **ENABLED** – Player gateway is available for routing player connections for this game session.
- **DISABLED** – Player gateway is not available for this game session.

computeName

A descriptive label for the compute resource that is hosting the game session. For EC2 fleets, this is the EC2 instance ID. For Container fleets, each game server container group on a fleet instance is assigned a compute name. For Anywhere fleets, this is the custom compute name.

placedPlayerSessions

The collection of player sessions that have been created for each player in the game session placement request.

Example

```
{
  "version": "0",
  "id": "1111aaaa-bb22-cc33-dd44-5555eeee66ff",
  "detail-type": "GameLift Queue Placement Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2021-03-01T15:50:52Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:gamelift:us-west-2:123456789012:gamesessionqueue/MegaFrogRace-NA"
  ],
  "detail": {
    "type": "PlacementFulfilled",
    "placementId": "9999ffff-88ee-77dd-66cc-5555bb44aa",
    "port": "6262",
    "gameSessionArn": "arn:aws:gamelift:us-west-2::gamesession/
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa/us-
east-1/4444dddd-55ee-66ff-77aa-8888bbbb99cc",
    "ipAddress": "98.987.98.987",
    "dnsName": "ec2-12-345-67-890.us-west-2.compute.amazonaws.com",
    "startTime": "2021-03-01T15:50:49.741Z",
    "endTime": "2021-03-01T15:50:52.084Z",
    "gameSessionRegion": "us-west-2",
    "playerGatewayStatus": "ENABLED",
    "gameSessionLocation": "us-east-1",
    "computeName": "i-1234567890abcdef0",
    "placedPlayerSessions": [
```

```
{
  "playerId": "player-1"
  "playerSessionId": "psess-1232131232324124123123"
}
]
```

PlacementCancelled

The placement request was canceled with a call to the GameLift service [StopGameSessionPlacement](#).

Detail:

placementId

A unique identifier assigned to the game session placement request.

startTime

Time stamp indicating when this request was placed in the queue, in ISO 8601 format.

endTime

Time stamp indicating when this request was cancelled, in ISO 8601 format.

Example

```
{
  "version": "0",
  "id": "1111aaaa-bb22-cc33-dd44-5555eeee66ff",
  "detail-type": "GameLift Queue Placement Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2021-03-01T15:50:52Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:gamelift:us-west-2:123456789012:gamesessionqueue/MegaFrogRace-NA"
  ],
  "detail": {
    "type": "PlacementCancelled",
```

```
"placementId": "9999ffff-88ee-77dd-66cc-5555bb44aa",
"startTime": "2021-03-01T15:50:49.741Z",
"endTime": "2021-03-01T15:50:52.084Z"
}
}
```

PlacementTimedOut

Game session placement did not successfully complete before the queue's time limit expired. The placement request can be resubmitted as needed.

Detail:

placementId

A unique identifier assigned to the game session placement request.

startTime

Time stamp indicating when this request was placed in the queue, in ISO 8601 format.

endTime

Time stamp indicating when this request was cancelled, in ISO 8601 format.

Example

```
{
  "version": "0",
  "id": "1111aaaa-bb22-cc33-dd44-5555eeee66ff",
  "detail-type": "GameLift Queue Placement Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2021-03-01T15:50:52Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:gamelift:us-west-2:123456789012:gamesessionqueue/MegaFrogRace-NA"
  ],
  "detail": {
    "type": "PlacementTimedOut",
    "placementId": "9999ffff-88ee-77dd-66cc-5555bb44aa",
    "startTime": "2021-03-01T15:50:49.741Z",
```

```
    "endTime": "2021-03-01T15:50:52.084Z"  
  }  
}
```

PlacementFailed

Amazon GameLift Servers was not able to fulfill the game session request. This is generally caused by an unexpected internal error. The placement request can be resubmitted as needed.

Detail:

placementId

A unique identifier assigned to the game session placement request.

startTime

Time stamp indicating when this request was placed in the queue, in ISO 8601 format.

endTime

Time stamp indicating when this request failed, in ISO 8601 format.

Example

```
{  
  "version": "0",  
  "id": "39c978f3-ba46-3f7c-e787-55bfcca1bd31",  
  "detail-type": "GameLift Queue Placement Event",  
  "source": "aws.gamelift",  
  "account": "252386620677",  
  "time": "2021-03-01T15:50:52Z",  
  "region": "us-east-1",  
  "resources": [  
    "arn:aws:gamelift:us-west-2:252386620677:gamesessionqueue/MegaFrogRace-NA"  
  ],  
  "detail": {  
  
    "type": "PlacementFailed",  
    "placementId": "e4a1119a-39af-45cf-a990-ef150fe0d453",  
    "startTime": "2021-03-01T15:50:49.741Z",  
    "endTime": "2021-03-01T15:50:52.084Z"  
  }  
}
```



```
}
```

Amazon GameLift Servers AMI versions

The following table identifies the latest Amazon machine images (AMIs) that Amazon GameLift Servers uses for managed EC2 hosting. As described in [Configuration and vulnerability analysis in Amazon GameLift Servers](#), you must regularly create new Amazon GameLift Servers managed EC2 fleets to deploy the latest AMI version updates.

AMIs for use with server SDK for Amazon GameLift Servers 5+

Amazon GameLift Servers uses the following AMIs to host game servers that are integrated with server SDK for Amazon GameLift Servers version 5.

Amazon GameLift Servers image	Architecture	Most recent patch	Base AWS image
Amazon Linux 2023 BASE_AMI_LINUX_2023	x86	2026-06-11	Amazon Linux 2023 AMI 2023.12.2 0260608.0 x86_64 HVM kernel-6.1 (release notes)
Amazon Linux 2 BASE_AMI_LINUX_2_SDK_5	x86	2026-06-11	Amazon Linux 2 Kernel 5.10 AMI 2.0.20260608.0 x86_64 HVM gp2 (release notes)
Windows 2016 BASE_AMI_WINDOWS_2016_SDK_5	x86	2026-06-11	Windows_Server-2016-English-Full-Base-2026.05.13 (release notes)
Amazon Linux 2023 BASE_AMI_LINUX_2023_ARM	ARM64	2026-06-11	Amazon Linux 2023 AMI 2023.12.2 0260608.0 arm64 HVM kernel-6.1 (release notes)
Amazon Linux 2	ARM64	2026-06-11	Amazon Linux 2 LTS Arm64 Kernel 5.10 AMI 2.0.20260608.0 arm64 HVM gp2 (release notes)

Amazon GameLift Servers image	Architecture	Most recent patch	Base AWS image
BASE_AMI_LINUX_2_ARM			
Windows 2022 BASE_AMI_WINDOWS_2022	x86	2026-06-11	Windows_Server-2022-English-Full-Base-2026.05.13 (release notes)

AMIs for use with server SDK for Amazon GameLift Servers 4

Amazon GameLift Servers uses the following AMIs to host game servers that are integrated with server SDK for Amazon GameLift Servers version 4 or earlier.

Amazon GameLift Servers image	Architecture	Most recent patch	Base AWS image
Amazon Linux 2 BASE_AMI_LINUX_2_SDK_4	x86	2026-06-11	Amazon Linux 2 AMI 2.0.20260608.0 x86_64 HVM gp2 (release notes)
Amazon Linux 2023 BASE_AMI_LINUX_2023_SDK_4 <i>Available in the China (Beijing) and China (Ningxia) Regions only.</i>	x86	2026-06-11	Amazon Linux 2023 AMI 2023.12.20260608.0 x86_64 HVM kernel-6.1 (release notes)
Windows 2016 BASE_AMI_WINDOWS_2016_SDK_4	x86	2026-06-11	Windows_Server-2016-English-Full-Base-2026.05.13 (release notes)

For more information, see the following resources:

- [Amazon Linux 2023 release notes](#)
- [Amazon Linux 2 release notes](#)
- [AWS Windows AMI version history](#)
- [Description of Software Update Services and Windows Server Update Services changes in content for 2024](#)

Amazon GameLift Servers service locations

Amazon GameLift Servers features are available across multiple AWS Regions and Local Zones. You can design a hosting solution that puts your game servers right where your players are located.

Supported AWS locations

The following table is a list of AWS Regions and Local Zones that support Amazon GameLift Servers resources. It indicates the types of resources that you can create in each location.

Geographic location	Location code	Home Region for managed fleets (single location)	Home Region for managed fleets (multi-location)	Remote location for managed fleets (multi-location)	Anywhere fleet	Game session queue	FlexMatch and rule set	DDoS protection (player gateway)
US East (N. Virginia)	us-east-1	Yes	Yes	Yes	Yes	Yes	Yes	Yes [†]
US East (Ohio)	us-east-2	Yes		Yes	Yes	Yes		
US West (N. California)	us-west-1	Yes		Yes	Yes	Yes		

Geographic location	Location code	Home Region for managed fleets (single location)	Home Region for managed fleets (multi-location)	Remote location for managed fleets (multi-location)	Anywhere fleet	Game session queue	FlexMatchmaker and rule set	DDoS protection (player gateway)
US West (Oregon)	us-west-2	Yes	Yes	Yes	Yes	Yes	Yes	Yes [†]
Africa (Cape Town)	af-south-1			Yes				
Asia Pacific (Thailand)	ap-southeast-7			Yes				
Asia Pacific (Hong Kong)	ap-east-1			Yes				
Asia Pacific (Malaysia)	ap-southeast-5			Yes				
Asia Pacific (Mumbai)	ap-south-1	Yes		Yes	Yes	Yes		
Asia Pacific (Osaka)	ap-northeast-3			Yes				
Asia Pacific (Seoul)	ap-northeast-2	Yes	Yes	Yes	Yes	Yes	Yes	Yes [†]

Geographic location	Location code	Home Region for managed fleets (single location)	Home Region for managed fleets (multi-location)	Remote location for managed fleets (multi-location)	Anywhere fleet	Game session queue	FlexMatchmaker and rule set	DDoS protection (player gateway)
Asia Pacific (Singapore)	ap-southeast-1	Yes		Yes	Yes	Yes		
Asia Pacific (Sydney)	ap-southeast-2	Yes	Yes	Yes	Yes	Yes	Yes	Yes [†]
Asia Pacific (Tokyo)	ap-northeast-1	Yes	Yes	Yes	Yes	Yes	Yes	Yes [†]
Canada (Central)	ca-central-1	Yes		Yes	Yes	Yes		
Europe (Frankfurt)	eu-central-1	Yes	Yes	Yes	Yes	Yes	Yes	Yes [†]
Europe (Ireland)	eu-west-1	Yes	Yes	Yes	Yes	Yes	Yes	Yes [†]
Europe (London)	eu-west-2	Yes		Yes	Yes	Yes		
Europe (Milan)	eu-south-1			Yes				

Geographic location	Location code	Home Region for managed fleets (single location)	Home Region for managed fleets (multi-location)	Remote location for managed fleets (multi-location)	Anywhere fleet	Game session queue	FlexMatchmaker and rule set	DDoS protection (player gateway)
Europe (Paris)	eu-west-3			Yes				
Europe (Stockholm)	eu-north-1			Yes				
Middle East (Bahrain)	me-south-1			Yes				
South America (São Paulo)	sa-east-1	Yes		Yes	Yes	Yes		
Atlanta local zone	us-east-1-atl-1			Yes				
Chicago local zone	us-east-1-chi-1			Yes				
Dallas local zone [*]	us-east-1-dfw-1			Yes				

Geographic location	Location code	Home Region for managed fleets (single location)	Home Region for managed fleets (multi-location)	Remote location for managed fleets (multi-location)	Anywhere fleet	Game session queue	FlexMatchmaker and rule set	DDoS protection (player gateway)
Dallas local zone 2	us-east-1-dfw-2			Yes				
Denver local zone	us-west-2-den-1			Yes				
Houston local zone	us-east-1-iah-1			Yes				
Kansas City local zone	us-east-1-mci-1			Yes				
Los Angeles local zone	us-west-2-lax-1			Yes				
Phoenix local zone	us-west-2-phx-1			Yes				
Lagos, Nigeria local zone	af-south-1-los-1			Yes				

* Available to AWS accounts that have already opted in.

† Valid fleet home region for DDoS protection (player gateway).

Note

Some AWS Regions and all Local Zones are not enabled by default for an AWS account. You must first opt in to these Regions or Local Zones before you can deploy game servers to those locations. For more information about Regions that aren't enabled by default and how to enable them, see [Managing AWS Regions](#) in the *AWS General Reference*. See [Getting started with Local Zones](#) in the *AWS Local Zones User Guide*. (Fleets created before February 28, 2022 are not affected by this requirement.)

In addition, you must update your Amazon GameLift Servers administrator policy to allow the `ec2:DescribeRegions` action. For a policy example with Regions that aren't enabled by default, see [Administration permission examples](#).

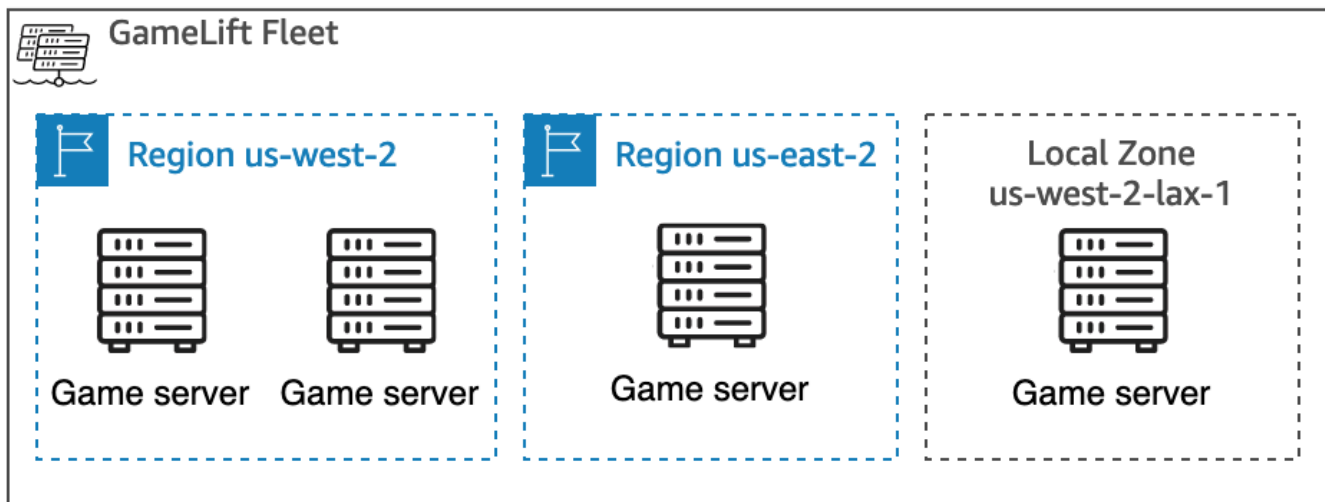
Locations for managed hosting

Amazon GameLift Servers managed hosting deploys fleets of game server resources. Each fleet is created in an AWS Region, which is the fleet's *home region*. A fleet's home Region is referenced in the fleet's Amazon Resource Number (ARN).

You can deploy a *single-region fleet*, with hosting resources in the home region only. Alternatively, you can deploy a *multi-location fleet*, with hosting resources in multiple geographic locations. A multi-location fleet has a home region and one or more *remote locations*. There are no limitations on how you group locations together in a fleet. When managing hosting capacity for a multi-location fleet, you can set capacity for each location individually.

Remote locations for a multi-location fleet can be other AWS Regions or Local Zones. A *Local Zone* is an extension of an AWS Region. It lets you place compute resources closer to users to provide lower-latency gameplay. For more information, see [AWS Local Zones](#). The location code for a Local Zone is its parent Region code followed by a physical location identifier. For example, the code for the Los Angeles Local Zone is `us-west-2-lax-1`.

The following diagram illustrates a multi-location fleet with resources in two AWS Regions and one Local Zone. The fleet's home Region is `us-west-2`, and it has two remote locations: `us-east-2` Region and `us-west-2-lax-1` Local Zone.



In addition to fleet resources, managed hosting with Amazon GameLift Servers can also use the following types of resources. You create these resources in a specific AWS Region that supports the resource type.

- *Build* – This is a game server build to be hosted with a managed EC2 fleet. Create a build resource in the same region as the fleet that it will be deployed to.
- *Script* – This is a configuration script for hosting a game with Amazon GameLift Servers Realtime. Create a script resource in the same Region as the fleet that it will be deployed to.
- *Container group definition* – This is a configuration for running containers on a managed container fleet. It identifies one or more *container images* with game server software to deploy to the container fleet. Create a container group definition in the same Region as the fleet they'll be deployed to.
- *Game session queue* – This resource processes requests for game sessions and initiates new game sessions. Processing takes place in the AWS Region where the queue is located. To reduce latency in the game session placement process, create a queue geographically near the players that will use it.

Locations for Amazon GameLift Servers Anywhere

An Amazon GameLift Servers Anywhere fleet is a collection of hosting hardware that you provide. You manage all activity on your hosting resources, including deploying game server software, keeping it updated, and starting server processes. You create an Anywhere fleet to connect the Amazon GameLift Servers service with your self-managed hosting resources. Amazon GameLift

Servers manages game session placement--processing player join requests, locating available hosting resources, initiating new game sessions, and providing game clients with connection information. You can create an Anywhere fleet in any of the AWS Regions that support them.

You add instances of hosting hardware to an Anywhere fleet by registering it. Each registered instance must have a custom location associated with it. Custom locations are not related to AWS Regions or Local Zones. They are used to represent the physical location of the hardware.

For more information about creating an Anywhere fleet and testing your game server integration, see [Create an Amazon GameLift Servers Anywhere fleet](#) and [Set up local testing with Amazon GameLift Servers Anywhere](#).

Locations for Amazon GameLift Servers FlexMatch

FlexMatch resources are used to process player requests for matchmaking. They include a matchmaking configuration resource and a rule set resource. Processing takes place in the AWS Region where the FlexMatch resources are located. To reduce latency in the matchmaking process, create the resources geographically near the players that will use it. A matchmaking configuration and the rule set it uses must be located in the same AWS Region. You can create FlexMatch resources in any of the AWS Regions that support them.

For more information about setting up FlexMatch for your hosting solution, see the [Amazon GameLift Servers FlexMatch developer guide](#).

Amazon GameLift Servers in China

When using Amazon GameLift Servers for resources in the China (Beijing) Region, operated by Sinnet, or the China (Ningxia) Region, operated by NWCD, you must have a separate AWS (China) account. Be aware that some features are unavailable in the China Regions. For more information about using Amazon GameLift Servers in these Regions, see the following resources:

- [Amazon Web Services in China](#)
- [Amazon GameLift Servers](#) (Getting Started with Amazon Web Services in China)

Amazon GameLift Servers endpoints and quotas

- For service endpoints and beacon endpoints that you can use to connect programmatically with the service, see [Amazon GameLift Servers service endpoints](#).

- For Amazon GameLift Servers quotas on the use of service resources or operations per AWS account, see [Amazon GameLift Servers service quotas](#). For more information on quotas and how to request an increase, see [AWS service quotas](#). You can request quota increases by using the Amazon GameLift Servers console.

Amazon GameLift Servers API limits

The tables in this section list the default rate limits for Amazon GameLift Servers API operations. These limits help maintain service stability and ensure fair usage. Each API operation has specific limits based on its resource impact and usage patterns.

By default, the throttle limit for each customer account is set to 500 for burst requests and 500 for normal traffic. In addition, each API has its own default limit pair as outlined below.

API limits reference

The following table lists the default rate limits for Amazon GameLift Servers API operations. APIs marked with resource-level throttling may be subject to additional limits to prevent database hot key issues.

API limits reference

API action	Burst limit	Rate limit	Resource-level throttling	Notes
AcceptMatch	100	100	No	Limit is per account.
ClaimGameServer	40	20	No	Limit is per account.
CreateAlias	20	20	No	Limit is per account.
CreateBuild	20	20	No	Limit is per account.
CreateContainerFleet	4	1	No	Limit is per account.
CreateContainerGroupDefinition	20	20	No	Limit is per account.

API action	Burst limit	Rate limit	Resource-level throttling	Notes
CreateFleet	4	1	No	Limit is per account.
CreateFleetLocations	20	20	No	Limit is per account.
CreateGameServerGroup	20	20	No	Limit is per account.
CreateGameSession	50	50	No	Limit is per account.
CreateGameSessionQueue	20	20	No	Limit is per account.
CreateLocation	20	20	No	Limit is per account.
CreateMatchmakingConfiguration	20	20	No	Limit is per account.
CreateMatchmakingRuleSet	20	20	No	Limit is per account.
CreatePlayerSession	200	200	No	Limit is per account.
CreatePlayerSessions	200	200	No	Limit is per account.
CreateScript	20	20	No	Limit is per account.
CreateVpcPeeringAuthorization	1	0.1	No	Limit is per account.
CreateVpcPeeringConnection	1	0.1	No	Limit is per account.
DeleteAlias	20	20	No	Limit is per account.
DeleteBuild	20	20	No	Limit is per account.

API action	Burst limit	Rate limit	Resource-level throttling	Notes
DeleteContainerFleet	20	20	No	Limit is per account.
DeleteContainerGroupDefinition	20	20	No	Limit is per account.
DeleteFleet	20	20	No	Limit is per account.
DeleteFleetLocations	20	20	No	Limit is per account.
DeleteGameServerGroup	20	20	No	Limit is per account.
DeleteGameSessionQueue	20	20	No	Limit is per account.
DeleteLocation	20	20	No	Limit is per account.
DeleteMatchmakingConfiguration	20	20	No	Limit is per account.
DeleteMatchmakingRuleSet	20	20	No	Limit is per account.
DeleteScalingPolicy	20	20	No	Limit is per account.
DeleteScript	20	20	No	Limit is per account.
DeleteVpcPeeringAuthorization	1	0.1	No	Limit is per account.
DeleteVpcPeeringConnection	1	0.1	No	Limit is per account.
DeregisterGameServer	40	20	No	Limit is per account.

API action	Burst limit	Rate limit	Resource-level throttling	Notes
DeregisterCompute	20	20	No	Limit is per account.
DescribeAlias	30	30	No	Limit is per account.
DescribeBuild	30	30	No	Limit is per account.
DescribeCompute	30	30	No	Limit is per account.
DescribeContainerFleet	30	30	No	Limit is per account.
DescribeContainerGroupDefinition	30	30	No	Limit is per account.
DescribeContainerGroupPortMappings	100	100	No	Limit is per account.
DescribeEC2InstanceLimits	40	30	No	Limit is per account.
DescribeFleetAttributes	30	30	No	Limit is per account.
DescribeFleetCapacity	30	30	No	Limit is per account.
DescribeFleetDeployment	30	30	No	Limit is per account.
DescribeFleetEvents	30	30	No	Limit is per account.
DescribeFleetLocationAttributes	30	30	No	Limit is per account.

API action	Burst limit	Rate limit	Resource-level throttling	Notes
DescribeFleetLocationCapacity	30	30	No	Limit is per account.
DescribeFleetLocationUtilization	30	30	No	Limit is per account.
DescribeFleetPortSettings	30	30	No	Limit is per account.
DescribeFleetUtilization	30	30	No	Limit is per account.
DescribeGameServer	40	20	No	Limit is per account.
DescribeGameServerGroup	30	30	No	Limit is per account.
DescribeGameServerInstances	30	30	No	Limit is per account.
DescribeGameSessionDetails	30	30	No	Limit is per account. Consider using DescribeGameSessions instead as it has a higher default limit and supports resource level limits.
DescribeGameSessionPlacement	75	75	Yes	Limit is per account. Subject to resource-level throttling.
DescribeGameSessionQueues	30	30	No	Limit is per account.
DescribeGameSessions	20	20	Yes	Limit is per account. Subject to resource-level throttling.

API action	Burst limit	Rate limit	Resource-level throttling	Notes
DescribeInstances	30	30	No	Limit is per account.
DescribeMatchmaking	200	200	Yes	Limit is per account. Subject to resource-level throttling.
DescribeMatchmakingConfigurations	30	30	No	Limit is per account.
DescribeMatchmakingRuleSets	30	30	No	Limit is per account.
DescribePlayerSessions	200	200	Yes	Limit is per account. Subject to resource-level throttling.
DescribeRuntimeConfiguration	30	30	No	Limit is per account.
DescribeScalingPolicies	30	30	No	Limit is per account.
DescribeScript	30	30	No	Limit is per account.
DescribeVpcPeeringAuthorizations	1	0.1	No	Limit is per account.
DescribeVpcPeeringConnections	1	0.1	No	Limit is per account.
GetComputeAccess	4	1	No	Limit is per account.
GetComputeAuthToken	10	5	No	Limit is per account.
GetGameSessionLogUrl	4	1	No	Limit is per account.

API action	Burst limit	Rate limit	Resource-level throttling	Notes
GetInstanceAccess	4	1	No	Limit is per account.
ListAliases	30	30	No	Limit is per account.
ListBuilds	30	30	No	Limit is per account.
ListContainerFleets	30	30	No	Limit is per account.
ListCompute	30	30	No	Limit is per account.
ListContainerGroup Definitions	30	30	No	Limit is per account.
ListContainerGroup DefinitionVersions	30	30	No	Limit is per account.
ListFleets	30	30	No	Limit is per account.
ListFleetDeployments	30	30	No	Limit is per account.
ListGameServerGroups	30	30	No	Limit is per account.
ListGameServers	30	30	No	Limit is per account.
ListLocations	30	30	No	Limit is per account.
ListScripts	30	30	No	Limit is per account.
ListTagsForResource	30	30	No	Limit is per account.
PutScalingPolicy	20	20	No	Limit is per account.
RegisterCompute	20	20	No	Limit is per account.
RegisterGameServer	40	20	No	Limit is per account.

API action	Burst limit	Rate limit	Resource-level throttling	Notes
RequestUploadCredentials	4	1	No	Limit is per account.
ResolveAlias	30	30	No	Limit is per account.
ResumeGameServerGroup	1	0.1	No	Limit is per account.
SearchGameSessions	30	30	No	Limit is per account.
StartFleetActions	20	20	No	Limit is per account.
StartGameSessionPlacement	75	75	No	Limit is per account.
StartMatchBackfill	70	70	No	Limit is per account.
StartMatchmaking	100	100	No	Limit is per account.
StopFleetActions	20	20	No	Limit is per account.
StopGameSessionPlacement	20	20	No	Limit is per account.
StopMatchmaking	100	100	No	Limit is per account.
SuspendGameServerGroup	1	0.1	No	Limit is per account.
TagResource	30	30	No	Limit is per account.
TerminateGameSession	20	20	No	Limit is per account.
UntagResource	30	30	No	Limit is per account.

API action	Burst limit	Rate limit	Resource-level throttling	Notes
UpdateAlias	20	20	No	Limit is per account.
UpdateBuild	20	20	No	Limit is per account.
UpdateContainerGroupDefinition	20	20	No	Limit is per account.
UpdateContainerFleet	20	20	No	Limit is per account.
UpdateFleetAttributes	20	20	No	Limit is per account.
UpdateFleetCapacity	20	20	No	Limit is per account.
UpdateFleetPortSettings	20	20	No	Limit is per account.
UpdateGameServer	40	20	No	Limit is per account.
UpdateGameServerGroup	20	20	No	Limit is per account.
UpdateGameSession	20	20	No	Limit is per account.
UpdateGameSessionQueue	20	20	No	Limit is per account.
UpdateMatchmakingConfiguration	20	20	No	Limit is per account.
UpdateRuntimeConfiguration	20	20	No	Limit is per account.

API action	Burst limit	Rate limit	Resource-level throttling	Notes
UpdateScript	20	20	No	Limit is per account.
ValidateMatchmakingRuleSet	30	30	No	Limit is per account.

Resource-level throttling

Some API operations are subject to resource-level throttling to prevent database hot key issues. While increasing API-level limits for these operations, they may still be throttled at the resource level.

The following API operations are subject to resource-level throttling:

- DescribeGameSessionPlacement
- DescribeGameSessions
- DescribeMatchmaking
- DescribePlayerSessions

Resource-level throttling is enforced using specific throttle keys that include resource identifiers. For example, DescribeGameSessionPlacement uses the key "Operation:GameLift/DescribeGameSessionPlacement,aws-account-placement-id:" which enforces throttling per account and placement ID combination. This prevents customers from overwhelming the system with requests for specific resources.

Amazon GameLift Servers SDK API limits

The following table lists the default rate limits for Amazon GameLift Servers SDK API operations. These limits optimize performance and prevent resource contention in game server environments. Understanding these limits is important for efficient server-side integration with Amazon GameLift Servers.

Server SDK limits are enforced using specific throttling keys that include account and process identifiers. Some limits are per-player to prevent abuse from individual players.

Note

This table is subject to change. The authoritative source for current limits is the internal SDC configuration. Contact AWS Support for the most up-to-date information if needed.

Server SDK API limits reference

API action	Burst limit	Rate limit	Throttling key	Notes
AcceptPlayerSession	3	1	aws-account-and-process-id-and-player-id	Limit is per player and per process. Should never need a limit increase as it's per-player.
RemovePlayerSession	3	1	aws-account-and-process-id-and-player-id	Limit is per player and per process. Should never need a limit increase as it's per-player.
ActivateGameSession	3	1	aws-account-and-process-id	Limit is per server process. Should never need a limit increase as it's per-process.
ActivateHostProcessV2	3	1	aws-account-and-process-id	Limit is per server process. Should never need a limit increase as it's per-process.
StartMatchBackfill	3	1	aws-account-and-process-id	Limit is per server process. Should never need a limit increase as it's per-process.
DescribePlayerSessions	10	5	aws-account-and-process-id	Limit is per server process. Should never need a limit increase as it's per-process.

API action	Burst limit	Rate limit	Throttling key	Notes
GetComputeCertificate	3	0.1	aws-account-and-process-id	Limit is per server process. Should never need a limit increase as it's per-process.
GetCustomRoleCredentials	3	0.1	aws-account-and-process-id	Limit is per server process. Should never need a limit increase as it's per-process.
InitSDK	3	0.1	aws-account-and-process-id	Limit is per server process. Should never need a limit increase as it's per-process.
RefreshHostProcess	3	0.1	aws-account-and-process-id	Limit is per server process. Should never need a limit increase as it's per-process.
StopMatchmaking	3	1	aws-account-and-process-id	Limit is per server process. Should never need a limit increase as it's per-process.
ProcessEnding	3	1	aws-account-and-process-id	Limit is per server process. Should never need a limit increase as it's per-process.
ProcessReady	3	1	aws-account-and-process-id	Limit is per server process. Should never need a limit increase as it's per-process.
UpdateGameSession	3	1	aws-account-and-process-id	Limit is per server process. Should never need a limit increase as it's per-process.
UpdateHostProcess	3	1	aws-account-and-process-id	Limit is per server process. Should never need a limit increase as it's per-process.

API action	Burst limit	Rate limit	Throttling key	Notes
GetGameSessionId	10	5	aws-account-and-process-id	Limit is per server process. Should never need a limit increase as it's per-process.
GetTerminationTime	3	1	aws-account-and-process-id	Limit is per server process. Should never need a limit increase as it's per-process.
UpdatePlayerSessionCreationPolicy	3	1	aws-account-and-process-id	Limit is per server process. Should never need a limit increase as it's per-process.

Note

These are the default limits. If you're experiencing throttling on these APIs, it may be due to account-level throttling. To request a limit increase, contact AWS Support.

Per-process and per-player limits are designed to accommodate normal usage patterns and should rarely need adjustment. If you're experiencing throttling with these APIs, review your implementation for potential optimization opportunities before requesting limit increases.

Account-level throttling may also apply through these additional rules:

- SdkWebSocket/AccountLevelHighUsage,aws-account
- SdkWebSocket/AccountLevelThrottle,aws-account

Amazon GameLift Servers UDP ping beacons

UDP ping beacons provide a way to measure network latency between player devices and Amazon GameLift Servers hosting locations. With ping beacons, you can collect accurate latency data to make informed decisions about game server placement and improve player matchmaking based on latency requirements.

How UDP ping beacons work

Amazon GameLift Servers provides fixed UDP endpoints (ping beacons) in each hosting location where you can deploy game servers. Because most game servers communicate using UDP, measuring latency with UDP ping beacons provides more accurate results than using ICMP pings. Network devices often handle ICMP packets differently from UDP packets, which can lead to latency measurements that don't reflect the true performance your players will experience.

With UDP ping beacons, your game client can send UDP messages to these endpoints and receive asynchronous responses, giving you latency measurements that better represent actual game traffic conditions between a player's device and potential hosting locations. The endpoints are permanent and remain available as long as Amazon GameLift Servers supports game hosting in that location.

Common use cases for UDP ping beacons

You can use UDP ping beacons in several ways to optimize your game's networking experience.

Choosing optimal hosting locations

Collect latency data across different geographic regions to identify the best primary and backup locations for hosting game servers for your player base.

Placing game sessions based on player latency

Include player latency data when requesting new game sessions to help pick locations that provide the lowest latency experience.

Optimizing matchmaking based on latency

Provide player latency data when requesting matchmaking to help match players with similar latency profiles and place game sessions in optimal locations for matched players.

Note

When creating matchmaking requests, you should not provide latency information to locations where you don't have fleets. If you do, Amazon GameLift Servers might try to place the game session in locations where there's no fleet capacity, resulting in matchmaking request failures.

Getting beacon endpoints

To retrieve ping beacon domain and port information for Amazon GameLift Servers locations, use the [ListLocations](#) API operation. The set of locations returned by this API depends on the AWS Region you specify when calling it (or your default Region if you don't specify one).

When you call from:

- **A home Region of a fleet that supports multi-locations:** API returns information for *all* hosting locations
- **A home Region of a fleet that supports a single location:** API returns information for that location

Note that if you call this API using a location that can only be a remote location in a multi-location fleet, the API will return an error because that type of location doesn't have a service endpoint.

Consult the table of supported locations in [Supported AWS locations](#) to identify home Regions that support single and multi-location fleets.

Example

```
aws gamelift list-locations --region ap-northeast-2
```

This AWS Region supports multi-location fleets, therefore multiple locations will be returned. Here is an example of one of the return values:

```
[...]
{
  "LocationName": "ap-northeast-1",
  "PingBeacon": {
    "UDPEndpoint": {
      "Domain": "gamelift-ping.ap-northeast-1.api.aws",
      "Port": 7770
    }
  }
}
```

⚠ Important

Cache the ping beacon information rather than calling `ListLocations` before each latency measurement. The domain and port information are static and the API isn't designed for high-volume requests.

Implementing latency measurements

Follow these best practices when implementing latency measurements using UDP ping beacons:

1. Store ping beacon information using one of these approaches:

- Hardcode the endpoints in your game client.
- Cache the information in your game backend.
- Implement a periodic update mechanism (daily/weekly) to refresh the information.

2. Send UDP ping messages:

- Put whatever you want in the message body, as long as it is not empty, and you keep messages under the maximum size of 300 bytes.
- Observe the following rate limits for each location:
 - 3 transactions per second (TPS) per unique sender IP address and port combination
 - 1000 TPS per unique sender IP address

3. Calculate latency:

- Send multiple pings to each location to calculate an average latency.
- Consider sending concurrent pings to multiple locations for faster results.
- Use retry logic as needed to send new packets for any packet that wasn't returned within a short time (typically 1 - 3 seconds), since UDP does not have 100% guaranteed delivery.
- Calculate the time difference between sending a message and receiving the response.
- If a large portion of UDP pings to a location consistently do not get a response back, calculate latency using ICMP pings to our standard [Amazon GameLift Servers service endpoints](#) as a fallback.

Tip

We recommend that you include port 7770 wherever you document the list of ports that your players must have open on their local network. This is another reason why you should have a fallback for measuring latency (using ICMP, for example) in case this port is blocked.

Code examples

Here are some simple examples showing how to send UDP pings and calculate latency.

C++

```
#include <iostream>
#include <cstring>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#include <chrono>

int main() {
    // Replace with Amazon GameLift Servers UDP ping beacon domain for your desired
    location
    const char* domain = "gamelift-ping.ap-south-1.api.aws";
    const int port = 7770;
    const char* message = "Ping";           // Your message
    const int num_pings = 3;                // Number of pings to send

    // Create socket
    int sock = socket(AF_INET, SOCK_DGRAM, 0);
    if (sock < 0) {
        std::cerr << "Error creating socket" << std::endl;
        return 1;
    }

    // Resolve domain name to IP address
    struct hostent* host = gethostbyname(domain);
    if (host == nullptr) {
        std::cerr << "Error resolving hostname" << std::endl;
        close(sock);
    }
}
```

```
        return 1;
    }

    // Set up the server address structure
    struct sockaddr_in server_addr;
    std::memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(port);
    std::memcpy(&server_addr.sin_addr, host->h_addr, host->h_length);

    // Set socket timeout
    struct timeval tv;
    tv.tv_sec = 1; // 1 second timeout
    tv.tv_usec = 0;
    if (setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv)) < 0) {
        std::cerr << "Error setting socket timeout" << std::endl;
        close(sock);
        return 1;
    }

    double total_latency = 0;
    int successful_pings = 0;

    for (int i = 0; i < num_pings; ++i) {
        auto start = std::chrono::high_resolution_clock::now();

        // Send the message
        ssize_t bytes_sent = sendto(sock, message, std::strlen(message), 0,
                                     (struct sockaddr*)&server_addr,
                                     sizeof(server_addr));

        if (bytes_sent < 0) {
            std::cerr << "Error sending message" << std::endl;
            continue;
        }

        // Receive response
        char buffer[1024];
        socklen_t server_addr_len = sizeof(server_addr);
        ssize_t bytes_received = recvfrom(sock, buffer, sizeof(buffer), 0,
                                           (struct sockaddr*)&server_addr,
                                           &server_addr_len);

        auto end = std::chrono::high_resolution_clock::now();
```

```

        if (bytes_received < 0) {
            std::cerr << "Error receiving response or timeout" << std::endl;
        } else {
            std::chrono::duration<double, std::milli> latency = end - start;
            total_latency += latency.count();
            successful_pings++;

            std::cout << "Received response, latency: " << latency.count() << " ms"
<< std::endl;
        }

        // Wait a bit before next ping
        usleep(1000000); // 1s
    }

    // Close the socket
    close(sock);

    if (successful_pings > 0) {
        double avg_latency = total_latency / successful_pings;
        std::cout << "Average latency: " << avg_latency << " ms" << std::endl;
    } else {
        std::cout << "No successful pings" << std::endl;
    }

    return 0;
}

```

C#

```

using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Diagnostics;
using System.Threading.Tasks;

class UdpLatencyTest
{
    static async Task Main()
    {

```

```
// Replace with Amazon GameLift Servers UDP ping beacon domain for your
desired location
string domain = "gamelift-ping.ap-south-1.api.aws";
int port = 7770;
string message = "Ping";           // Your message
int numPings = 3;                  // Number of pings to send
int timeoutMs = 1000;              // Timeout in milliseconds

await MeasureLatency(domain, port, message, numPings, timeoutMs);
}

static async Task MeasureLatency(string domain, int port, string message, int
numPings, int timeoutMs)
{
    using (var udpClient = new UdpClient())
    {
        try
        {
            // Resolve domain name to IP address
            IPAddress[] addresses = await Dns.GetHostAddressesAsync(domain);
            if (addresses.Length == 0)
            {
                Console.WriteLine("Could not resolve domain name.");
                return;
            }

            IPEndPoint endPoint = new IPEndPoint(addresses[0], port);
            byte[] messageBytes = Encoding.UTF8.GetBytes(message);

            // Set receive timeout
            udpClient.Client.ReceiveTimeout = timeoutMs;

            double totalLatency = 0;
            int successfulPings = 0;
            var stopwatch = new Stopwatch();

            for (int i = 0; i < numPings; i++)
            {
                try
                {
                    stopwatch.Restart();

                    // Send message
```

```

        await udpClient.SendAsync(messageBytes, messageBytes.Length,
endPoint);

        // Wait for response
        UdpReceiveResult result = await
ReceiveWithTimeoutAsync(udpClient, timeoutMs);

        stopwatch.Stop();
        double latency = stopwatch.Elapsed.TotalMilliseconds;

        totalLatency += latency;
        successfulPings++;

        string response = Encoding.UTF8.GetString(result.Buffer);
        Console.WriteLine($"Ping {i + 1}: {latency:F2}ms - Response:
{response}");
    }
    catch (SocketException ex)
    {
        Console.WriteLine($"Ping {i + 1}: Failed - {ex.Message}");
    }
    catch (TimeoutException)
    {
        Console.WriteLine($"Ping {i + 1}: Timeout");
    }

    // Wait before next ping
    await Task.Delay(1000); // 1s between pings
}

if (successfulPings > 0)
{
    double averageLatency = totalLatency / successfulPings;
    Console.WriteLine($"Summary:");
    Console.WriteLine($"Successful pings: {successfulPings}/
{numPings}");
    Console.WriteLine($"Average latency: {averageLatency:F2}ms");
}
else
{
    Console.WriteLine("\nNo successful pings");
}
}
catch (Exception ex)

```

```

        {
            Console.WriteLine($"Error: {ex.Message}");
        }
    }

    static async Task<UdpReceiveResult> ReceiveWithTimeoutAsync(UdpClient client,
int timeoutMs)
    {
        using var cts = new System.Threading.CancellationTokenSource(timeoutMs);
        try
        {
            return await client.ReceiveAsync().WaitAsync(cts.Token);
        }
        catch (OperationCanceledException)
        {
            throw new TimeoutException("Receive operation timed out");
        }
    }
}

```

Python

```

import socket
import time
import statistics
from datetime import datetime

def udp_ping(host, port, timeout=2):
    """
    Send a UDP ping and return the round trip time in milliseconds.
    Returns None if timeout occurs.
    """
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.settimeout(timeout)
    message = b'ping'

    try:
        # Resolve hostname first
        try:
            socket.gethostbyname(host)
        except socket.gaierror as e:
            print(f"Could not resolve hostname: {e}")
    
```



```

        return None

    start_time = time.time()
    sock.sendto(message, (host, port))

    # Wait for response
    data, server = sock.recvfrom(1024)
    end_time = time.time()

    # Calculate round trip time in milliseconds
    rtt = (end_time - start_time) * 1000
    return rtt

except socket.timeout:
    print(f"Request timed out")
    return None
except Exception as e:
    print(f"Error: {type(e).__name__}: {e}")
    return None
finally:
    sock.close()

def main():
    # Replace with Amazon GameLift Servers UDP ping beacon domain for your desired
    location
    host = "gamelift-ping.ap-south-1.api.aws"
    port = 7770
    num_pings = 3
    latencies = []

    print(f"\nPinging {host}:{port} {num_pings} times...")
    print(f"Start time: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")

    for i in range(num_pings):
        print(f"Ping {i+1}:")
        rtt = udp_ping(host, port)

        if rtt is not None:
            print(f"Response from {host}: time={rtt:.2f}ms")
            latencies.append(rtt)

    # Wait 1 second between pings
    if i < num_pings - 1:
        time.sleep(1)

```

```
print()

# Calculate and display statistics
print("-" * 50)
print(f"Ping statistics for {host}:")
print(f"    Packets: Sent = {num_pings}, Received = {len(latencies)}, "
      f"    Lost = {num_pings - len(latencies)} "
      f"    f"({((num_pings - len(latencies)) / num_pings * 100):.1f}% loss)")

if latencies:
    print("\nRound-trip latency statistics:")
    print(f"    Minimum = {min(latencies):.2f}ms")
    print(f"    Maximum = {max(latencies):.2f}ms")
    print(f"    Average = {statistics.mean(latencies):.2f}ms")

print(f"\nEnd time: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")

if __name__ == "__main__":
    main()
```

FAQs on supported protocols

The Amazon GameLift Servers is adding SDK support for the [Smithy RPC v2 CBOR](#) protocol in addition to the existing [AWS JSON](#) protocol. The Smithy RPC v2 CBOR protocol is more performant than AWS JSON for most workloads. AWS SDKs will prioritize the protocol that is the most performant for each language.

What are the AWS JSON and Smithy RPC v2 CBOR protocols, and how do they differ?

A wire protocol acts as a mediator between a client and the service. The AWS SDK handles the serialization (converting an object to the wire format) and deserialization (converting the wire format back to an object) between the client and server transparently for both protocols.

AWS JSON - JSON is one of the most widely used and accepted data formats for communication between systems. This service uses AWS JSON as a medium to communicate between an AWS SDK client and the server. An HTTP request for a service API operation accepts input in the form of JSON, the operation is executed, and the response is returned to the SDK client in the form of JSON.

Smithy RPC v2 CBOR - Smithy RPC v2 CBOR (Concise Binary Object Representation) is a protocol developed by Amazon for more efficient data transmission. It is a data format designed to keep message sizes compact and to allow the format to evolve without requiring version negotiation between systems. The key difference from AWS JSON is that AWS JSON uses human-readable text (like `{"name": "John"}`), while Smithy RPC v2 CBOR uses a binary format (sequences of 1s and 0s). This binary approach creates smaller data packages that travel faster over the network and reduces serialization time since computers process binary data directly.

How do I get started with the Smithy RPC v2 CBOR protocol?

To get started, use the latest AWS SDK/CLI version to use the Smithy RPC v2 CBOR protocol for this service. The SDK will automatically use the Smithy RPC v2 CBOR protocol when available. Upgrade your AWS SDK to the specified version or any subsequent version.

Starting 2026-04-16, the following AWS SDKs will use the Smithy RPC v2 CBOR protocol, while the other SDKs will continue using the AWS JSON protocol:

- AWS SDK for C++
- AWS SDK for Go V2
- AWS SDK for Java 2.x
- AWS SDK for .NET v4
- AWS Tools for PowerShell v5
- AWS SDK for Swift
- AWS SDK for Kotlin

What are the risks of enabling the Smithy RPC v2 CBOR protocol?

The added support of Smithy RPC v2 CBOR in the SDKs is expected to be fully transparent and effortless for customers. However, if you are using a custom implementation of the AWS SDK, or a combination of custom clients and the AWS SDK, that depends on AWS JSON-specific serialization behavior, it may be incompatible with the Smithy RPC v2 CBOR protocol.

What if my solution does not support Smithy RPC v2 CBOR?

To use the AWS JSON protocol, downgrade your SDK to a version released before 2026-04-16.

What AWS Regions support the Smithy RPC v2 CBOR protocol?

The Smithy RPC v2 CBOR protocol is supported in all [AWS Regions](#) where this service is available.

What latency improvements can I expect when upgrading to SDK versions that support Smithy RPC v2 CBOR?

Smithy RPC v2 CBOR provides up to 13% lower latency and up to 15% reduction in payload size compared to AWS JSON. The binary encoding eliminates the overhead of JSON text parsing and string escaping, which reduces both network transfer time and serialization cost.

Will the AWS JSON protocol be deprecated?

AWS JSON protocol will continue to be supported through older versions of SDKs. Also, some language-specific SDKs will continue using AWS JSON as it is more performant than Smithy RPC

v2 CBOR due to the available JSON implementation for the language being faster than its CBOR implementation.

Where can I find more information about Smithy RPC v2 CBOR?

You can find more information about the AWS JSON protocol at [AWS JSON 1.0 protocol](#) and [AWS JSON 1.1 protocol](#) in the Smithy documentation, and about Smithy RPC v2 CBOR at [Smithy RPC v2 CBOR protocol](#) in the Smithy documentation.

Amazon GameLift Servers release notes

The Amazon GameLift Servers release notes provide details about new features, updates, and fixes related to the service.

SDK versions

The following tables list all Amazon GameLift Servers releases with SDK version information. There is no requirement to use comparable SDKs for your game server and client integrations. However, earlier versions of one SDK may not fully support the latest features in another.

Amazon GameLift Servers SDKs and plugins are open source. See [Get Amazon GameLift Servers development tools](#). To get the latest versions, see the [Amazon GameLift Servers GitHub organization](#).

Current version

GameTime
Client
Unity
Unreal
Signal
API
+
.)
2.3
ter

1. **Time**
 2. **Content**
 3. **Unit**
 4. **Practical**
 5. **Practical**
 6. **(1st)**
 7. **+**
 8. **Practical**

[illegible]

2025-2

Letter

195-0

16-10005-1

 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10
 2018.1.10

	Time
	ent
	l
	l
	l
	(y)
	+
	n)real
	<u>2005-1</u>
	-
	ter
	<u>2005-1</u>
	-
	ter
	<u>2005-1</u>
	-
	ter
	<u>2005-1</u>
	-
	ter
	<u>2003-1</u>
	-
	ter
	<u>2003-0</u>
	-
	ter

	Time	
	ent	
	l	
	l	
	(y)	
	+	
	n)real	
	04-2	
	-	
	ter	
	04-1	
	-	
	ter	
	07-2	
	-	
	ter	
	05-2	
	-	
	ter	
	06-1	
	-	
	ter	
	05-2	
	-	
	ter	
	04-2	
	-	
	ter	

[illegible]

	Time	Sent	Unit
	Initial	(City)	+ J)real
	<u>12-18-2</u>	-	ter
	<u>12-18-1</u>	-	ter
	<u>12-19-0</u>	-	ter
	<u>12-50-2</u>	-	ter
	<u>12-50-2</u>	-	ter
	<u>12-50-1</u>	-	ter
	<u>12-10-1</u>	-	ter

	Time
	Kent
	Unit
	Unl
	Unl
	(C)
	+ In)real
	VOC-2
	- ter
	VOC-1
	- ter
	VOC-0
	- ter
	VOC-1
	- ter
	VOC-1
	- ter
	VOC-2
	- ter
	VOC-0
	- ter

Flight Number	Time Unit
3000-0	Real
3001-2	Real
3003-0	Real
3002-0	Real
3012-1	Real
2109-2	Real
2106-1	Real

5. **Initial time**
6. **Initial**
7. **Initial**
8. **Initial**
9. **Initial**
10. **Initial**
11. **Initial**
12. **Initial**
13. **Initial**
14. **Initial**
15. **Initial**
16. **Initial**
17. **Initial**
18. **Initial**
19. **Initial**
20. **Initial**
21. **Initial**
22. **Initial**
23. **Initial**
24. **Initial**
25. **Initial**
26. **Initial**
27. **Initial**
28. **Initial**
29. **Initial**
30. **Initial**
31. **Initial**
32. **Initial**
33. **Initial**
34. **Initial**
35. **Initial**
36. **Initial**
37. **Initial**
38. **Initial**
39. **Initial**
40. **Initial**
41. **Initial**
42. **Initial**
43. **Initial**
44. **Initial**
45. **Initial**
46. **Initial**
47. **Initial**
48. **Initial**
49. **Initial**
50. **Initial**
51. **Initial**
52. **Initial**
53. **Initial**
54. **Initial**
55. **Initial**
56. **Initial**
57. **Initial**
58. **Initial**
59. **Initial**
60. **Initial**
61. **Initial**
62. **Initial**
63. **Initial**
64. **Initial**
65. **Initial**
66. **Initial**
67. **Initial**
68. **Initial**
69. **Initial**
70. **Initial**
71. **Initial**
72. **Initial**
73. **Initial**
74. **Initial**
75. **Initial**
76. **Initial**
77. **Initial**
78. **Initial**
79. **Initial**
80. **Initial**
81. **Initial**
82. **Initial**
83. **Initial**
84. **Initial**
85. **Initial**
86. **Initial**
87. **Initial**
88. **Initial**
89. **Initial**
90. **Initial**
91. **Initial**
92. **Initial**
93. **Initial**
94. **Initial**
95. **Initial**
96. **Initial**
97. **Initial**
98. **Initial**
99. **Initial**
100. **Initial**

[v2.1.05-1](#)
later

[v2.1.02-1](#)
later

[v2.1.02-0](#)
later

[v2.1.09-0](#)
later

[v2.1.08-1](#)
later

[v2.1.05-1](#)
later

[v2.1.04-1](#)
later

[illegible]

Release notes

The following release notes are in chronological order, with the latest updates listed first. Amazon GameLift Servers was first released in 2016. For release notes dated earlier than those listed here, see the release date links in [SDK versions](#).

June 30, 2026: Amazon GameLift Servers releases C# and Unity client SDKs for game clients

Amazon GameLift Servers now provides client SDKs for C# and Unity that integrate game clients with player gateway and UDP ping beacons.

Learn more:

- [DDoS protection with Amazon GameLift Servers](#), *Amazon GameLift Servers Developer Guide*
- [Integrate player gateway into a game](#), *Amazon GameLift Servers Developer Guide*
- [Client SDK for C#](#)
- [Client SDK for Unity](#)

June 18, 2026: Amazon GameLift Servers container fleet improvements: Linux capabilities and network discovery

This release adds two improvements for container fleets: support for customizing Linux capabilities for containers, and a new server SDK action for discovering container network information.

Customize Linux capabilities

You can now specify additional Linux capabilities in a container group definition, giving you finer control over the default Docker capabilities. This is useful for game server or support containers that require capabilities beyond the default set, such as `NET_RAW` for custom networking or `SYS_PTRACE` for debugging. You can add Linux capabilities when you create a container group definition with `CreateContainerGroupDefinition`, or update an existing one with `UpdateContainerGroupDefinition`. This feature is available through the Amazon GameLift Servers console, AWS CLI, AWS SDK, and AWS CloudFormation.

Discover container network information

Game servers running on container fleets can now call the new `ListContainersNetworkInfo()` server SDK action to retrieve network information (including the name, ID, local IP address, and container group type) for all containers running on the same instance. With this information, containers can discover one another and establish network communication across different container group definitions on the same instance. For example, a container in the game server container group can now emit metrics to a container in the per-instance container group over the

instance's local network. This action is available in the Amazon GameLift Servers server SDK 5.5 or later for Go, C++, and C#, and in the Amazon GameLift Servers plugins for Unreal Engine and Unity.

Learn more:

- [CreateContainerGroupDefinition](#), *Amazon GameLift Servers API Reference*
- [UpdateContainerGroupDefinition](#), *Amazon GameLift Servers API Reference*
- [Go server SDK for Amazon GameLift Servers -- Actions](#), *Amazon GameLift Servers Developer Guide*
- [C# server SDK 5.x for Amazon GameLift Servers -- Actions](#), *Amazon GameLift Servers Developer Guide*
- [C++ server SDK 5.x for Amazon GameLift Servers -- Actions](#), *Amazon GameLift Servers Developer Guide*
- [C++ \(Unreal\) server SDK 5.x for Amazon GameLift Servers -- Actions](#), *Amazon GameLift Servers Developer Guide*

June 15, 2026: Amazon GameLift Servers introduces free network bandwidth

Starting today, Amazon GameLift Servers provides network bandwidth in and out of AWS at no additional charge for all instance types from generation 6 and later, including On-Demand and Spot, with no commitment required. You now pay only for your Amazon GameLift Servers instance hours; all network bandwidth is free.

Multiplayer game servers generate continuous network traffic to connected players, making bandwidth one of the most unpredictable cost components for game studio customers. With free network bandwidth included, Amazon GameLift Servers eliminates this cost, giving you the simplicity of bare-metal hosting with the global reach of AWS.

Free network bandwidth applies with no enrollment, pricing agreement, or configuration change required. Existing customers on eligible fleets receive the benefit immediately.

April 29, 2026: Amazon GameLift Servers adds new support for viewing container port mappings

Amazon GameLift Servers has launched a new public API for container fleets, `DescribeContainerGroupPortMappings`. This operation retrieves the port mappings for all

containers in a container group, listing how each container port maps to a connection port on the fleet instance. It supports retrieving port mappings for both game server and per-instance container groups.

This new feature addresses the need to discover port mappings for containers beyond the game server port. Previously, customers were required to remotely access their compute to retrieve all other port mappings. The new API is available through the Amazon GameLift Servers console, AWS CLI, and AWS SDK.

Learn more:

- [View container port mappings](#), *Amazon GameLift Servers Developer Guide*
- [DescribeContainerGroupPortMappings](#), *Amazon GameLift Servers API Reference*

April 2, 2026: Amazon GameLift Servers adds compute name to game session data, new event fields, and console improvements

This release makes it simpler to onboard games and integrate with Amazon GameLift Servers, based on customer feedback. These updates help customers get started and manage game sessions within fleets more easily.

Amazon GameLift Servers now returns a `ComputeName` field in game session API responses. You can use this field to identify which compute is hosting a game session, without cross-referencing IP addresses. This field is available for managed EC2 fleets, managed container fleets, and Anywhere fleets. The `ComputeName` and `GameSessionLocation` fields are also now included in [PlacementFulfilled](#) and [MatchmakingSucceeded](#) events.

Amazon GameLift Servers API Reference links:

- [GameSession](#)

This release also includes several Amazon GameLift Servers console improvements. With these updates, you can now:

- **Upload builds directly** – In addition to the Amazon S3 import method, you can select a build file and upload it directly to Amazon GameLift Servers from the console.
- **Clone fleet configurations** – Clone an existing managed EC2 fleet configuration to create a new fleet.

- **Connect to instances and containers** – Use SSM to connect directly from game session, player session, and compute details pages, including new access commands for containers.
- **View compute and instance data** – Game session tables now display ComputeName and InstanceId columns for managed EC2 and container fleets.
- **View container fleet compute details** – Container fleets now include compute tables and compute detail views.
- **View game session log locations** – The game session details page now displays Amazon S3 URIs and Amazon CloudWatch log group ARNs for container fleets.
- **Refresh data** – Refresh buttons are now available on all list pages and detail views.

Learn more:

- [Amazon GameLift Servers console](#)
- [PlacementFulfilled](#), *Amazon GameLift Servers Developer Guide*
- [MatchmakingSucceeded](#), *Amazon GameLift Servers FlexMatch Developer Guide*
- [Connecting to a container fleet instance](#), *Amazon GameLift Servers Developer Guide*

March 27, 2026: Amazon GameLift Servers now supports next-generation EC2 instances in additional AWS Regions

Amazon GameLift Servers now supports its existing selection of Amazon EC2 instances in additional AWS Regions, giving you more flexibility to host game servers closer to your players. This expansion covers instances across the following families:

- [General purpose](#) (M-series)
- [Compute optimized](#) (C-series)
- [Memory optimized](#) (R-series)

These instance types are now available for use with Amazon GameLift Servers managed EC2 fleets and managed container fleets. No changes to your fleet configuration are required – simply deploy fleets in the newly supported Regions using the same instance types and settings you use today. They are available in all AWS Regions that are supported by Amazon GameLift Servers, with the exception of the AWS China Regions.

Learn more:

- [Amazon GameLift Servers Instance Pricing](#)
- [Amazon GameLift Servers service locations](#), *Amazon GameLift Servers Developer Guide*
- [CreateFleet](#), *Amazon GameLift Servers API Reference*
- [CreateContainerFleet](#), *Amazon GameLift Servers API Reference*

March 4, 2026: Amazon GameLift Servers now offers DDoS protection for Linux-based EC2 and Container Fleets on SDKv5

Amazon GameLift Servers now offers distributed-denial-of-service (DDoS) protection for Linux-based EC2 and Container Fleets on SDKv5. The player gateway proxy relay network is co-located with your game server instances, providing traffic validation, per-player rate limiting and game server IP address obfuscation all with negligible added latency and no additional cost. This feature is initially available in the following regions: US East (N. Virginia), US West (Oregon), Europe (Frankfurt), Europe (Ireland), Asia Pacific (Sydney), Asia Pacific (Tokyo), Asia Pacific (Seoul). For more information about service locations, see [Amazon GameLift Servers service locations](#).

Learn more:

- [Blog post: Introducing Amazon GameLift Servers DDoS protection](#)
- [DDoS protection with Amazon GameLift Servers](#), *Amazon GameLift Servers Developer Guide*
- [How player gateway works](#), *Amazon GameLift Servers Developer Guide*
- [Integrate player gateway into a game](#), *Amazon GameLift Servers Developer Guide*
- [Client SDK for C++](#)
- [Client SDK for Unreal](#). Access requires membership in the Epic Games organization on GitHub. See [Unreal Engine on GitHub](#) for details.

February 24, 2026: Amazon GameLift Servers plugins now support Unity 6.3 and Unreal Engine 5.7**Updated plugin versions:**

- [Plugin for Unity](#), version 3.2.1 (includes the server SDK for Unity, version 5.4.0)
- [Plugin for Unreal](#), version 3.1.1 (includes the server SDK for Unreal, version 5.4.0)

Amazon GameLift Servers releases new versions of the plugin and server SDK for Unreal Engine and Unity. These latest versions include the following updates:

- They now support use with Unreal Engine 5.7 and Unity 6.3.

Learn more:

- [Integrating games with the Amazon GameLift Servers plugin for Unreal Engine](#), *Amazon GameLift Servers Developer Guide*
- [Integrate Amazon GameLift Servers into an Unreal Engine project](#), *Amazon GameLift Servers Developer Guide*
- [Integrating games with the Amazon GameLift Servers plugin for Unity](#), *Amazon GameLift Servers Developer Guide*
- [Integrate Amazon GameLift Servers into a Unity Engine project](#), *Amazon GameLift Servers Developer Guide*

January 29, 2026: Amazon GameLift Servers now supports scaling Fleets to/from zero instances

Amazon GameLift Servers now supports automatic scaling to and from zero instances based on game session activity. Fleets scale down to zero following a defined period of no game session activity and scale up from zero when game sessions are requested, providing an option for cost optimization.

Amazon GameLift Servers API Reference links:

- [AWS SDK action UpdateFleetCapacity](#)

Learn more:

- [Manage Scaling an Amazon GameLift Servers Fleet To/From Zero](#)

January 22, 2026: Amazon GameLift Servers Realtime now supports Node.js 24.x on Amazon Linux 2023

Amazon GameLift Servers Realtime now supports Node.js 24.x on Amazon Linux 2023. This update provides access to the latest Node.js features and security improvements while running on the modern Amazon Linux 2023 operating system.

You can now also include an optional install script when creating a Realtime fleet, allowing the installation of software during instance startup for your specific requirements. To use the new runtime version, specify Node.js 24.x when creating Realtime scripts.

Amazon GameLift Servers API Reference links:

- [AWS SDK action CreateScript](#)

Learn more:

- [Node.js runtime versions](#)
- [Add an install script](#)

December 9, 2025: Amazon GameLift Servers now provides AI-powered assistance in the web console

Amazon GameLift Servers now provides AI-powered assistance in the web console, leveraging Amazon Q Developer to provide tailored guidance for game developers. This new feature integrates specialized Amazon GameLift Servers knowledge to help customers navigate complex workflows, troubleshoot issues, and optimize their game server deployments more efficiently.

This new feature provides AI-assisted recommendations for game server integration, fleet configuration, and performance optimization directly within the Amazon GameLift Servers console. This enhancement aims to streamline decision making processes, reduce troubleshooting time, and improve overall resource utilization, leading to cost savings and better player experiences.

Learn more:

- [Amazon GameLift Servers console](#)

November 6, 2025: Amazon GameLift Servers now supports game builds that use the Windows Server 2022 operating system

Amazon GameLift Servers customers can now use the Windows Server 2022 operating system to host their game servers. Windows Server 2022 offers several improvements over Windows Server 2016 including security. This operating system is available in all AWS Regions with the exception of the China Regions.

Customers can use the newer Windows operating systems and continue to receive critical security updates when support ends for Windows Server 2016 in January 2027. Support for Windows Server 2022 continues through October 13, 2031.

Amazon GameLift Servers API Reference links:

- [AWS SDK action `CreateBuild`](#)
- [CLI command `upload-build`](#)
- [CLI command `create-build`](#)

October 28, 2025: Amazon GameLift Servers adds OpenTelemetry (OTel) metrics collector to all server SDKs and plugins

Amazon GameLift Servers now includes built-in OpenTelemetry (OTel) metrics collection in all server SDKs and plugins, providing standardized observability for game server performance monitoring. This integration brings industry-standard telemetry capabilities directly to your game servers, enabling comprehensive monitoring and troubleshooting without requiring additional instrumentation or configuration.

With OpenTelemetry integration, you can now:

- Automatically capture essential game server metrics, logs, and traces using standardized protocols
- Integrate seamlessly with existing monitoring platforms and observability tools
- Monitor game server performance and health in real-time across all fleet instances
- Troubleshoot issues faster with telemetry data that works across different monitoring solutions

OpenTelemetry is an open-source observability framework that provides a standardized way to collect, process, and export telemetry data.

Learn more:

- [Monitor with server telemetry metrics](#), *Amazon GameLift Servers Developer Guide*

October 2, 2025: Amazon GameLift Servers adds console-based remote access to fleet instances

Amazon GameLift Servers now provides direct remote access to managed EC2 and container fleet instances through the Amazon GameLift Servers console, eliminating the need for authentication management. This new feature uses Amazon EC2 Systems Manager (SSM) to provide secure, browser-based terminal access without requiring additional setup or credential management.

With console-based remote access, you can now:

- Troubleshoot game server integration issues in real-time
- Monitor game server activity and access log files during active game sessions
- Run diagnostics and benchmarking tools using actual player traffic

This feature is available for fleets running server SDK version 5.x. For fleets running earlier SDK versions, you can continue to use the existing AWS CLI method for remote access.

To remotely connect to a fleet instance, open the Amazon GameLift Servers console and find the managed EC2 or container fleet you want to access. In the fleet's details, open the **Instances** tab, select an instance from the list, and choose **Connect**.

Learn more:

- [Connect to fleet instances](#), *Amazon GameLift Servers Developer Guide*

September 23, 2025: Amazon GameLift Servers adds support for new Local Zone in Dallas, Texas

With Amazon GameLift Servers managed hosting, you can now deploy game server resources in a new Dallas, TX, Local Zone (us-east-1-dfw-2). Local Zones give you the ability to place your game servers geographically closer to your players to reduce latency and significantly improve gameplay. In this new Local Zone, you can use the following EC2 instance types: C6gn, C6i, C6in, M6g, M6i, M6in, M8g, and R6i. For detailed descriptions of each instance type, see [AWS EC2 instance types](#).

To start hosting game sessions in the new Dallas Local Zone, first opt in to the Local Zone for your AWS account. Then you can add it as a remote location to a new or existing multi-location fleet. If your game uses Amazon GameLift Servers FlexMatch, update any of the fleets in your matchmaking queue to include the new Local Zone. With multi-location fleets, you can directly manage hosting capacity in each location.

The parent AWS Region for the new Dallas, TX, Local Zone is us-east-1 (Virginia). The original Dallas, TX, Local Zone (us-east-1-dfw-1) is no longer accepting new opt-in requests.

Learn more:

- [Getting started with AWS Local Zones](#), *AWS Local Zones User Guide*
- [Amazon GameLift Servers service locations](#), *Amazon GameLift Servers Developer Guide*
- [Update fleet locations](#), *Amazon GameLift Servers Developer Guide*

August 12, 2025: Amazon GameLift Servers plugin for Unreal Engine now supports UE version 5.6

Updated plugin version:

- [Plugin for Unreal](#), version 3.0.1 (includes the server SDK for Unreal, version 5.3.1)

Amazon GameLift Servers releases new versions of the plugin for Unreal Engine and the server SDK for Unreal Engine. These latest versions include the following updates:

- They now support use with Unreal Engine 5.6.
- Compilation issues when packaging Android client targets are resolved.

Learn more:

- [Integrating games with the Amazon GameLift Servers plugin for Unreal Engine](#), *Amazon GameLift Servers Developer Guide*
- [Integrate Amazon GameLift Servers into an Unreal Engine project](#), *Amazon GameLift Servers Developer Guide*

June 24, 2025: Amazon GameLift Servers launches UDP ping beacons to help measure real-time network latency for multiplayer games

Updated SDK versions:

- AWS SDK 1.11.595

Amazon GameLift Servers releases for general availability a set of fixed endpoints called UDP ping beacons to help you accurately measure latency between player devices and game server locations. The UDP ping beacon endpoints are available in all AWS Global Regions and Local Zones supported by Amazon GameLift Servers, with the exception of the AWS China Regions.

Most multiplayer games use UDP (User Datagram Protocol) as their primary packet transmission protocol due to its performance benefits for real-time gaming. Understanding and optimizing network latency is crucial for delivering the best possible player experience. UDP ping beacons provide a consistent and reliable way to measure actual UDP packet latency between players and game servers, helping you make better decisions about player-to-server matching and game session placement.

The [ListLocations](#) API has been extended to include endpoint domain and port information as part of the list of locations it returns, making it easy to programmatically access the endpoints.

Your game client can send UDP messages to these endpoints and receive asynchronous responses with the same data, giving you latency measurements that better represent actual game traffic conditions between a player's device and potential hosting locations. These endpoints are permanent and remain available as long as Amazon GameLift Servers supports game hosting in that location.

Learn more:

- [UDP ping beacons](#), *Amazon GameLift Servers Developer Guide*
- [ListLocations](#), *Amazon GameLift Servers API Reference*

June 24, 2025: Amazon GameLift Servers adds location support for Thailand and Malaysia

With Amazon GameLift Servers managed hosting, you can now deploy game server resources in Bangkok, Thailand, and Kuala Lumpur, Malaysia, extending the reach of your games to players

throughout Southeast Asia. These new Regions help reduce latency and improve gameplay experience for players in those areas.

The following AWS Regions are available as remote locations for multi-location fleets. To begin hosting game sessions in these locations, add them as remote locations to a new or existing multi-location fleet. With multi-location fleets, you can directly manage hosting capacity in each location.

- Asia Pacific (Thailand) (ap-southeast-7)
- Asia Pacific (Malaysia) (ap-southeast-5)

These AWS Regions are not enabled by default for an AWS account. You must opt in to each Region before you can deploy Amazon GameLift Servers resources there.

Learn more:

- [Amazon GameLift Servers service locations](#), *Amazon GameLift Servers Hosting Developer Guide*
- [Update fleet locations](#), *Amazon GameLift Servers Hosting Developer Guide*
- [Enable or disable AWS Regions in your account](#), *AWS Account Management Reference Guide*.

May 29, 2025: Amazon GameLift Servers releases new versions of the server SDK and plugins for Unreal Engine and Unity

Updated SDK versions:

- [C++ server SDK](#), version 5.3.0
- [C# server SDK](#), version 5.3.0
- [Go server SDK](#), version 5.3.0
- [Server SDK for Unreal](#), version 5.3.0
- [Server SDK for Unity](#), version 5.3.0

Updated plugin versions:

- [Plugin for Unreal](#), version 3.0.0
- [Plugin for Unity](#), version 3.1.0

New versions of the server SDK for C++, C#, Go, Unreal and Unity, as well as new plugin versions for Unreal Engine and Unity, are now open source. They are all available in the [Amazon GameLift Servers GitHub organization](#). For detailed update information, see the release notes and readmes in each repository.

Key server SDK updates:

- Improved client-side validation and error responses in all server SDKs.
- The [OnProcessTerminate](#) callback function now has default logic to end the game server process.
- The function [InitSDK\(\)](#) now uses an idempotency token to support multiple retries.
- The [OnUpdateGameSession](#) callback can now pass data for `autoBackfillMode`.

Key plugin updates:

- The plugin for Unreal Engine now has a more streamlined install and setup process, with more automation and fewer prerequisites (CMake, OpenSSL, and the Unreal cross-compile tool chain).
- The plugin for Unreal Engine has improved UI experience for the Managed EC2 workflow, including support for spaces in the game client and server build paths. In addition, you can now add command-line arguments when launching a game client from the Editor.
- The plugin for Unreal Engine now supports ARM server builds in UE5.

Learn more:

- [Amazon GameLift Servers plugin for Unreal Engine](#), *Amazon GameLift Servers Developer Guide*
- [Amazon GameLift Servers plugin for Unity \(server SDK 5.x\)](#), *Amazon GameLift Servers Developer Guide*
- [Server SDK 5.x for Amazon GameLift Servers](#), *Amazon GameLift Servers Developer Guide*
- [Integrate Amazon GameLift Servers into an Unreal Engine project](#), *Amazon GameLift Servers Developer Guide*
- [Integrate Amazon GameLift Servers into a Unity project](#), *Amazon GameLift Servers Developer Guide*

April 24, 2025: C# server SDK 5 for Amazon GameLift Servers now supports .NET 8

Updated SDK versions:

- C# Server SDK, version 5.2.1

For game developers using C#, you can now use .NET 8 as a target framework for your Amazon GameLift Servers projects. With .NET 8, you can take advantage of performance improvements, including improved just-in-time (JIT) compilation, memory usage optimization, and faster startup times. If you're currently using .NET 6, we recommend that you plan a migration to .NET 8, including updating your C# server SDK to the latest version. Microsoft announced .NET 8 support, with security patches and technical updates, through November 2026.

[Download the latest version of the C# server SDK for Amazon GameLift Servers](#)

Learn more:

- [Get Amazon GameLift Servers development tools](#), *Amazon GameLift Servers Developer Guide*

March 27, 2025: Amazon GameLift Servers expands support for next-generation EC2 instance families

Updated SDK versions:

- AWS SDK 1.11.535

You can now fine-tune your game server hosting with Amazon GameLift Servers by selecting from a larger selection of Amazon EC2 instances across the 5th through 8th generation instance families. Each new EC2 generation offers advancements in EC2 compute, memory and networking, with the 8th generation instances delivering cutting-edge AWS Graviton4 and Intel Xeon-based instances. Next generation instances are available in the following instance families:

- [General purpose](#) (M-series)
- [Compute optimized](#) (C-series)
- [Memory optimized](#) (R-series)

You can also choose variants that offer local storage (d), enhanced networking (n), and specific processor architectures (Intel/AMD/Graviton – i/a/g). The next-generation instances are available in all AWS Regions that are supported by Amazon GameLift Servers, with the exception of the AWS China Regions. For more details, see [Amazon GameLift Servers service locations](#).

Use these new instance types with Amazon GameLift Servers managed EC2 fleets and managed container fleets. When switching your existing game hosting to a new instance type (same architecture), simply deploy new fleets with all configuration settings unchanged except the instance type.

Learn more:

- [Amazon GameLift Servers Instance Pricing](#)
- [CreateFleet](#) *Amazon GameLift Servers API Reference*
- [CreateContainerFleet](#), *Amazon GameLift Servers API Reference*

March 11, 2025: Amazon GameLift Servers launches new Game Server Wrapper for rapid onboarding

The new game server wrapper for Amazon GameLift Servers significantly reduces the time required to get a game server hosted on Amazon GameLift Servers. With no code changes required, you can use the wrapper to add basic game session management functionality to your game and deploy it to an Amazon GameLift Servers Anywhere fleet, managed EC2 fleet, or managed container fleet. This tool is ideal for doing a hands-on evaluation of Amazon GameLift Servers features, using your own game server or one from a sample game. It is also useful for quickly deploying game server iterations, such as for rapid prototyping or testing.

With the basic game session management features, your game server can initialize a connection with the Amazon GameLift Servers service, respond to prompts to start and stop game sessions, and shut down when a game session is complete.

[Download the game server wrapper from GitHub](#). See the readme on GitHub for how to install and use the wrapper with your game projects.

March 6, 2025: Amazon GameLift is now Amazon GameLift Servers and Amazon GameLift Streams

Amazon GameLift is a fully-managed service that's dedicated to helping developers build, scale, and deliver the world's most demanding games. With the general availability release of Amazon

GameLift Streams, Amazon GameLift now offers both high-scaling game server and smooth-gameplay streaming capabilities.

[Amazon GameLift Servers](#) gives game developers the capability to deploy, operate, and scale dedicated game servers. You can deploy high-performance game servers in the cloud in minutes and scale up and down to meet player demand. Built on AWS proven computing environment, Amazon GameLift Servers supports 100 million concurrent players in a single game, 100 thousand player adds per second, and 9 thousand new compute instances per minute. And with enterprise-grade security, matchmaking for the largest crowds, and pay-as-you-go flexibility, it helps you get started whether you're working on a new game idea or running a game with millions of players.

[Amazon GameLift Streams](#) helps game developers deliver game streaming experiences at up to 1080p resolution and 60 frames-per-second (fps) with no perceivable lag across devices including iOS, Android, FireOS, and PCs for gamers. Using a single AWS offering, publishers can deploy their game content in minutes, without modifications, onto fully-managed cloud-based GPU instances and deliver them through the AWS Network Backbone directly to any end-player device with a web browser. Players can start gaming in just a few seconds, without waiting for a download or an install and it delivers a gameplay experience that is nearly indistinguishable from playing the game locally on a PC or gaming console.

January 14, 2025: Amazon GameLift Servers releases expanded support for game session placement prioritization

Updated SDK versions:

- AWS SDK 1.11.485

In response to customer feedback, we're releasing new functionality that lets you prioritize locations for individual game session placement requests. For your queues that are configured to prioritize placement by location, you can now provide a customized list of priority locations with each placement request.

This new feature lets customers dynamically change location priorities for each placement request as needed. The additional flexibility means that you can better respond to changing conditions, such as player locations, fleet load, or server health. It can also support customers who want to further customize how placement locations are selected.

Learn more:

- [Prioritize game session placement](#), *Amazon GameLift Servers Developer Guide*
- [StartGameSessionPlacement](#), *Amazon GameLift Servers API Reference*

January 2, 2025: Amazon GameLift Servers releases new support for terminating game sessions**Updated SDK versions:**

- AWS SDK 1.11.477

In response to customer feedback, we're releasing new functionality that lets you more easily terminate individual game sessions. With this release, you can now terminate a game session directly in the Amazon GameLift Servers console or by using the AWS CLI or AWS SDK for Amazon GameLift Servers.

This new feature addresses the need to resolve game sessions that remain active but in a bad state, which prevents compute resources from hosting new game sessions. Previously, customers were required to remotely access the compute to manually terminate a game session.

You have two termination methods to choose from. The first method attempts to gracefully terminate a game session using its custom shutdown sequence, which might include actions to notify players and resolve game data. The second method forces the server process to stop, which terminates the game session immediately. This second method ensures that the game session ends even when the server process is not responding.

Learn more:

- [Shut down a game session using the Amazon GameLift Servers console](#), *Amazon GameLift Servers Developer Guide*
- [TerminateGameSession](#), *Amazon GameLift Servers API Reference*

December 19, 2024: Amazon GameLift Servers releases game engine plugin support for Managed Containers**Updated plugin versions:**

Amazon GameLift Servers plugin for Unreal Engine, version 2.0.0

- Upgraded to support C++ server SDK 5.2.0 with managed containers support.
- Added support for Unreal Engine 5.4 and 5.5.

Amazon GameLift Servers plugin for Unity, version 3.0.0

- Upgraded to support C++ server SDK 5.2.0 with managed containers support.
- Support for Unity 2021.3 LTS and 2022.3 LTS for Windows and Mac OS.

The Amazon GameLift Servers plugin for the Unreal and Unity game engines provides tools and workflows that streamline your steps to getting a game up and running with Amazon GameLift Servers. Amazon GameLift Servers is a fully managed cloud hosting service that game developers can use to manage and scale dedicated game servers for session-based multiplayer games.

The latest plugin versions offer the following enhancements:

- **Guided workflow for hosting with Managed Containers.** This workflow walks you through the steps to set up a container image with your game server software, and deploy a cloud-based hosting solution for your game server. The workflow offers two different deployment scenarios: a simple deployment and a more complete deployment with a game session placement queue and a FlexMatch matchmaker. Each scenario generates Amazon GameLift Servers container fleets and supporting AWS resources.
- **Improved process for setting up AWS user profiles and managing AWS access credentials for plugin use.** You can maintain multiple profiles to work with different AWS accounts, account users, and regions.
- **Additional functionality to update existing container fleets.** You can deploy new container images (such as for game server version updates) and change fleet configuration settings without having to start from the beginning.
- **Improved workflows for hosting with Amazon GameLift Servers Anywhere fleets and Managed EC2.** Improvements based on customer feedback include better guidance with tips and links to helpful resources.

The deployment scenarios for Managed Containers and Managed EC2 solutions use AWS CloudFormation templates to create and deploy the AWS resources for each scenario. These

templates are included in the Amazon GameLift Servers plugin download and are editable. You can use them as is or modify them for your game.

Learn more:

- [Plugin for Unreal: Deploy your game to a managed container fleet](#), *Amazon GameLift Servers Developer Guide*
- [Plugin for Unity: Deploy your game to a managed container fleet](#), *Amazon GameLift Servers Developer Guide*
- [Download the plugin for Unreal from GitHub](#)
- [Download the plugin for Unity from GitHub](#)
- [Game hosting with Amazon GameLift Servers](#)
- [Amazon GameLift Servers forum](#)

November 12, 2024: Amazon GameLift Servers launches multiplayer game hosting with Managed Containers

Updated SDK versions:

- AWS SDK 1.11.445
- Server SDK, version 5.2.0 (all languages)

Amazon GameLift Servers releases for general availability a new hosting solution for containerized game server workloads. With this release, game developers can now take advantage of the benefits of containerization including consistent, secure environments, a simplified deployment process, and optimized resource utilization.

Managed container fleets use Amazon EC2 instances that are managed by Amazon GameLift Servers on your behalf and based on your configurations. You build a custom container architecture for your game and provide container images by storing them in a Amazon Elastic Container Registry (Amazon ECR) repository. Container fleets are available for Linux-based game servers only. Game servers must be integrated with Server SDK 5.2.0 or greater.

With managed container fleets, you get the same benefits as with managed EC2 fleets. This includes support for On-Demand and Spot instance types, intelligent capacity scaling, game session placement with queues, and matchmaking. You also get the same metrics as other fleet types along with some new ones for containers. Other features for container fleets include:

- **Alignment with serverless experience for containerized workloads.** Run one game server process per container and pack many containers onto each fleet instance for optimal resource usage. If you prefer to have containers with multiple game server processes, you can use the Amazon GameLift Servers Agent for automated host management.
- **Streamlined fleet creation.** Container fleets are designed to require minimal deployment configuration settings, with sensible suggested/default values. You can quickly deploy a working fleet, and then customize individual settings as needed.
- **Versioning tools for container architecture.** You can now update a container group definition (which is similar to a container "task"), maintain multiple versions, and specify which version to deploy to a fleet.
- **Fleet update tools.** With container fleets, you no longer need to create a new fleet when you want to release a game server version update. Instead, you can now update your container image and deploy the updates to existing fleets.

You can build Amazon GameLift Servers container fleets in any AWS Region where Amazon GameLift Servers supports multi-location fleets, and you can deploy container fleet instances to any supported remote location. For more details, see [Amazon GameLift Servers service locations](#). Managed containers is not currently available in AWS China Regions.

Learn more:

- Blog post: [Leverage fully managed containers to host multiplayer games at global scale on Amazon GameLift Servers](#)
- [Managed containers](#) overview, *Amazon GameLift Servers Developer Guide*
- [How containers work in Amazon GameLift Servers](#), *Amazon GameLift Servers Developer Guide*
- [Development roadmap for hosting with Amazon GameLift Servers managed containers](#), *Amazon GameLift Servers Developer Guide*
- [CreateContainerFleet](#), *Amazon GameLift Servers API Reference*

September 19, 2024: Amazon GameLift Servers releases update to the C++ server SDK and plugins for Unreal Engine

Updated SDK versions:

C++ Server SDK, version 5.1.3

- New logging capabilities. You can now access SDK request logs.
- Improved SDK message transmission reliability. The SDK now uses more robust reconnection mechanisms to recover in the event of network interruptions or random message drops.

Updated plugin versions:

Amazon GameLift Servers plugin for Unreal Engine, version 1.1.2

- Upgraded to support the latest version of the C++ server SDK 5.1.3.
- In the Amazon GameLift Servers plugin for Unreal Engine, when browsing for a server build executable for a fleet, you now have the option to browse **All Files**.

C++ Server SDK Plugin for Unreal, version 5.1.2

- Upgraded to support the latest version of the C++ server SDK 5.1.3.

Learn more:

- [Integrating games with the Amazon GameLift Servers plugin for Unreal Engine](#), *Amazon GameLift Servers Developer Guide*
- [Amazon GameLift Servers plugin and SDK downloads](#)

September 5, 2024: Amazon GameLift Servers improves observability of the fleet creation process

Based on customer feedback, we've clarified the Amazon GameLift Servers workflow for creating a managed EC2 fleet and getting it ready to host game sessions. Improvements include:

- We've provided more specific and accurate descriptions of each phase of the fleet creation process. This improved visibility makes it easier to pinpoint and resolve issues faster.
- The Building and Activating phases better separate instance deployment tasks (building) from tasks to start game server processes and connect to the Amazon GameLift Servers service (activating). This change makes it easier to recognize the likely cause of issues. In addition, you can now remotely connect to fleets when they're in the Activating phase.

- Two new fleet creation events communicate the success or failure of game server install scripts. If your game server build includes an install script, Amazon GameLift Servers attempts to run the script and emits one of the following new events:
 - FLEET_CREATION_COMPLETED_INSTALLER
 - FLEET_CREATION_FAILED_INSTALLER

Learn more:

- [Debug Amazon GameLift Servers fleet issues](#), *Amazon GameLift Servers Developer Guide*
- [Event data type](#), *Amazon GameLift Servers API Reference*

August 15, 2024: Amazon GameLift Servers improves console experience

Based on customer feedback, we've made the following updates to the [Amazon GameLift Servers console](#) experience:

- Your display preferences for pages are now automatically saved to your AWS account user and applied whenever you return to the page. Display preferences let you choose what information to include in a table display, such as on the Fleets listing page. Customize your display preferences by using the  icon in the upper right corner of a table.
- The Create Fleet workflow for managed EC2 fleets has been streamlined to combine the selection of fleet locations and instance types. We've made it easier for you to find the right instance type for your fleet, even when you change your locations selections.

Learn more:

- [Create an Amazon GameLift Servers managed EC2 fleet](#), *Amazon GameLift Servers Developer Guide*

July 25, 2024: Amazon GameLift Servers adds support for AWS Nigeria Local Zone

With Amazon GameLift Servers managed hosting, you can now deploy game server resources in Nigeria, West Africa, and extend the reach of your games to players throughout Africa. Use AWS

Local Zones to place game servers geographically closer to your players to reduce latency and significantly improve gameplay.

To immediately begin hosting game sessions in Nigeria, add the new Nigeria Local Zone as a remote location to a new or existing multi-location fleet. If your game uses Amazon GameLift Servers FlexMatch, update fleets in your matchmaking queue to include the new Local Zone. With multi-location fleets, you can directly manage hosting capacity in each location.

The parent AWS Region for the Lagos, Nigeria Local Zone is the Africa (Cape Town) Region (af-south-1), which Amazon GameLift Servers also supports as a remote location. The Nigeria Local Zone name is af-south-1-los-1.

Learn more:

- [Amazon GameLift Servers service locations](#), *Amazon GameLift Servers Developer Guide*
- [Update fleet locations](#), *Amazon GameLift Servers Developer Guide*

July 02, 2024: Amazon GameLift Servers releases new console tool to view player session data

The Amazon GameLift Servers console now offers a player session lookup tool that lets you retrieve player session information by game session ID, player session ID, or player ID. Games that use FlexMatch matchmaking automatically generate player sessions for every matched player. For all other games, player sessions are an optional feature.

You can find the player session lookup tool in the main navigation for the Amazon GameLift Servers console. View individual player sessions or compare data across multiple player sessions. You can also open player session data when viewing a game session detail page.

Learn more:

- [Game and player sessions in the Amazon GameLift Servers console](#), *Amazon GameLift Servers Developer Guide*

April 24, 2024: Amazon GameLift Servers releases container fleets preview

Amazon GameLift Servers is now offering a preview of container fleets, which give you improved portability, scalability, fault tolerance, and agility.

In container fleets, Amazon EC2 instances host one or more of your containers. These containers include your game server along with whatever it requires, including dependencies and configurations. Examples of dependencies include SDKs and software packages. After you upload your container to your private Amazon Elastic Container Registry, Amazon GameLift Servers populates your fleet with the container.

To function in a container fleet, your game server must run in Linux and be integrated with Server SDK 5.x. In a container fleet, you have fine-tuned control of hosting resources so that you can optimize consumption of resources such as CPU units and memory. You can also host multiple game servers in a container to reduce the use of resources.

In a container fleet you get many of the same benefits that other types of fleets have such as On-Demand instance types, scaling (automatic and manual), queues, and matchmaking. You also get the same metrics as other fleet types along with some new ones for containers. Container fleets give you global reach to players in these locations regions:

- ap-northeast-1
- ap-northeast-2
- ap-southeast-2
- eu-central-1
- eu-west-1
- us-east-1
- us-west-2

To reach even more regions and local zones, create multi-location containers fleets.

Learn more:

- [Managing hosting with Amazon GameLift Servers containers](#), *Amazon GameLift Servers Developer Guide*
- [CreateContainerGroupDefinition](#), *Amazon GameLift Servers API Reference*

February 13, 2024: Amazon GameLift Servers launches improvements to SDKs, and simplifies installation of the Amazon GameLift Servers plugin for Unreal Engine

Updated SDK versions:

- Go Server SDK, version 5.1.0
- C# Server SDK, version 5.1.2
- C++ Server SDK, version 5.1.2

We made the following improvements:

- Improved the reliability of the SDK by adding automatic reconnection in the event of network interruption.
- [Go] You can now call `InitSDK()` with or without server parameters. Game servers that run on Amazon GameLift Servers managed EC2 fleets read the server parameters directly from environment variables. Game servers on Amazon GameLift Servers Anywhere fleets must call `InitSDK()` with server parameters.

Updated plugin versions:

- Amazon GameLift Servers plugin for Unreal Engine, version 1.1.0
- Amazon GameLift Servers plugin for Unity, version 2.1.0
- C++ Server SDK Plugin for Unreal, version 5.1.1
- C# Server SDK Plugin for Unity, version 5.1.2

We made the following improvements:

- [Amazon GameLift Servers plugin for Unreal Engine] Updated the installation instructions and simplified the packaging. This plugin now includes the latest version of the C++ Server SDK for Unreal.
- Upgraded the plugins to support the latest version of the server SDK for Amazon GameLift Servers.

Learn more:

- [Integrating games with the Amazon GameLift Servers plugin for Unreal Engine](#), *Amazon GameLift Servers Developer Guide*
- [Amazon GameLift Servers plugin and SDK downloads](#)

December 14, 2023: Amazon GameLift Servers adds ability to update the game properties of active game sessions

You've already been able to set game properties when creating game sessions, and to search game sessions for specified properties. Now you can also add and update these properties in an active game session.

For example, your players vote on a map that they want to play on. Your game client calls `UpdateGameSession` to modify a `GameProperty` value to `{"Key": "map", "Value": "jungle"}`. Your game then implements the new map for the players in the game session.

Game administrators can also retrieve useful data from game properties by using the `SearchGameSessions` operation. For example, administrators can list game sessions that have a `Status` value of `ACTIVE` and this game property: `{"Key": "map", "Value": "desert"}`.

Learn more:

- [the section called "Integrate game client functionality"](#), *Amazon GameLift Servers Developer Guide*
- [GameProperty](#), *Amazon GameLift Servers API Reference*
- [UpdateGameSession](#), *Amazon GameLift Servers API Reference*
- [SearchGameSessions](#), *Amazon GameLift Servers API Reference*

November 21, 2023: Amazon GameLift Servers launches support for Infrastructure as Code tools like Terraform and Pulumi powered by AWS Cloud Control API

You can now manage your entire Amazon GameLift Servers resource stack using Infrastructure as Code (IaC) tools. These tools include AWS CloudFormation, and also third-party tools such as Terraform and Pulumi. With this added support, you can now focus on building your game, and

leverage DevOps strategies to take care of resource management, CI/CD, and deployment to your customers.

You can also now provision and configure all Amazon GameLift Servers resources types by using the AWS Cloud Control API. You can continue to work with resources using the Amazon GameLift Servers APIs or the AWS CloudFormation templates for Amazon GameLift Servers.

For details about the Amazon GameLift Servers resources available through IaC, see the [Amazon GameLift Servers resource type reference](#) Amazon GameLift Servers resource type reference.

In addition, you can now automatically scale your fleets using AWS CloudFormation templates or the AWS Cloud Control API by using the new [Fleet](#) property: `ScalingPolicies`.

The Cloud Control API gives developers a standard set of APIs to create, read, update, delete, and list resources (CRUDL) across hundreds of AWS services and multiple third-party tools like Terraform and Pulumi.

Learn more:

- [AWS CloudFormation](#)
- [AWS Cloud Control API](#)
- [AWS CC Terraform Provider](#)
- [Pulumi](#)

November 16, 2023: Amazon GameLift Servers updates standalone plugin for Unity

Updated SDK versions: Amazon GameLift Servers plugin for Unity, version 2.0.0

The Amazon GameLift Servers plugin for Unity provides tools and workflows that streamline the steps to getting your Unity game up and running for cloud hosting with Amazon GameLift Servers. Amazon GameLift Servers is a fully managed service that lets game developers manage and scale dedicated game servers for session-based multiplayer games.

With this version, the plugin for Unity is updated to use the latest Amazon GameLift Servers features, including server SDK version 5.x and support for local testing with Amazon GameLift Servers Anywhere. The plugin is compatible with Unity versions Unity 2021.3 LTS and 2022.3 LTS.

Key plugin features include:

- Guided UI workflows in the Unity editor for the following scenarios:
 - Test your game integration with Amazon GameLift Servers using your local workstation as a host. This workflow helps you set up an Amazon GameLift Servers Anywhere fleet for your local machine, launch instances of your game server and client, request a game session through Amazon GameLift Servers, and join the game.
 - Deploy cloud hosting solution for your integrated game server with Amazon GameLift Servers managed EC2 and supporting AWS resources. This workflow helps you configure your game for cloud hosting, and provides three deployment scenarios:
 - Deploy the game server to a single fleet.
 - Deploy the game server to a set of low-cost Spot fleets in multiple AWS Regions.
 - Deploy the game server with a FlexMatch matchmaker.
- Ability to set up user profiles that link to an AWS account user and set a default AWS Region. You can maintain multiple profiles to work in different AWS accounts, account users, and regions.
- Special conveniences that help streamline the Amazon GameLift Servers integration and deployment processes, including:
 - Each hosting solution includes supporting AWS resources, including an Amazon Cognito user pool that provides unique player IDs and player validation. The solutions also include an Amazon S3 bucket for storage, Amazon SNS event notification, AWS Lambda functions, and other resources.
 - For the Anywhere workflow, the plugin automates the required server parameter settings.
 - For the Amazon EC2 workflow, each deployment solution provides a built-in client backend service using Lambda functions. The backend service sits between the game client and the Amazon GameLift Servers service and manages all direct calls to the Amazon GameLift Servers service.
- Content for integration testing, including assets and code for a simple sample multiplayer game to illustrate game server and game client integration.
- Plugin documentation with detailed integration guidance and sample code.

All deployment scenarios, including for Anywhere and Amazon EC2 fleets, use AWS CloudFormation templates to describe and deploy the AWS resources for your game's solution. These templates are included in the Amazon GameLift Servers plugin download. You can use them as is or customize them for your game.

Learn more:

- [Amazon GameLift Servers plugin for Unity \(server SDK 5.x\)](#), *Amazon GameLift Servers Developer Guide*
- [Download the plugin from GitHub](#)
- [About Amazon GameLift Servers hosting](#)
- [Amazon GameLift Servers forum](#)

November 2, 2023: Amazon GameLift Servers adds support for shared credentials**Updated SDK versions:** AWS SDK 1.11.193

The new Amazon GameLift Servers shared credentials feature allows applications that are deployed on managed EC2 fleets to interact with other AWS resources. This update affects applications that you bundle and deploy along with game server binaries integrated with server SDK version 5.x or later. (Game server executables can already request credentials using the server SDK 5.x `GetFleetRoleCredentials()` action.)

For example, if you want to deploy your game server build with an Amazon CloudWatch agent to collect EC2 instance metrics and other data, the agent needs permission to interact with your CloudWatch resources. To do this, you must first set up an AWS Identity and Access Management (IAM) role with permissions to use the CloudWatch resources, and then configure a fleet with the IAM role and shared credentials enabled. When Amazon GameLift Servers deploys your game server build to each EC2 instance, it generates a shared credentials file and stores it on the instance. All applications on the instance can use the shared credentials. Amazon GameLift Servers automatically refreshes the temporary credentials throughout the life of the instance.

You can enable shared credentials when you create a managed EC2 fleet using the following methods:

- In the Amazon GameLift Servers console fleet creation workflow.
- When calling the service API operation `CreateFleet` using the new parameter `InstanceRoleCredentialsProvider`.
- When calling the AWS CLI operation `aws gamelift create-fleet` with the parameter `instance-role-credentials-provider`.

Learn more:

- [Communicate with other AWS resources from your fleets](#), *Amazon GameLift Servers Developer Guide*
- [CreateFleet, InstanceRoleCredentialsProvider](#), *Amazon GameLift Servers API Reference*
- [Set up an IAM service role](#), *Amazon GameLift Servers Developer Guide*

September 28, 2023: Amazon GameLift Servers releases new standalone plugin for Unreal Engine Engine

Updated SDK versions: Amazon GameLift Servers plugin for Unreal Engine version 1.0.0

The Amazon GameLift Servers plugin for Unreal Engine provides tools and workflows that streamline your steps to getting a game up and running with Amazon GameLift Servers for cloud hosting. Amazon GameLift Servers is a fully managed service that lets game developers manage and scale dedicated game servers for session-based multiplayer games. The plugin supports UE versions 5.0, 5.1, and 5.2. Key features include:

- Guided UI workflows in the Unreal editor]step through the following paths:
 - Test your game integration with Amazon GameLift Servers using your local workstation as a host. This workflow helps you set up an Amazon GameLift Servers Anywhere fleet for your local machine, launch instances of your game server and client, request a game session through Amazon GameLift Servers, and get connection information for the new game session.
 - Deploy an Amazon EC2 cloud hosting solution for your integrated game server. This workflow helps you configure your game for cloud hosting, and provides three different deployment scenarios: deploy to a single fleet, deploy to a set of spot fleets in multiple regions, or deploy to a set of fleets with a FlexMatch matchmaker. The solution for each deployment scenario includes Amazon GameLift Servers resources and supporting AWS resources.
- Ability to set up user profiles that link to an AWS account user and define a default AWS Region. You can maintain multiple profiles to work in different AWS accounts, account users, and regions.
- Special conveniences that help streamline the Amazon GameLift Servers integration and deployment processes, including:
 - Each hosting solution includes supporting AWS resources, including a basic Amazon Cognito user pool that provides unique player IDs, an Amazon S3 bucket for storage, Amazon SNS event notification, and AWS Lambda functions.

- For the Anywhere workflow, the plugin automates the required server parameter settings using command line arguments.
- For the Amazon EC2 workflow, each deployment solution provides a built-in client backend service using Lambda functions. The backend service receives requests from game clients and passes them on to the Amazon GameLift Servers service.
- Content for integration testing, including a starter game map and two testing maps with basic blueprints and UI elements.
- Plugin documentation with detailed integration guidance and sample code.

All deployment scenarios, including for Anywhere and Amazon EC2 fleets, use AWS CloudFormation templates to describe the solutions. The plugin uses these templates when deploying Amazon GameLift Servers resources for your game. These templates are included in the Amazon GameLift Servers plugin download and are editable. You can use them as is or modify them for your game.

Learn more:

- [Amazon GameLift Servers plugin for Unreal Engine](#), *Amazon GameLift Servers Developer Guide*
- [Download the plugin from GitHub](#)
- [About Amazon GameLift Servers hosting](#)
- [Amazon GameLift Servers forum](#)

August 17, 2023: Amazon GameLift Servers offers game server hosting with AWS Graviton processors

Updated SDK versions: AWS SDK 1.11.144

With Amazon GameLift Servers you can now host your games in the cloud using EC2 instances with AWS Graviton processors. Designed by AWS with Arm64-based processors, Graviton instances deliver the best price performance for cloud workloads using EC2, with up to 40% improvement over comparable x86-based instances. The latest Graviton3 processors offer up to 25% better compute performance over earlier versions.

With Amazon GameLift Servers, you can now select from these new instances in the AWS Graviton family:

- Graviton2-based instances: c6g, c6gn, r6g, m6g, g5g

- Graviton3-based instances: c7g, r7g, m7g

Learn more:

- [AWS Graviton Processor](#): Learn about the benefits and practical uses of Graviton-based EC2 instances.
- [Getting started with Graviton](#): Get an overview of the Graviton-based instances and insights on how applications run on them depending on their operating system, languages, and run times.

Note

Graviton Arm instances require a server build for a Linux AMI. Server SDK 5.1.1 or newer is required for C++ and C#. Server SDK 5.0 or newer is required for Go. These instances do not provide out-of-the-box support for Mono installation on Amazon Linux 2023 (AL2023) or Amazon Linux 2 (AL2).

July 27, 2023: Amazon GameLift Servers releases server SDK 5.1.0 with added support for Unity development

Updated SDK versions: Server SDK for C++, C#/Unity, Unreal 5.1.0

The newest release of the Amazon GameLift Servers server SDK delivers updates for C++, C#, and the Unreal plugin, and a new plugin for use with the Unity game engine. Game developers integrate the Amazon GameLift Servers server SDK into game servers that they deploy for hosting on Amazon GameLift Servers.

The latest server SDK version contains the following updates, which include a number of customer requests:

- **Download language-specific SDK packages** – The updated [Amazon GameLift Servers download site](#) contains SDK packages for each language. You can download current or previous versions.
- **New C# server SDK plugin for Unity** – The new server SDK package for Unity contains built C# libraries that you can install using the package manager in Unity Editor (see the new [Unity integration guide](#)). These libraries include the required dependencies through UnityNuGet. You can use this plugin with Unity 2020.3 LTS, 2021.3 LTS and 2022.3 LTS for Windows and Mac OS.

It supports Unity's .NET Framework and .NET Standard profiles, with .NET Standard 2.1 and .NET 4.x.

- **Consolidated .NET solution for C#** – The server SDK for C# now supports .NET Framework 4.6.2 (upgraded from 4.6.1) and .NET 6.0 in a single solution. .NET Standard 2.1 is available with the Unity-built libraries.
- **Server SDK 5.1.0 updates**
 - [C++, C#, Unreal] You can now call `InitSDK()` with or without server parameters. Game servers that run on Amazon GameLift Servers managed EC2 fleets read the server parameters directly from environment variables. Game servers on Amazon GameLift Servers Anywhere fleets must call `InitSDK()` with server parameters.
 - [C++, C#, Unreal] Server SDK calls have improved error messaging.
 - [C++ SDK] To improve Server SDK build times, the build flag `-DRUN_CLANG_FORMAT` is disabled by default. You can enable it with `-DRUN_CLANG_FORMAT=1`.
 - [C++ SDK] When building the libraries without the standard libraries (`-DGAMELIFT_USE_STD=0`), `InitSDK()` no longer uses `std::` data types.
- **Expanded server SDK 5.x documentation**
 - Updated server SDK reference guides for C++, C#/Unity, and Unreal including expanded coverage of all data types.
 - [C# server SDK 5.x for Amazon GameLift Servers -- Actions](#)
 - [C++ server SDK 5.x for Amazon GameLift Servers -- Actions](#)
 - [C++ \(Unreal\) server SDK 5.x for Amazon GameLift Servers -- Actions](#)
 - New versions of the server SDK 5 integration guides for Unity and Unreal plugins
 - [Integrate Amazon GameLift Servers into a Unity project](#)
 - [Integrate Amazon GameLift Servers into an Unreal Engine project](#)
- **Additional documentation updates**
 - Revised documentation for Amazon GameLift Servers service API operations [GetComputeAccess](#) and [GetInstanceAccess](#) to clarify remote access procedures based on the Amazon GameLift Servers server SDK version in use.
 - Revised descriptions for [GameSessionPlacement](#) to document how game session information is transient when a placement is in "pending" status.

July 13, 2023: Amazon GameLift Servers adds fleet hardware metrics

You can now track hardware performance metrics for your Amazon GameLift Servers managed EC2 fleets. Metrics include EC2 instance metrics for CPU utilization, network traffic volume, and disk read/write activity. For Amazon GameLift Servers, these metrics describe all active instances in a fleet location. You can view these fleet hardware metrics using an Amazon CloudWatch dashboard in the AWS Management Console. You can also view them in the Amazon GameLift Servers console in fleet details.

Learn more:

- [Monitor Amazon GameLift Servers with Amazon CloudWatch](#) (Metrics for fleets), *Amazon GameLift Servers Developer Guide*

June 29, 2023: Amazon GameLift Servers launches support for Amazon Linux 2023

Updated SDK versions: AWS SDK 1.11.111

Amazon GameLift Servers customers can now use the Amazon Linux 2023 operating system to host their game servers. AL2023 offers several improvements over AL2 including security. This operating system is available in all AWS Regions with the exception of the China Regions.

Customers can use the newer Linux operating systems and continue to receive critical security updates when support ends for Amazon Linux (AL1) in December 2023. Support for Amazon Linux 2 continues through June 30, 2026.

Learn more:

- [Amazon GameLift Servers Linux Server FAQ](#)
- [Comparing Amazon Linux 2 and Amazon Linux 2023](#)
- Amazon GameLift Servers API Reference links:
 - [AWS SDK action CreateBuild](#)
 - [CLI command upload-build](#)
 - [CLI command create-build](#)

May 25, 2023: Amazon GameLift Servers FleetIQ adds filter to exclude game session placements on draining instances

Updated SDK versions: AWS SDK 1.11.87

If you use Amazon GameLift Servers FleetIQ for game hosting, you can now prevent game session placements on instances that are currently draining. Draining instances are flagged for shutdown, but they can still be selected to host new game sessions if no other hosting resources are available. With this new feature, you can exclude the use of draining instances entirely.

Use this feature when calling `ClaimGameServer` to find available game servers. Add the new `FilterOption` parameter and set allowed instance statuses to `ACTIVE` only. In response, Amazon GameLift Servers FleetIQ looks only at active instances when searching for and claiming an available game server.

Learn more:

- [ClaimGameServer](#) in the *Amazon GameLift Servers API Reference*
- [How FleetIQ works](#) in the *Amazon GameLift Servers FleetIQ Developer Guide*

May 16, 2023: Amazon GameLift Servers supports cost allocation tagging for fleets

Amazon GameLift Servers customers can now use AWS Billing cost allocation tags to organize their game hosting costs. You can assign cost allocation tags to individual Amazon GameLift Servers EC2 fleet resources to track how your fleets are contributing to the overall hosting costs.

Learn more:

- [Resource cost and utilization tools](#)
- [Using AWS cost allocation tags](#), *AWS Billing User Guide*

April 20, 2023: Amazon GameLift Servers launches support for Windows Server 2016

Updated SDK versions: AWS SDK 1.11.63

Amazon GameLift Servers customers can now use the Windows Server 2016 operating system to host their game servers. This operating system is available in all AWS Regions. Customers can use

the newer Windows operating system and continue to receive critical security updates as Microsoft ends its support for Windows Server 2012 in October 2023.

Starting today, new customers who require a Windows runtime environment must specify Windows Server 2016 when creating new game server builds for hosting. Existing customers can continue to create new builds and fleets with Windows Server 2012 but must complete migration with Windows Server 2016 before the Microsoft end of support date on October 10, 2023.

This update includes the following service changes:

- When creating a game server build using Amazon GameLift Servers SDK or CLI commands, you must now explicitly set the operating system. There is no longer a default value. To deploy your game server on Windows Server 2016, use the value `WINDOWS_2016`.
- When creating a game server build using the Amazon GameLift Servers console, you must select an operating system from the available values. If you're an existing customer with active Windows Server 2012 fleets, you can choose either `WINDOWS_2012` or `WINDOWS_2016`.

Learn more:

- Amazon GameLift Servers API Reference links:
 - [CLI command `upload-build`](#)
 - [CLI command `create-build`](#)
 - [AWS SDK action `CreateBuild`](#)
- [Amazon GameLift Servers FAQ for Windows 2012](#)

April 13, 2023: Amazon GameLift Servers launches server SDK 5.x for Unreal

Updated SDK versions: Server SDK 5.0.0 for Unreal

The latest version of the Amazon GameLift Servers lightweight plugin for Unreal Engine is now based on the Amazon GameLift Servers server SDK 5.x. To start integrating your Unreal Engine environment with Amazon GameLift Servers see the following links.

Learn more:

- [Integrate Amazon GameLift Servers into an Unreal Engine project](#)
- [Add Amazon GameLift Servers to your game server with the server SDK](#)

- [C++ server SDK 5.x for Amazon GameLift Servers -- Actions](#)

March 14, 2023: Amazon GameLift Servers launches a new console experience

The new Amazon GameLift Servers console includes these improvements:

- **Improved navigation** – The new navigation pane facilitates navigation between Amazon GameLift Servers resources.
- **Amazon GameLift Servers landing page** – The new landing page provides links to helpful documentation, displays a high-level overview of Amazon GameLift Servers, and provides support through links to documentation, frequently asked questions, and AWS re:Post.
- **Improved Amazon CloudWatch metrics** – Amazon GameLift Servers metrics are now available in both the Amazon GameLift Servers console and your CloudWatch dashboards. This update also includes new metrics for performance, utilization, and player sessions.

Learn more:

- [Manage game hosting resources with Amazon GameLift Servers](#)
- [Building a FlexMatch matchmaker](#)

February 14, 2023: Amazon GameLift Servers now supports server side encryption for Amazon SNS topics

Server Side Encryption ((SSE)) for SNS topics encrypts your sensitive data at rest. SSE uses AWS Key Management Service (AWS KMS) keys to protect the contents of your SNS topics.

Learn more:

- [Set up event notification for game session placement](#)
- [FlexMatch matchmaking events](#)
- [Encryption at rest](#)

February 9, 2023: Amazon GameLift Servers server SDK supports .NET 6 with C#10

Updated SDK versions: Server SDK 5.0.0 for .NET 6. No SDK updates are required.

If you use the Unity Real-Time Development Platform, continue to use the Amazon GameLift Servers server SDK 5.0.0 with .NET 4.6. Unity doesn't support .NET 6.

Learn more:

- Download the latest version of the Amazon GameLift Servers server SDK at [Amazon GameLift Servers getting started](#)
- [C# server SDK 5.x for Amazon GameLift Servers -- Actions](#)

January 31, 2023: Amazon GameLift Servers server SDK supports the Go language

Updated SDK versions: Server SDK 5.0.0 for Go

Learn more:

- Download the latest version of the Amazon GameLift Servers server SDK at [Amazon GameLift Servers getting started](#)
- [Go server SDK for Amazon GameLift Servers -- Actions](#)

December 1, 2022: Amazon GameLift Servers launches Amazon GameLift Servers Anywhere and Amazon GameLift Servers Server SDK 5.0

Updated SDK versions: AWS SDK 1.10.21, Server SDK 5.0.0 for C++ and C#

Amazon GameLift Servers Anywhere uses your game server resources to host Amazon GameLift Servers game servers. You can use Amazon GameLift Servers Anywhere to integrate your own compute resources with Amazon GameLift Servers managed EC2 compute to distribute your game servers across multiple compute types. You can also use Amazon GameLift Servers Anywhere to iteratively test your game servers without uploading the build to Amazon GameLift Servers for every iteration.

Highlights:

- New Amazon GameLift Servers Anywhere fleet and compute types
- Amazon GameLift Servers Anywhere compute resource registration
- Improved testing iteration cycle

Amazon GameLift Servers Server SDK 5.0.0 introduces improvements to the existing server SDK and a new resource type, compute. Server SDK 5.0.0 supports Amazon GameLift Servers Anywhere and the use of your own compute resources for game server hosting.

Learn more:

- [Server SDK 5.x for Amazon GameLift Servers](#)
- [Geographic locations](#)
- [Choose compute resources for a managed fleet](#)
- [Create an Amazon GameLift Servers Anywhere fleet](#)

August 25, 2022: Amazon GameLift Servers launches support for Local Zones

Updated SDK versions: AWS SDK 1.9.333

Amazon GameLift Servers is now available in eight Local Zones in the United States, so you can deploy your fleets closer to players. You can use all managed Amazon GameLift Servers features with Local Zones by adding the Local Zones to your fleets.

Local Zones extend AWS resources and services to the edge of the cloud, near large population, industry, and information technology (IT) centers. This means that you can deploy applications that require single-digit millisecond latency closer to end users or to on-premises data centers.

Learn more:

- [Amazon GameLift Servers local zones](#)
- [Geographic locations](#)
- [Create an Amazon GameLift Servers managed EC2 fleet](#)

June 28, 2022: Amazon GameLift Servers launches a new opt-in console experience

The new Amazon GameLift Servers console includes these improvements:

- **Improved navigation** – The new navigation pane facilitates navigation between Amazon GameLift Servers resources.

- **Amazon GameLift Servers landing page** – The new landing page provides links to helpful documentation, displays a high-level overview of Amazon GameLift Servers, and provides support through links to documentation, frequently asked questions, and AWS re:Post.
- **Improved Amazon CloudWatch metrics** – Amazon GameLift Servers metrics are now available in both the Amazon GameLift Servers console and your CloudWatch dashboards. This update also includes new metrics for performance, utilization, and player sessions.

Learn more:

- [Manage game hosting resources with Amazon GameLift Servers](#)
- [Building a FlexMatch matchmaker](#)

February 15, 2022: FlexMatch adds compound rule and additional improvements

FlexMatch users now have access to the following features:

- **Compound rule** – Added support for compound matchmaking rules for matches of 40 or fewer players. You can now use logical statements to create a compound rule to form a match. Without a compound rule in your rule set, to form a match, all the rules in the rule set must be true. With compound rules, you can choose which rules to apply using the following logical operators: `and`, `or`, `not`, and `xor`.
- **Flexible team selection** – Updated matchmaking property expressions to support selecting a subset of all available teams.
- **Longer string lists** – Increased the maximum number of strings from 10 to 100 in a list of strings of player attribute values.

Learn more:

- [Amazon GameLift Servers FlexMatch developer guide:](#)
 - [FlexMatch rule types](#)
 - [FlexMatch property expressions](#)
- [AttributeValue: SL](#)

October 28, 2021: Amazon GameLift Servers adds support for multi-Region fleets in the Asia Pacific (Osaka) Region; Amazon GameLift Servers FleetIQ adds support for AWS Graviton2 processors

Updated SDK versions: AWS SDK [1.9.133](#)

Amazon GameLift Servers is now available in the Asia Pacific (Osaka) Region. Game developers can now deploy instances in Osaka using GameLift multi-Region fleet.

You can now use Graviton2-hosted game servers, based on the Arm-based processor architecture, to achieve increased performance at a lower cost when compared to the equivalent Intel-based compute options.

Highlights:

- Amazon GameLift Servers is now available in the Asia Pacific (Osaka) Region.
- Amazon GameLift Servers FleetIQ game server groups can now be configured to manage the Graviton2 instance families c6g, m6g, and r6g.

Learn more:

- [Amazon GameLift Servers multi-Region fleet](#)
- [CreateGameServerGroup](#)
- [AWS graviton processor](#)

September 20, 2021: Amazon GameLift Servers releases plugin for Unity

The Amazon GameLift Servers plugin for Unity version 1.0.0 contains libraries and native UI that makes it easier to access Amazon GameLift Servers resources and integrate Amazon GameLift Servers into your Unity game. You can use the Amazon GameLift Servers plugin for Unity to access Amazon GameLift Servers APIs and deploy CloudFormation templates for common gaming scenarios. The plugin also includes a sample game that works with the sample scenarios. You can use Amazon GameLift Servers Local to see messages passed between the game client and the game server to learn how a typical game interacts with Amazon GameLift Servers.

The plugin for Unity supports Unity 2019.4 LTS and 2020.3 LTS.

Highlights:

- Build, run, and modify a sample game with different scenarios, or create your own.
- Deploy sample AWS CloudFormation scenarios for typical game scenarios including auth only, single-Region fleet, multi-Region fleets with queue and custom matchmaker, Spot Fleets with queue and custom matchmaker, and FlexMatch.

Learn more:

- [Integrating games with the Amazon GameLift Servers plugin for Unity](#)

June 30, 2021: FlexMatch adds batchDistance rule

You can use the batchDistance rule type to specify a string or numeric attribute, bringing a host of benefits to each segment.

Highlights:

- For large matches (>40 players), instead of evenly balancing players by skill only, you can now get that same balance based on skill, modes, and maps. Ensure that everyone in the match is in a skill band, band multiple numeric attributes such as league or play style, and group according to string attributes such as map or game mode. You can also create expansions over time. For example, you can create an expansion to allow a greater skill level range to enter the match the longer the player is waiting.

For matches under 40 players, you can use a new simplified rules expression.

June 3, 2021: Amazon GameLift Servers realtime client SDK and server SDK updates

Updated SDK versions: Realtime Client SDK 1.2.0, Server SDK 3.4.0 for Unreal

With this latest SDK update, you can now integrate IL2CPP into your mobile applications that use the RTS Client SDK and follow best practices with frameworks. You can also now build the Amazon GameLift Servers Server SDK for Unreal Version 4.26. This update contains components that integrate with your Windows or Linux game server, including C++ and C# versions of the Amazon GameLift Servers Server SDK, Amazon GameLift Servers Local, and an Unreal Engine plugin.

Highlights:

- Added support for IL2CPP in the RTS Client SDK and for building the native libraries as frameworks, so you can build RTS clients for the latest mobile devices.
- You can use [DescribePlayerSessions\(\)](#) to get information for a single player session, for all player sessions in a game session, or for all player sessions associated with a single player ID.
- You can use [GetInstanceCertificate\(\)](#) to retrieve the file location of a PEM-encoded TLS certificate that is associated with the fleet and its instances.
- Created Server SDK support for Unreal version 4.26.
- The existing C# SDK, version 4.0.2, has been verified compatible with Unity 2020.3. No SDK updates were required.

Learn more:

- [Amazon GameLift Servers Developer Guide:](#)
 - [DescribePlayerSessions\(\)](#)
 - [GetInstanceCertificate\(\)](#)

March 23, 2021: Amazon GameLift Servers adds notifications to game session placement

Updated SDK versions: AWS SDK [1.8.168](#)

You can now use events to monitor game session placement activity for a game session queue. Create an Amazon Simple Notification Service (Amazon SNS) topic to publish event notifications, or set up event tracking using CloudWatch Events.

Highlights:

- For each queue, you can set a custom text string to be included in all event messaging.
- When using an Amazon SNS topic, you can set additional access conditions that limit publishing to specific queues.

Learn more:

- Amazon GameLift Servers Developer Guide:
 - [Set up event notification for game session placement](#) (new)
 - [Game session placement events](#) (new)

- [API reference \(AWS SDK\)](#)
 - New game session queue parameters `NotificationTarget` and `CustomEventData`:
[GameSessionQueue](#), [CreateGameSessionQueue](#), [UpdateGameSessionQueue](#)
- [Amazon GameLift Servers forum](#)

March 16, 2021: Amazon GameLift Servers adds multi-region fleets, six new regions

Updated SDK versions: AWS SDK [1.8.163](#)

Amazon GameLift Servers managed hosting is now available in 21 AWS Regions. The new Regions are Cape Town (af-south-1), Bahrain (me-south-1), Hong Kong (ap-east-1), Milan (eu-south-1), Paris (eu-west-3), and Stockholm (eu-north-1).

With the new Amazon GameLift Servers multi-location fleets feature, you can now set up a single fleet to host your game servers in any or all of 20 Amazon GameLift Servers-supported Regions (Beijing Region excepted). This feature aims to significantly reduce the work required to set up and maintain Amazon GameLift Servers hosting resources globally. Multi-location fleets can be created in the following AWS Regions: us-east-1 (N. Virginia), us-west-2 (Oregon), eu-central-1 (Frankfurt), eu-west-1 (Ireland), ap-southeast-2 (Sydney), ap-northeast-1 (Tokyo), and ap-northeast-2 (Seoul). In all other Regions, you can continue to set up single-location fleets as needed. All fleets that were created before this release are single-location fleets. Using multi-location fleets does not affect your hosting costs. Amazon GameLift Servers pricing is based on the type, location, and volume of instances that you use. (For more information, see [Amazon GameLift Servers pricing](#).) CloudFormation support for multi-location fleets will be available soon.

Note

Multi-location fleets are not available in the China Regions. Amazon GameLift Servers resources that reside in China Regions cannot interact with or be used by resources in other Amazon GameLift Servers Regions.

Highlights:

- With a multi-location fleet, explicitly add a list of remote locations. Amazon GameLift Servers deploys instances of the same type and configuration, including the build and runtime configuration, to the fleet's home Region and all added locations.

- Adjust capacity settings and scaling for each location independently. Auto scaling policies apply to an entire fleet, but you can turn them on or off by location.
- Start new game sessions at specific fleet locations. When using game session queues or matchmaking to place game sessions, you can now prioritize where new game sessions start by location, hosting cost, and player latency.
- Get hosting metrics in the Amazon GameLift Servers console, aggregated for all locations in a fleet or broken out by each fleet location.

Learn more:

- [Amazon game tech blog](#)
- [API reference \(AWS SDK\)](#)
 - New fleet location operations: [CreateFleetLocations](#), [DescribeFleetLocationAttributes](#), [DescribeFleetLocationCapacity](#), [DescribeFleetLocationUtilization](#), [DeleteFleetLocations](#)
 - Updated fleet operations, with new multi-location support: [CreateFleet](#), [UpdateFleetCapacity](#), [DescribeEC2InstanceLimits](#), [DescribeInstances](#), [StopFleetActions](#), [StartFleetActions](#)
 - Updated game session placement operations, with new priority and filtering capability: [CreateGameSessionQueue](#), [DescribeGameSessionQueues](#), [UpdateGameSessionQueue](#)
 - Updated game session creation operations, with new location support: [CreateGameSession](#), [DescribeGameSessions](#), [DescribeGameSessionDetails](#), [SearchGameSessions](#)
- [Amazon GameLift Servers Developer Guide](#):
 - [Amazon GameLift Servers service locations](#) (updated)
 - [Hosting resource customizations](#) (new)
 - [Scaling game hosting capacity with Amazon GameLift Servers](#) (updated)
 - [Customize a game session queue](#) (new)
 - [Fleet details in the Amazon GameLift Servers console](#) (updated)
- [Amazon GameLift Servers forum](#)

February 9, 2021: Amazon GameLift Servers extends support for AMD instances, standalone FlexMatch

Updated SDK versions: AWS SDK [1.8.139](#)

This release includes the following updates:

- Amazon GameLift Servers FleetIQ game server groups can now be configured to manage the AMD instance families C5a, M5a, and R5a. The supported Amazon EC2 instance types, as listed for the GameServerGroup [InstanceDefinition](#), now include the following:
 - c5a.large, c5a.xlarge, c5a.2xlarge, c5a.4xlarge, c5a.8xlarge, c5a.12xlarge, c5a.16xlarge, c5a.24xlarge
 - m5a.large, m5a.xlarge, m5a.2xlarge, m5a.4xlarge, m5a.8xlarge, m5a.12xlarge, m5a.16xlarge, m5a.24xlarge
 - r5a.large, r5a.xlarge, r5a.2xlarge, r5a.4xlarge, r5a.8xlarge, r5a.12xlarge, r5a.16xlarge, r5a.24xlarge

Note: AMD instances for FleetIQ are currently not available for use in the China (Beijing) AWS Region. See [Feature availability and implementation differences](#) in China.

- Amazon GameLift Servers managed game hosting now supports AMD instances in the China (Beijing) Region, operated by Sinnet. The new AMD instance families include M5a and R5a. Supported EC2 instance types, as listed for fleet [InstanceType](#), now include the following:
 - m5a.large, m5a.xlarge, m5a.2xlarge, m5a.4xlarge, m5a.8xlarge, m5a.12xlarge, m5a.16xlarge, m5a.24xlarge
 - r5a.large, r5a.xlarge, r5a.2xlarge, r5a.4xlarge, r5a.8xlarge, r5a.12xlarge, r5a.16xlarge, r5a.24xlarge
- Amazon GameLift Servers FlexMatch can now be used as a standalone matchmaking solution in the China (Beijing) Region, operated by Sinnet. Customers can create a FlexMatch matchmaker in the Beijing Region and configure the [FlexMatchMode](#) parameter to STANDALONE. For more information about FlexMatch, either with Amazon GameLift Servers managed hosting or with a non-Amazon GameLift Servers hosting solution, in the [Amazon GameLift Servers FlexMatch Developer Guide](#).
- When setting up event notifications for Amazon GameLift Servers FlexMatch, you can now designate an Amazon SNS FIFO topic as the notification target. For more information, see:
 - [MatchmakingConfiguration NotificationTarget](#), *Amazon GameLift Servers API Reference*
 - [Set up FlexMatch event notification](#), *Amazon GameLift Servers FlexMatch Developer Guide*
 - [Introducing Amazon SNS FIFO – First-in-first-out Pub/Sub messaging](#), *AWS News Blog*

December 22, 2020: Amazon GameLift Servers server SDK supports Unreal Engine 4.25 and Unity 2020

Updated SDK versions: Amazon GameLift Servers Server SDK 4.0.2, Unreal plugin version 3.3.3

The latest version of the Amazon GameLift Servers Server SDK contains the following components:

- The updated Unreal plugin has been updated for compatibility with Unreal Engine 4.25. The API was not changed.
- The existing C# SDK, version 4.0.2, has been verified compatible with Unity 2020. No SDK updates were required.

Download the latest version of the Amazon GameLift Servers Server SDK at [Amazon GameLift Servers getting started](#).

November 24, 2020: Amazon GameLift Servers FlexMatch now available for games hosted anywhere

Updated SDK versions: AWS SDK [1.8.95](#)

Amazon GameLift Servers FlexMatch is a customizable matchmaking service for multiplayer games. Initially designed for users of Amazon GameLift Servers managed hosting, FlexMatch can now be integrated into games that use other hosting systems, including peer-to-peer, proprietary on-premises computing, and cloud compute primitive types. Games that use Amazon GameLift Servers FleetIQ for game hosting on Amazon EC2 can now implement matchmaking with FlexMatch.

FlexMatch provides a robust matchmaking algorithm and rules language that gives you wide latitude to customize the matchmaking process so that players are matched together based on key player characteristics and reported latency. In addition, FlexMatch offers a matchmaking request workflow that supports features such as player parties, player acceptance, and match backfill. When you use FlexMatch with Amazon GameLift Servers managed hosting or Amazon GameLift Servers Realtime, the matchmaker automatically uses Amazon GameLift Servers to find hosting resources and start a new game session for newly formed matches. When using FlexMatch as a standalone service, the matchmaker delivers match results back to your game, which can then start a new game session using your hosting solution.

API operations for FlexMatch are part of the Amazon GameLift Servers service API, which is included in the AWS SDK and the AWS Command Line Interface (AWS CLI). This release includes these updates to support standalone matchmaking:

- The API resource `MatchmakingConfiguration` has the following changes:
 - New property, `FlexMatchMode` indicates whether the matchmaker is being used with Amazon GameLift Servers managed hosting or as standalone matchmaking.
 - Property `GameSessionQueueArns` is not required when `FlexMatchMode` is set to `standalone`.
 - These properties are not used with standalone matchmaking: `AdditionalPlayerCount`, `BackfillMode`, `GameProperties`, `GameSessionData`.
- The automatic backfill feature is not available with standalone matchmaking.

November 24, 2020: AMD instances now available on Amazon GameLift Servers

Updated SDK versions: AWS SDK [1.8.95](#)

The list of Amazon EC2 instance types supported by Amazon GameLift Servers now includes three new instance families: C5a, M5a, and R5a. These families consist of AMD compute-optimized instances that are powered by AMD EPYC processors running at frequencies up to 3.3. GHz. The AMD instances are x86 compatible; games that are currently running on Amazon GameLift Servers can be deployed to AMD instance types without alteration. The new instances are available in the following AWS Regions: US East (N. Virginia and Ohio), US West (Oregon and N. California), Central Canada (Montreal), South America (Sao Paulo), EU Central (Frankfurt), EU West (London and Ireland), Asia Pacific South (Mumbai), Asia Pacific Northeast (Seoul and Tokyo), and Asia Pacific Southeast (Singapore and Sydney).

The new AMD instances include:

- c5a.large, c5a.xlarge, c5a.2xlarge, c5a.4xlarge, c5a.8xlarge, c5a.12xlarge, c5a.16xlarge, c5a.24xlarge
- m5a.large, m5a.xlarge, m5a.2xlarge, m5a.4xlarge, m5a.8xlarge, m5a.12xlarge, m5a.16xlarge, m5a.24xlarge
- r5a.large, r5a.xlarge, r5a.2xlarge, r5a.4xlarge, r5a.8xlarge, r5a.12xlarge, r5a.16xlarge, r5a.24xlarge

Learn more:

- [Amazon game tech blog](#)
- [Amazon GameLift Servers instance pricing](#)
- [Amazon EC2 instances featuring AMD EPYC processors](#)
- [Amazon GameLift Servers forum](#)

November 11, 2020: Version update to Amazon GameLift Servers server SDK**Updated SDK versions:** Amazon GameLift Servers Server SDK 4.0.2

The new Server SDK version 4.0.2 fixes a known issue with the API operation `StartMatchBackfill()`. This operation now returns a correct response to a match backfill request.

The issue did not affect the match backfill process, and there is no change to how this feature works. The issue may have impacted log messaging and error handling for match backfill requests.

Download the latest version of the Amazon GameLift Servers Server SDK at [Amazon GameLift Servers getting started](#).

November 5, 2020: New FlexMatch algorithm customizations

FlexMatch users can now adjust the following default behaviors for the matchmaking process. These customizations are set in a matchmaking rule set. There are no changes to the Amazon GameLift Servers SDKs.

- **Prioritize backfill tickets:** You can choose to raise or lower how match backfill tickets are prioritized when searching for acceptable matches. Prioritizing backfill tickets is useful when the auto-backfill feature is enabled. Use the algorithm property `backfillPriority`.
- **Pre-sort to optimize match consistency and efficiency:** Configure your matchmaker to pre-sort the ticket pool before batching tickets for evaluation. By pre-sorting tickets based on key player attributes, your resulting matches tend to have players who are more similar in those attributes. You can also boost efficiency in the evaluation process by pre-sorting on the same attributes that are used in match rules. Use the algorithm property `sortByAttributes` with the `strategy` property set to "sorted".
- **Adjust how expansion wait times are triggered:** Choose between triggering expansions based on the age of the newest (default) or oldest ticket in an incomplete match. Triggering on the oldest

ticket tends to complete matches faster, while triggering on the newest ticket leads to higher match quality. Use the algorithm property `expansionAgeSelection`.

September 17, 2020: Amazon GameLift Servers updates server SDK

Updated SDK versions: Amazon GameLift Servers Server SDK 4.0.1

The new Server SDK contains the following updates:

- C# API version 4.0.1
 - The API operation [TerminateGameSession\(\)](#) is no longer supported. Replace with a call to [ProcessEnding\(\)](#) to end both a game session and the server process.
 - A known issue with the operation [GetInstanceCertificate\(\)](#) is fixed.
 - The operation [GetTerminationTime\(\)](#) now returns a value of data type `AwsDateTimeOutcome`.
- C++ API version 3.4.1
 - The operation [TerminateGameSession\(\)](#) is no longer supported. Replace it with a call to [ProcessEnding\(\)](#) to end both a game session and the server process.
- Unreal Engine plugin version 3.3.2
 - The operation [TerminateGameSession\(\)](#) is no longer supported. Replace it with a call to [ProcessEnding\(\)](#) to end both a game session and the server process.
 - The callback operation `OnUpdateGameSession` is added to [FProcessParameters](#) to support match backfill.

Download the latest version of the Amazon GameLift Servers Server SDK at [Amazon GameLift Servers getting started](#).

August 27, 2020: Amazon GameLift Servers FleetIQ for game hosting with Amazon EC2 (general availability)

Updated SDK versions: AWS SDK [1.8.36](#)

The Amazon GameLift Servers FleetIQ solution for low-cost, cloud-based game hosting on Amazon EC2 is now generally available. Amazon GameLift Servers FleetIQ gives developers the ability to host game servers directly on Amazon EC2 Spot Instances by optimizing their viability for game hosting. Game developers can use Amazon GameLift Servers FleetIQ with new games or to

supplement capacity for existing games. This solution supports the use of containers or other AWS services such as AWS Shield and Amazon Elastic Container Service (Amazon ECS).

This general availability release includes the following updates to the Amazon GameLift Servers FleetIQ solution:

- New API operation `DescribeGameServerInstances` returns information, including status, on all active instances for an Amazon GameLift Servers FleetIQ game server group.
- New balancing strategy, `ON_DEMAND_ONLY`, configures a game server group to use On-Demand Instances only. You can update a game server group's balancing strategy at any time, making it possible to switch between using Spot Instances and On-Demand Instances as needed.
- The following preview elements have been dropped for general availability:
 - Use of custom sort keys for game server resources. Game servers can be sorted based on registration timestamp.
 - Tagging for game server resources.

April 16, 2020: Amazon GameLift Servers updates server SDK for Unity and Unreal Engine

Updated SDK versions: Amazon GameLift Servers Server SDK 4.0.0, Amazon GameLift Servers Local 1.0.5

The latest version of the Amazon GameLift Servers Server SDK contains the following updated components:

- C# SDK version 4.0.0 updated for Unity 2019.
- Unreal plugin version 3.3.1 updated for Unreal Engine versions 4.22, 4.23, and 4.24.
- Amazon GameLift Servers Local version 1.0.5 updated to test integrations that use the C# server SDK version 4.0.0.

Download the latest version of the Amazon GameLift Servers Server SDK at [Amazon GameLift Servers getting started](#).

April 2, 2020: Amazon GameLift Servers FleetIQ available for game hosting on EC2 (public preview)

Updated SDK versions: AWS SDK [1.7.310](#)

The Amazon GameLift Servers FleetIQ feature optimizes the viability of low-cost Spot Instances for use with game hosting. This feature is now extended for customers who want to manage their hosting resources directly rather than through the managed Amazon GameLift Servers service. This solution supports the use of containers or other AWS services such as AWS Shield and Amazon Elastic Container Service (Amazon ECS).

Learn more:

[GameTech blog post](#) on Amazon GameLift Servers FleetIQ

December 19, 2019: Improved AWS resource management for Amazon GameLift Servers resources

Updated SDK versions: AWS SDK [1.7.249](#)

You can now take advantage of AWS resource management tools with Amazon GameLift Servers resources. In particular, all key Amazon GameLift Servers resources—builds, scripts, fleets, game session queues, matchmaking configurations, and matchmaking rule sets—are now assigned Amazon Resource Name (ARN) values. A resource ARN provides a consistent identifier that is unique across all AWS Regions. They can be used to create resource-specific AWS Identity and Access Management (IAM) permissions policies. Resources are now assigned an ARN and also the pre-existing resource identifier, which is not Region-specific.

In addition, Amazon GameLift Servers resources now support tagging. You can use tags to organize resources, create IAM permissions policies to manage access to groups of resources, customize AWS cost breakdowns, etc. When managing tags for Amazon GameLift Servers resources, use the Amazon GameLift Servers API actions `TagResource()`, `UntagResource()`, and `ListTagsForResource()`.

Learn more:

- [TagResource](#) in the *Amazon GameLift Servers API Reference*
- [Tagging AWS resources](#) in the *AWS General Reference*
- [Amazon resource names](#) in the *AWS General Reference*

November 14, 2019: New CloudFormation templates, updates in China (Beijing) Region

Updated SDK versions: AWS SDK [1.7.210](#)

CloudFormation templates for Amazon GameLift Servers

Amazon GameLift Servers resources can now be created and managed through CloudFormation. The existing CloudFormation build and fleet templates have been updated to align with the current resources, and new templates are now available for scripts, queues, matchmaking configurations, and matchmaking rule sets. CloudFormation templates greatly simplify the task of managing groups of related AWS resources, particularly when deploying games across multiple Regions.

Learn more:

- [Amazon GameLift Servers resource type reference](#) in the *AWS CloudFormation User Guide*
- [Manage Amazon GameLift Servers hosting resources using CloudFormation](#) in the *Amazon GameLift Servers Developer Guide*