### Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is Amazon GameLift?</td>
<td>1</td>
</tr>
<tr>
<td>Why Amazon GameLift?</td>
<td>1</td>
</tr>
<tr>
<td>Key Features</td>
<td>1</td>
</tr>
<tr>
<td>How GameLift Works</td>
<td>2</td>
</tr>
<tr>
<td>Key Components</td>
<td>2</td>
</tr>
<tr>
<td>Hosting Game Servers</td>
<td>2</td>
</tr>
<tr>
<td>Placing Game Sessions</td>
<td>3</td>
</tr>
<tr>
<td>Managing Capacity and Scaling</td>
<td>3</td>
</tr>
<tr>
<td>Monitoring Fleet Activity and Troubleshooting</td>
<td>5</td>
</tr>
<tr>
<td>Networking With AWS Resources</td>
<td>5</td>
</tr>
<tr>
<td>Game Architecture with GameLift</td>
<td>6</td>
</tr>
<tr>
<td>How Players Connect to Games</td>
<td>8</td>
</tr>
<tr>
<td>Game and Player Session Features</td>
<td>8</td>
</tr>
<tr>
<td>GameLift FlexMatch</td>
<td>10</td>
</tr>
<tr>
<td>How GameLift FlexMatch Works</td>
<td>11</td>
</tr>
<tr>
<td>Tools and Resources</td>
<td>13</td>
</tr>
<tr>
<td>Core Tools</td>
<td>13</td>
</tr>
<tr>
<td>Additional Resources</td>
<td>13</td>
</tr>
<tr>
<td>GameLift SDKs</td>
<td>14</td>
</tr>
<tr>
<td>For Game Servers</td>
<td>14</td>
</tr>
<tr>
<td>For Game Clients and Game Services</td>
<td>15</td>
</tr>
<tr>
<td>SDK Compatibility</td>
<td>15</td>
</tr>
<tr>
<td>Free Tier and Billing Alerts</td>
<td>16</td>
</tr>
<tr>
<td>Setting Up</td>
<td>17</td>
</tr>
<tr>
<td>Set Up Your Project</td>
<td>17</td>
</tr>
<tr>
<td>Set up an AWS Account</td>
<td>17</td>
</tr>
<tr>
<td>IAM Policy Examples</td>
<td>18</td>
</tr>
<tr>
<td>Install the AWS CLI</td>
<td>18</td>
</tr>
<tr>
<td>Integrating GameLift</td>
<td>20</td>
</tr>
<tr>
<td>Integration Plan</td>
<td>20</td>
</tr>
<tr>
<td>Game Engines and GameLift</td>
<td>22</td>
</tr>
<tr>
<td>Amazon Lumberyard</td>
<td>22</td>
</tr>
<tr>
<td>Unreal Engine</td>
<td>22</td>
</tr>
<tr>
<td>Unity</td>
<td>23</td>
</tr>
<tr>
<td>Other Engines</td>
<td>23</td>
</tr>
<tr>
<td>Lumberyard: Prep a Game Client</td>
<td>23</td>
</tr>
<tr>
<td>Unreal Engine: Add GameLift to a Game Server Project</td>
<td>24</td>
</tr>
<tr>
<td>Unity: Add GameLift to a Game Server Project</td>
<td>27</td>
</tr>
<tr>
<td>Integrating a Game Server</td>
<td>29</td>
</tr>
<tr>
<td>Add GameLift to a Game Server</td>
<td>30</td>
</tr>
<tr>
<td>Integrating a Game Client</td>
<td>33</td>
</tr>
<tr>
<td>Add GameLift to a Game Client</td>
<td>33</td>
</tr>
<tr>
<td>Create Game Sessions with Queues</td>
<td>36</td>
</tr>
<tr>
<td>Generate Player IDs</td>
<td>37</td>
</tr>
<tr>
<td>GameLift Interactions</td>
<td>37</td>
</tr>
<tr>
<td>Setting Up a New Server Process</td>
<td>37</td>
</tr>
<tr>
<td>Creating a Game Session</td>
<td>38</td>
</tr>
<tr>
<td>Adding a Player to a Game Session</td>
<td>38</td>
</tr>
<tr>
<td>Removing a Player From a Game Session</td>
<td>38</td>
</tr>
<tr>
<td>Shutting down a Game Session</td>
<td>38</td>
</tr>
<tr>
<td>Terminating a Server Process</td>
<td>39</td>
</tr>
<tr>
<td>Responding to a Shutdown Request</td>
<td>39</td>
</tr>
<tr>
<td>GameLift–Game Server/Client Interactions</td>
<td>40</td>
</tr>
<tr>
<td>Testing Your Integration</td>
<td>41</td>
</tr>
</tbody>
</table>
# Amazon GameLift Developer Guide

## Table of Contents

- Set Up Amazon GameLift Local ................................................................. 41
- Test a Game Server .................................................................................. 41
- Test a Game Server and Client ................................................................. 43
- Variations with Local ............................................................................... 45

### Uploading Your Game to GameLift

- Package Build Files .................................................................................. 46
- Creating an Install Script .......................................................................... 47
- Upload Build Files ..................................................................................... 47
- Create a Build with Files in a Directory .................................................. 48
- Create a Build with Files in Amazon S3 .................................................... 49
- Update Your Build Files ........................................................................... 51

### Working with Fleets

- Choose Computing Resources .................................................................. 52
- Operating Systems .................................................................................... 52
- Instance Types .......................................................................................... 53
- On-Demand versus Spot Instances .......................................................... 53
- Instance Service Limits ............................................................................. 54
- Create a Fleet ............................................................................................ 54
- Create a Fleet (Console) ............................................................................ 54
- Create a Fleet (AWS CLI) .......................................................................... 56
- Debug Fleet Creation Issues ...................................................................... 58
- Update a Fleet ........................................................................................... 59
- Delete a Fleet ............................................................................................ 60

### Run Multiple Processes

- Set Fleet Capacity Limits .......................................................................... 61
- Manually Set Fleet Capacity ...................................................................... 62
- Auto-Scale Fleet Capacity ......................................................................... 64
- Create a Fleet (AWS CLI) .......................................................................... 56

### Spot Fleet Integration Guide

- Auto-Scale Fleet Capacity ......................................................................... 64
- Manually Set Fleet Capacity ...................................................................... 62
- Set Fleet Capacity Limits .......................................................................... 61

### Instance Service Limits

- On-Demand versus Spot Instances .......................................................... 53
- Instance Types .......................................................................................... 53
- Operating Systems .................................................................................... 52

### Instance Types

- Operating Systems .................................................................................... 52
- Instance Types .......................................................................................... 53
- On-Demand versus Spot Instances .......................................................... 53
- Instance Service Limits ............................................................................. 54

### Debug Fleet Creation Issues

- Create a Fleet (AWS CLI) .......................................................................... 56
- Debug Fleet Creation Issues ...................................................................... 58

### Remotely Access Fleet Instances

- Connect to an Instance ............................................................................. 72
- View and Update Remote Instances .......................................................... 74
- Set Up VPC Peering ................................................................................... 74

### Working with Aliases

- Create an Alias .......................................................................................... 77
- Edit an Alias ............................................................................................. 77

### Working with Queues

- Design a Queue .......................................................................................... 79
- Why Use Queues? ....................................................................................... 79
- Tips on Setting Queue Destinations .......................................................... 80
- Design a Multi-Region Queue ..................................................................... 80
- Design Player Latency Policies ................................................................... 81
- Design a Queue for Spot Instances ........................................................... 82
- How Perfect Placement Works ................................................................. 82
- Create a Queue (Console) ......................................................................... 83
- Create a Queue (AWS CLI) ....................................................................... 84

### View Your Queues

- View Queue Details ................................................................................... 86
- Summary .................................................................................................... 86
- Destinations ............................................................................................... 86
- Player Latency Policies .............................................................................. 87
- Queue Metrics ............................................................................................ 87

### Matchmaking with FlexMatch

- Matchmaking with FlexMatch .................................................................... 89
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server API (C++) Reference</td>
<td>137</td>
</tr>
<tr>
<td>Actions</td>
<td>137</td>
</tr>
<tr>
<td>Data Types</td>
<td>148</td>
</tr>
<tr>
<td>Server API (C#) Reference</td>
<td>152</td>
</tr>
<tr>
<td>Actions</td>
<td>153</td>
</tr>
<tr>
<td>Data Types</td>
<td>162</td>
</tr>
<tr>
<td>Server API (Unreal Engine) Reference</td>
<td>166</td>
</tr>
<tr>
<td>Actions</td>
<td>167</td>
</tr>
<tr>
<td>Data Types</td>
<td>171</td>
</tr>
<tr>
<td>FlexMatch Rules</td>
<td>172</td>
</tr>
<tr>
<td>Rule Types</td>
<td>173</td>
</tr>
<tr>
<td>Property Expressions</td>
<td>175</td>
</tr>
<tr>
<td>FlexMatch Events</td>
<td>177</td>
</tr>
<tr>
<td>Document History</td>
<td>188</td>
</tr>
</tbody>
</table>
What is Amazon GameLift?

Amazon GameLift is a fully managed service for deploying, operating, and scaling your session-based multiplayer game servers in the cloud. Amazon GameLift replaces the work required to host your own game servers, including buying and setting up hardware, and managing ongoing activity, security, storage, and performance tracking. Amazon GameLift's auto-scaling features provide additional protection from having to pay for more resources than you need, while helping to ensure that your players can find and join games fast.

Why Amazon GameLift?

Here are some of the benefits of using Amazon GameLift:

- Provide low-latency player experience to support fast-action game play.
- Release session-based, multiplayer games fast, with little or no back-end experience required.
- Enhance your matchmaking services with intelligent queuing, game session placement, and backfill.
- Reduce engineering and operational effort to deploy and operate game servers.
- Get started fast and pay as you go, with no upfront costs and no long-term commitments.
- Reduce costs by up to 90% with spot instances.
- Rely on Amazon Web Services (AWS), including Amazon Elastic Compute Cloud (Amazon EC2) for web-scale cloud computing resources and auto-scaling to manage your hosting capacity.

Key Features

Amazon GameLift includes these features:

- Use FlexMatch and Queues to build a custom matchmaking service for your game.
- Use game session queues to place new game sessions more effectively across Amazon GameLift resources.
- Deploy game servers to run on either Amazon Linux or Windows Server operating systems.
- Provide high-quality game hosting to players around the world by deploying to computing resources in multiple regions.
- Let FleetIQ optimize the use of spot instances to reduce hosting costs by using spot instances with low prices and low interruption frequencies.
- Use auto-scaling tools to adjust your game hosting capacity to meet actual player usage and balance player experience against cost savings.
- Configure game session characteristics, such as maximum number of players allowed, join rules, and game-specific properties.
- Help players find sessions to join quickly with game session search features.
- Analyze game performance using the Amazon GameLift console to track metrics, view game session logs, and review data on individual game sessions and player sessions.
• Set up customized health tracking for server processes to detect problems fast and resolve poor-performing processes.

For more product information about Amazon GameLift, including specifications and pricing, see the Amazon GameLift product pages.

How Amazon GameLift Works

This topic provides an overview of Amazon GameLift components and how the service works to deploy your multiplayer game servers and manage player traffic.

Key Components

Setting up Amazon GameLift to host your game involves working with the following components:

• A **game server** is your game’s server software running in the cloud. You upload a **game server build** to Amazon GameLift, including the server executables, supporting assets, libraries, and dependencies. Amazon GameLift deploys your game server to virtual computing resources for hosting.

• A **game session** is a instance of your game server, running on Amazon GameLift, that players connect to and interact with. A game defines the basic characteristics of a game session, such as its life span or number of players.

• The **Amazon GameLift service** manages the computing resources needed to host your game server and makes it possible for players to connect to games. It regulates the number of resources for player demand, starts and stops game sessions, and handles player join requests by finding and reserving player slots in active game sessions. The service also collects performance data on server process health and player usage.

• A **game client** is your game’s software running on a player’s device. Using the game client, a player can connect to a game session that is being hosted on Amazon GameLift.

• **Game services** might communicate with the Amazon GameLift service for a variety of purposes. For example, you might create a game service to act as an intermediary between game clients and servers, such as to manage matchmaking or player authentication.

See Amazon GameLift and Game Client/Server Interactions (p. 37) for a detailed description of how these components interact.

Hosting Game Servers

To host a game server on Amazon GameLift, you need a **fleet** of virtual computing resources, called **instances**. The instance type you choose for a fleet determines the computing hardware (including power, memory, networking capacity, etc.) that will be used to host your game servers. Learn more about how to Choose Computing Resources (p. 52), including how to work with spot instances to reduce hosting costs.

Each instance in a fleet can run multiple **server processes** simultaneously, depending on the hardware capability, and each server process can host one game session. Since a game server build can have one or multiple executable files, you can configure a fleet to run multiple server processes of each executable on each instance. You’ll need to find the right configuration to balance the computing requirements of your game server—and the number of server processes to run—against the capabilities of the instance type you choose. Learn more about running multiple processes on a fleet (p. 70).

A fleet deploys your game server build to a single AWS **region**. For each region where you want to deploy your game server build, you must set up a separate fleet. See a list of available regions for Amazon GameLift at AWS Regions and Endpoints.
You may want to assign an alias to a fleet. An alias is a convenient way to genericize how game clients connect to game servers. Because an alias can be changed to point to any fleet you want, referencing an alias ID in your client instead of a fleet ID helps you gracefully transition players from one fleet to another—without having to deploy game client updates.

### Placing Game Sessions

Once a game server build is successfully deployed to a fleet, the fleet is ready to host game sessions. A game client or client service sends a request, on behalf of one or more players, for a new game session. When a request for a new game session is received, Amazon GameLift's FleetIQ feature searches available hosting resources and places the new game session with the best possible fleet.

You define what "best possible fleet" means for your game by creating a game session queue. A queue identifies a list of fleets where new game sessions can be placed and defines how to choose the best fleet for each new game session. When your game client or client service requests a new game session, it specifies which queue to use for placement. The new game session can be placed with any fleet in the queue that has an available server process. Queues can contain fleets that are located in different regions. For example, you might set up a queue to place game sessions in any of the five North American regions. With a multi-region queue, Amazon GameLift is more likely to find available resources, even during heavy traffic, and start up a new game session quickly.

There are two ways that FleetIQ locates the best possible placement for a new game session. The most effective way is by evaluating player latency. Placement requests may include players' latency data for each region covered in the queue. If latency data is provided, FleetIQ searches for hosting availability in the region that offers the lowest latency for the request's players. If you opt not to provide latency data, you can manually prioritize a queue's fleet list. When searching for hosting availability, Amazon GameLift starts at the top of the list and works down. In this scenario, game sessions are usually placed with the top-priority fleet, with additional fleets acting as backup.

The queue is a powerful concept that can be used to solve a range of issues. Queues can balance resource usage across regions and reduce the amount of wait time for players during unexpected spikes in demand. They can be used to mitigate region slowdowns or outages. You can also use them to create player pools that span regions. Queues are required when using FlexMatch matchmaking or Amazon GameLift spot fleets. Learn more about how to Design a Game Session Queue (p. 79).

### Managing Capacity and Scaling

Once a fleet is active and ready to host game sessions, you'll need to adjust fleet capacity to meet player demand. Since the cost of hosting is based on the amount of capacity you use, it is important to find a balance between maintaining enough fleet capacity for incoming players and overspending for resources that sit unused during idle times.

Fleet capacity is measured in instances. A fleet's configuration determines how many game sessions and players each instance can host. To adjust fleet capacity, you increase or decrease the number of instances in the fleet. Amazon GameLift provides a range of options for managing fleet capacity. The most effective method, by far, is to use Amazon GameLift's auto-scaling feature, but you can also set fleet capacity manually. Learn more about how to Manage Fleet Capacity (p. 60).

### Auto-scaling

Auto-scaling is a fast, efficient, and accurate way to match fleet capacity to player usage. With auto-scaling, Amazon GameLift tracks the fleet's hosting metrics and determines when to add or remove instances based on a set of guidelines, called policies, that you define. With the right auto-scaling policies in place, Amazon GameLift can adjust capacity directly in response to changes in player demand, so that the fleet always has room for new players without maintaining an excessive amount of idle resources. Learn more about improving cost efficiency with automatic scaling.
There are two types of auto-scaling available, target-based and rule-based. The recommended option is to use target-based auto-scaling as the simplest and most effective option. Rule-based auto-scaling provides more fine-grained control over scaling actions, but is difficult to set up and manage. For most games, target tracking is sufficient.

Target-based auto-scaling allows you to select a desired outcome and have Amazon GameLift scale the fleet up or down to achieve that outcome. The Target Tracking tool is based on a single key metric—the percentage of resources that are available to host game sessions but are currently unused. These resources are your buffer—since they are ready to host games, they represent the number of new game sessions and new players that can join you game with minimal waiting. Target tracking lets you choose a buffer size, as a percentage of total fleet capacity, that best fits your game. For example, if demand for your game is highly volatile, you may want to use a larger buffer size. With target tracking on, Amazon GameLift adds and removes instances as needed to maintain the target buffer size. For most games, target tracking represents the best option for managing fleet capacity. Learn how to Auto-Scale with Target Tracking (p. 64).

Rule-based auto-scaling provides more fine-grained control over auto-scaling activity, but it is significantly more complicated to manage. Each rule-based policy specifies when and how to trigger a scaling event. For example, this policy statement might be used to trigger a scale-up event: "If idle instances falls below 20 for 10 consecutive minutes, then increase capacity by 10%." With rule-based auto-scaling, most fleets require multiple policies to automatically maintain fleet capacity. Since policies are evaluated independently and can have unexpected compound effects, this option adds greater complexity to game hosting. Learn how to Auto-Scale with Rule-Based Policies (p. 66).

Fleet scaling in action

A fleet scaling event can be triggered in several ways, either by making a change to the desired capacity through auto-scaling or manual scaling, or when instances are shut down for health or other reasons. Essentially, all scaling events are triggered when a fleet’s "desired" instance count does not match its "active" instance count. This circumstance causes Amazon GameLift to add or remove instances, as needed, in order to make the active instance count match the desired instance count.

- When desired instance count exceeds active instance count, Amazon GameLift requests additional instances and, once available, begins the process of installing the game server build to the new instances and starting up the game server processes. As soon as one server process is active on an instance, the number of active instances is increased by one. Amazon GameLift continues to add instances until the two count values are even.
- When active instance count exceeds desired instance count, Amazon GameLift begins searching for instances it can terminate. Any available instance (that is, not hosting any game sessions) can be terminated, as well as any non-protected instance even when hosting active game sessions. If no instances can be terminated, the scale-down event fails. In this circumstance, the disparity between desired and active instance counts will continue to trigger scale-down events until an instance is free to terminate. Amazon GameLift then starts the termination process, which includes notifying all server processes on the instance to initiate a graceful shutdown. Once the instance is terminated, the number of active instances is decreased by one. Amazon GameLift continues to add instances until the two count values are even.

Additional scaling features

Additional features related to fleet capacity and scaling include:

- **Game session protection** – Prevent game sessions that are hosting active players from being terminated during a scale-down event. Game session protection can be turned on fleet-wide, or it can be turned on for individual game sessions. An instance cannot be terminated if any of its server processes are hosting protected game sessions. Game sessions are not protected from termination due to health or for spot-instance-related interruptions (see On-Demand versus Spot Instances (p. 53)).
Monitoring Fleet Activity and Troubleshooting

Once you have fleets up and running, Amazon GameLift collects a variety of information to help you monitor the performance of your deployed game servers. Use this information to optimize your use of resources, troubleshoot issues, and gain insight into how players are active in your games.

- **Fleet, game session, and player session details** – This data includes status, which can help identify health issues, as well as details such as game session length and player connection time.
- **Utilization metrics** – Amazon GameLift tracks fleet metrics over time:
  - For instances: network activity and CPU utilization
  - For server processes: number of active processes, new activations, and terminations
  - For games and players: number of active game sessions and player sessions
- **Server process health** – Amazon GameLift tracks the health of each server process running on a fleet, including the number of healthy processes, percent of active processes that are healthy, and number of abnormal terminations.
- **Game session logs** – You can have your game servers log session data and set Amazon GameLift to collect and store the logs once the game session ends. Logs can then be downloaded from the service.

All of this data is available through the Amazon GameLift console. The console dashboard presents an overview of activity across all you builds and fleets as well as the option to drill down to more detailed information.

Networking With AWS Resources

If your game server communicates with other resources that are hosted on AWS, you can use Amazon Virtual Private Cloud (VPC) peering to establish fast and secure connections. An Amazon VPC is a virtual network, defined by you, that includes a set of resources managed with your AWS account. In addition, each Amazon GameLift fleet, including the game server processes running on each instance in the fleet, has its own VPC. With VPC peering, you can establish a direct network connection between a VPC for your game server processes and a VPC for your other AWS resources.

For example, you might have a set of web services that support your game, such as for player authentication or social networking. You can set up a VPC for these resources, and then use Amazon GameLift’s VPC peering feature to enable your game servers to make direct network calls to the web services. With VPC peering, calls from your game server processes incur minimal latency and, since they are not routed over the public Internet, are not exposed externally.
See more information on Amazon’s virtual private clouds and VPC peering. You can peer your Amazon GameLift fleets with VPCs in any AWS account that you have access to.

Amazon GameLift 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 more information on how to set up VPC peering for your game servers, see Set Up VPC Peering (p. 74).

Game Architecture with Amazon GameLift

The diagram shown below illustrates the key components of an Amazon GameLift-hosted game and how they interact.
Key components are described as follows.

**Amazon GameLift service**

The service deploys your game servers to Amazon GameLift hosting resources and controls capacity based on your settings. It manages player demand by starting new game sessions and reserving game slots for new players. It continually tracks the status of game resources, game sessions, and players in order to maintain game availability and balance player traffic.

**Game clients**
Your game client may interact with Amazon GameLift in two ways:

- It communicates with the Amazon GameLift service to retrieve information on current game sessions, request new game sessions, and request player slots in games. Game clients can communicate directly with the service (as shown by a dotted line), but it is common to have game services handle this communication as part of other core tasks and then relay information as needed to the game client.
- It connects directly to a game session running on a game server. A game session’s connection details are provided in response to a player slot request. If you use a game service to handle player requests to the Amazon GameLift service, it must relay the connection information to the game client.

**Game services**

Your game may choose to handle certain tasks through game services rather than in the game client. These services are developed and managed by you. Game services are often used for tasks like player authentication and authorization, team building and matchmaking, and inventory or currency control. If your game uses services, it may make sense to call the Amazon GameLift service from a game service instead of from the game client. For example, when a player wants to join a game, your game client might call your authentication service to first verify the player’s identity, and only then send a player slot request to the Amazon GameLift service.

**Game servers**

Your game server software is deployed by Amazon GameLift based on your configuration. You can tell Amazon GameLift to deploy one or multiple game servers in as many AWS regions as you need. Game servers communicate with the Amazon GameLift service to start new game sessions, validate newly connected players, and report on the status of game sessions, player connections, and available resources.

**External services**

Your game may rely on an external service, such as for validating a subscription membership. The architecture diagram illustrates how information from an external service can be passed to your game servers (via a custom game service and the Amazon GameLift service) without going through the game client.

**Developer tools**

The Amazon GameLift developer tool set is used to configure your hosting resources and game deployment, scale capacity, monitor current status of resources, and track metrics on game server performance and game and player activity.

**How Players Connect to Games**

A *game session* is an instance of your game running on Amazon GameLift. To play your game, players can either find and join an existing game session or create a new game session and join it. Players join by creating a *player session* for the game session. If the game session is open for players—that is, it is accepting new players and has an open player slot—Amazon GameLift reserves a slot for the player and provides connection information back to the player. The player can then connect to the game session and claim the reserved slot.

For detailed information on creating and managing games sessions and player sessions, see *Add Amazon GameLift to Your Game Client* (p. 33).

**Game and Player Session Features**

Amazon GameLift provides several features related to game and player sessions:
Host game sessions on best available resources across multiple regions

Choose from multiple options when configuring how Amazon GameLift selects resources to host new game sessions. If you’re running multiple fleets in more than one region, you can set up game session queues that can place a new game session on any fleet regardless of region. This feature can significantly improve Amazon GameLift’s ability to efficiently balance resource usage and respond to changes in player demand, decreased capacity, outage events, and other issues. As a result, using queues can decrease the manual overhead needed to monitor and balance resources. You can manage queues and track queue performance metrics in the Amazon GameLift Console.

With the queues feature, you have the option of placing game sessions based on player latency information. This feature is particularly effective when supporting a matchmaking service. Requests for a new game session can also request new player sessions for one or more players. If you include latency data for each player by region, Amazon GameLift can choose a fleet in a region that provides the best possible experience for all the players.

Control player access to game sessions

Set a game session to allow or deny join requests from new players, regardless of the number of players currently connected. You might use this feature to enable private sessions, to limit access for troubleshooting or other problem resolution, etc.

Add custom game and player data

You can add custom data to game session and player session objects, which contain all the settings and metadata for a session. Custom data is stored with Amazon GameLift and can be retrieved by other components as needed. The Amazon GameLift service passes game session data to a game server when starting a new game session, and passes player session data to the game server when a player connects to the game session. Custom game and player data is not used by Amazon GameLift; it can be formatted as needed for use by your game client, game server, or other game services.

Game data may be useful for a variety of reasons. For example, when matching prospective players to game sessions, your game might use game properties to inform a best-match algorithm or help players choose from a list of game sessions. Alternatively, you might use game properties to pass information that a game server needs when setting up a new game session, such as a game mode or map.

Player data has a range of uses as well. For example, a matchmaking service might use player data to select a best match or team placement. A game server might customize a player’s experience based on their guild membership. Another use for player data is to provide secure information to a game server from a source other than a game client; for example, when communication between game client and game server is encrypted, such as with certain devices, encryption keys can be passed directly to the game server.

Filter and sort available game sessions

Use session search and sort to find the best match for a prospective player or allow players to browse a list of available game sessions. With this feature, you can effectively lead players to sessions that are most likely to result in a positive gaming experience. For example, if your game requires a minimum number of players, directing new players into nearly-filled games will minimize wait time for all players. Alternatively, you’ll likely want to hide sessions that are nearly finished. Session search can be very useful for implementing a “join now” feature backed by a well-formulated search and sort expression that gets players into positive gaming experiences fast. Use session search and sort to find game sessions based on characteristics like session age, available player slots, current player count, maximum players allowed, and custom game session data. You can also search and sort based on your own custom game data.
Track game and player usage data

Have Amazon GameLift automatically store logs for completed game sessions. Set up log storage when integrating Amazon GameLift into your game servers. You can access stored logs by downloading them through the Amazon GameLift console or programmatically with the AWS SDK for Amazon GameLift.

Use the Amazon GameLift console to view detailed information on game sessions, including session metadata and settings as well as player session data. For each game session, you can view a list of player sessions along with total times played. You can also view metrics data and graphs that track the number of active game sessions and player sessions over time. See more information at View Data on Game and Player Sessions (p. 122) and Metrics (p. 119).

Amazon GameLift FlexMatch

FlexMatch is Amazon GameLift’s customizable matchmaking service. It offers flexible tools that let you manage the full matchmaking experience in a way that best fits your game. You can use FlexMatch to build game matches, select compatible players for each match, and find the best available hosting resources for an optimum player experience. You can also use FlexMatch backfill to select compatible new players for existing games, ensuring that your games stay full and deliver the best possible player experience.

With FlexMatch you can create and run multiple matchmakers to fit your game modes and your players. For example, you would likely use a different matchmaker for each game mode, such as team battle, free-for-all, or cage match.

Key FlexMatch Features

- **Customize player matching.** Design and build the types of multiplayer experiences that your players will find most compelling. For each game mode, define the team structure and set other game attributes. Build a set of custom rules to evaluate player attributes (such as skill level or role) and form the best possible player matches for a game. Use these rules to group players for new matches or find players to fill open slots in existing matches (“match backfill”).

- **Get player acceptance.** Require all players to accept a proposed match before starting. If this feature is enabled, FlexMatch waits for all players assigned to a match to accept it before the match begins.

- **Support player parties.** Generate matches for a group of players who want to play together on the same team. Find additional players to fill out the match as needed.

- **Match players based on latency.** Use player latency information to ensure that matched players have similar response times. This feature avoids disparities in gameplay that might give some players undue advantage. It is particularly valuable when creating matches that span multiple geographic areas.

- **Relax player matching rules over time.** Strike the right balance between creating the best possible player matches and getting players into good matches fast. You decide where and when to relax strict matching rules in order to get players into games with minimal wait time.

- **Find the best hosting resources.** Use game and player information to select the best available resources to host the match for optimal gameplay experience.

- **Keep games filled with matched players.** Use FlexMatch backfill to fill empty player slots with well-matched new players throughout the life span of the game session.

Topics

- How Amazon GameLift FlexMatch Works (p. 11)
- Matchmaking with FlexMatch (p. 89)
How Amazon GameLift FlexMatch Works

This topic describes the core FlexMatch components and how FlexMatch processes matchmaking requests. For detailed help with adding FlexMatch to your game, including how to set up a matchmaker and customize player matching, see Matchmaking with FlexMatch (p. 89).

Amazon GameLift FlexMatch requires these three key components to work together:

- **Mechanisms to trigger player matchmaking.** One mechanism determines when to initiate matchmaking for players. A second (optional) mechanism determines when to find new players for empty slots in an existing match (backfilling). Matchmaking and match backfill requests are handed to a matchmaker for processing.

- **FlexMatch matchmaker to evaluate players and create matches.** A matchmaker builds the best possible player matches from the requests it receives. It has a rule set that defines a match's team structure and sets the criteria to use when evaluating players for a match. A game can have multiple matchmakers, with each building a different type of match.

- **Game session queue to place new matches.** A game session queue finds available computing resources to host a match. It determines where (in what regions) to look for resources and how to select the best available host for each match.

The following sections detail how matchmaking proceeds to form new game matches or to find new players for existing game matches.

**Matchmaking Requests**

Here's how requests for a new game match are handled with FlexMatch. This description assumes that a client-side game service is initiating matchmaking requests and tracking the matchmaking ticket status.

1. **Request matchmaking.** Players take some action in your game that triggers matchmaking, such as clicking a "Join Now" button or a group of players forming a party. Your game initiates a matchmaking request, identifying which matchmaker to use and including one or more players to be matched. The request includes any player information, such as skill level or preferences, that the matchmaker requires to build matches. Each request gets a matchmaking ticket ID, which your game uses to track the request status and take action as needed.

2. **Discover potential matches.** All matchmaking tickets are passed to the specified matchmaker and placed in its ticket pool for processing. A ticket remains in a ticket pool until it is matched or until it reaches the matchmaker's maximum time limit.

   To find player matches, the matchmaker makes continual passes through the ticket pool. On each pass, the matchmaker starts with the oldest ticket in the pool and evaluates the other tickets against it to find the best possible matches. A matchmaker's rule set determines (1) how many teams to create for a match, (2) the number of players to assign to each team, and (3) how to evaluate each prospective player. Rules might set requirements for individual players, teams, or matches. For example, a rule might require all matched players to have a certain talent, or it might require at least one player on a team to play a certain character. A commonly used rule requires that all players in a match have similar skill ratings.

   When the matchmaker evaluates a ticket, it either passes or fails the ticket based on the rules. For tickets with multiple players, the matchmaker assumes these players want to player together and attempts to place them all in the same match. This means that, for any potential match, all the players in a ticket must be acceptable. If any player fails any rule, the entire ticket is considered not a match. Tickets that fail remain in the ticket pool and are evaluated again on the next pass. Once a potential match is filled, the status of all tickets in the match are updated.

3. **Get player acceptance.** If the matchmaker requires players to accept a potential match, FlexMatch cannot proceed with the match until every player accepts. The matchmaking ticket status is changed...
to indicate that acceptance is required, which prompts your game to request acceptances from all
players in each matched ticket.

Players can choose to accept or reject a potential match. Your game collects the player responses and
reports them back to FlexMatch. All players in the potential match must accept the match within a
certain time limit to continue. If any player rejects the match or fails to respond before the time limit,
the matchmaker drops the potential match. Tickets for players who accepted the match are returned
to the matchmaker’s ticket pool; tickets for players who did not accept the match move to a failure
status and are no longer processed.

4. Find resources to host the match. Once a potential match is made and accepted, FlexMatch tries
to place the match with available hosting resources. The matchmaker is configured to use a specific
game session queue, and it passes the potential match to that queue for placement. The queue uses a
set of rules to search regions and fleets for the best available server process to host the match. If the
original matchmaking request contained player latency data, the queue uses this information to find
resources that offer the lowest latency and most consistent gameplay experience for players in the
match.

Once an available server process is located, Amazon GameLift creates a game session record with
game properties and matchmaker data, including team structure and sizes, player assignments, and
relevant player characteristics.

5. Start a new game session. As when starting any new game sessions, Amazon GameLift sends a start
request to the server process along with the game session and matchmaker information. The server
process takes the information and uses it to start a new game session for a matched game. When the
game session is ready to accept players, the server process notifies Amazon GameLift.

6. Connect players to the new game session. Once the game session is ready for players, Amazon
GameLift creates new player sessions for every player in the match. It then updates all matchmaking
tickets, changing the ticket status to indicate success and adding connection information for all
players. This change in ticket status, prompts your game to relay the connection information to game
clients. Players can now join the game and claim their slots in the match and their team assignments.

Backfill Requests

Here’s how finding new players for an existing match is handled with FlexMatch. Since match backfill
requires up-to-date information on player slot availability in game sessions, we recommend initiating
match backfill requests from the game server. Another option is to use a client-side game service, such as
a session directory service, that tracks game session and player activity. See more on adding the match
backfill feature to your game at Backfill Existing Games with FlexMatch (p. 112).

1. Request backfill matchmaking. A matched game has empty player slots that need to be filled. Your
game initiates a backfill request, identifying which matchmaker to use and describing the current
players in the game session. Each request has a matchmaking ticket ID, which your game uses to track
the request status and take action as needed.

2. Discover potential matches. Matchmaking tickets for backfills are passed to the specified
matchmaker and placed in the same pool as tickets for new matches. The matchmaker’s evaluation of
players is essentially identical whether the players looking for a match or are currently in a match that
is being backfilled. The one exception is that a potential match cannot have more than one backfill
ticket.

A backfill ticket must be matched with at least one other ticket in order to complete successfully, even
when the matchmaker’s rules allow a match to complete with empty player slots. Once a potential
match is filled, the status of all tickets in the match is updated.

3. Get player acceptance. If acceptance is required, only the new players need to accept a backfill match,
and this step is handled as described for matchmaking requests. The current players do not need to
accept a match that they’re already playing in. As a result, even though the backfill request’s ticket
status indicates that acceptance is required, your game does not need to take action.
If any of the proposed new players fails to accept the match within the time limit, the potential match is dropped and no new players are added to the existing match. When this happens, the ticket for the backfill request returns to the ticket pool for processing.

4. **Update existing game session with new match data.** When a backfill match is successfully made there is no need to place a new game session. Instead, Amazon GameLift updates the match data for the existing game session, adding the new players and team assignments. Amazon GameLift sends the updated game session information to the server process that is hosting the existing game.

5. **Connect new players to the existing game session.** Amazon GameLift creates player sessions for the new players and updates the matchmaking tickets with current status, player sessions, and connection information. Your client game service, which is tracking ticket status of the new players, relays the connection information to the game clients. Players can now join the existing game and claim their player slot.

## Tools and Resources

Amazon GameLift provides a collection of tools and resources for you to use.

### Core Tools

Use these tools to work with Amazon GameLift.

**Amazon GameLift SDKs**

The Amazon GameLift SDKs contain the libraries needed to communicate with the Amazon GameLift service from your game clients, game servers and game services. Versions of these SDKs are available with Lumberyard or you can download the latest versions separately. See details in Amazon GameLift SDKs (p. 14).

**AWS console for Amazon GameLift**

Use the AWS Management Console for Amazon GameLift to manage your game deployments, configure resources, and track player usage and performance metrics. The Amazon GameLift console provides a GUI alternative to managing resources programmatically with the AWS SDK.

**AWS CLI for Amazon GameLift**

Use this command line tool to make calls to the AWS SDK, including the Amazon GameLift API. Download the AWS Command Line Interface or view the AWS CLI Command Reference for Amazon GameLift.

**Amazon GameLift Local**

This client-side debugging tool emulates a subset of the Amazon GameLift API on your local development machine. You can test iterative code changes without needing to upload and run your game server on Amazon GameLift instances. Amazon GameLift Local can be used on Windows and Linux devices to test game clients and servers that use the Amazon GameLift SDKs. Amazon GameLift Local is available in the Server SDK download. See details in Testing Your Integration (p. 41).

### Additional Resources

Use these resources to learn and experiment with Amazon GameLift for your multiplayer games.

**Five-click sample**

Get a sample multiplayer game up and running on Amazon GameLift in under an hour. With this tool, you can start experimenting with Amazon GameLift tools in real time with minimal setting
Amazon GameLift Developer Guide
GameLift SDKs

You can find this tool in the Amazon GameLift console; from the intro page click "Test Amazon GameLift", or from any other page select "Sample game" from the Amazon GameLift navigation menu.

Getting Started tutorials

Use these tutorials to walk through the process of getting a sample multiplayer game up and running on Amazon GameLift. Once you complete the series, you can use the game to explore other Amazon GameLift features and tools, such as auto-scaling and performance metrics (no charge if you use the Amazon GameLift free tier).

Amazon GameLift forum

Use the Amazon GameLift forum to exchange ideas and knowledge, pick up tips, and get help with issues.

Amazon Lumberyard game engine

Amazon Lumberyard comes with the Amazon GameLift SDKs built in, and integration is handled automatically. It is bundled with a sample multiplayer game that illustrates how to use Lumberyard to integrate a game with Amazon GameLift. Find more information about this sample project in the Lumberyard User Guide.

GameDev blog

Watch this game development blog to keep up with new features about Amazon GameLift, Amazon Lumberyard, learn more about game development with AWS, and get expert tips from the teams.

Amazon GameLift product information

Check these pages for detailed descriptions of Amazon GameLift, service overview, FAQ, and pricing details.

Amazon GameLift SDKs

Use Amazon GameLift software development kits (SDKs) to develop Amazon GameLift-enabled multiplayer game servers, game clients and game services that need to connect to the Amazon GameLift service.

For detailed information on using the Amazon GameLift SDKs with your game engine, see Game Engines and Amazon GameLift (p. 22).

For Game Servers

Create and deploy 64-bit game servers with the Amazon GameLift Server SDK. This SDK enables the Amazon GameLift service to deploy and manage game server processes across Amazon GameLift virtual resources. To use the Amazon GameLift Server SDK in your game project, download the Server SDK and review the API documentation (p. 30).

SDK support

The Amazon GameLift Server SDK download contains source for the following versions:

- C++
- C++ for Unreal Engine (plugin)
- C# (.NET)

Development environments
Build the SDK from source as needed for these supported development OSs and game engines:

- **Operating systems** – Windows, Linux
- **Game engines** – Amazon Lumberyard, Unreal Engine, Unity, engines that support C++ or C# libraries

**Game server operating systems**

Use the Amazon GameLift Server SDK to create game servers that run on the following platforms:

- Windows Server 2012 R2
- Amazon Linux

**For Game Clients and Game Services**

Create 64-bit game clients and services using the AWS SDK with the Amazon GameLift API. This SDK enables client apps and services to find and manage game sessions and connect players to games being hosted on Amazon GameLift. Download the AWS SDK or view the Amazon GameLift API reference documentation.

**SDK support**

The AWS SDK with Amazon GameLift is available in the following languages. See documentation for each language for details on support for development environments and game client platforms.

- C++ ([SDK docs](https://docs.aws.amazon.com/gamelift/latest/DeveloperGuide/)) (Amazon GameLift)
- Java ([SDK docs](https://docs.aws.amazon.com/gamelift/latest/DeveloperGuide/)) (Amazon GameLift)
- .NET ([SDK docs](https://docs.aws.amazon.com/gamelift/latest/DeveloperGuide/)) (Amazon GameLift)
- Go ([SDK docs](https://docs.aws.amazon.com/gamelift/latest/DeveloperGuide/)) (Amazon GameLift)
- Python ([SDK docs](https://docs.aws.amazon.com/gamelift/latest/DeveloperGuide/)) (Amazon GameLift)
- Ruby ([SDK docs](https://docs.aws.amazon.com/gamelift/latest/DeveloperGuide/)) (Amazon GameLift)
- PHP ([SDK docs](https://docs.aws.amazon.com/gamelift/latest/DeveloperGuide/)) (Amazon GameLift)
- JavaScript/Node.js ([SDK docs](https://docs.aws.amazon.com/gamelift/latest/DeveloperGuide/)) (Amazon GameLift)

**SDK Compatibility**

If you use the Amazon GameLift SDKs bundled inside a version of Amazon Lumberyard, your game clients and servers will be compatible. If you upgrade the Amazon GameLift Server SDK independently, however, you need to use a compatible version of the AWS SDK to ensure that your game clients and services can successfully connect to your game servers on Amazon GameLift.

<table>
<thead>
<tr>
<th>If your game server uses this Server SDK version:</th>
<th>It can host game clients built with this AWS SDK for C++ version*:</th>
<th>Server SDK versions are available in:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Versions bundled into Amazon Lumberyard.</td>
<td>Versions bundled into Amazon Lumberyard.</td>
<td>Lumberyard 1.0 to 1.3 (beta)</td>
</tr>
<tr>
<td>version 3.0.7</td>
<td>version 0.12.16 (<a href="https://github.com/aws/aws-sdk-cpp/commit/">commit</a>) or later</td>
<td>Lumberyard 1.4 to 1.5 (beta)</td>
</tr>
<tr>
<td>version 3.1.0</td>
<td>version 0.14.9 (<a href="https://github.com/aws/aws-sdk-cpp/commit/">commit</a>) or later</td>
<td>• Lumberyard 1.6 to 1.7 (beta)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Downloaded from Amazon GameLift</td>
</tr>
</tbody>
</table>

Version 15
Free Tier and Billing Alerts

Amazon GameLift includes a free tier for one year with a c4.large instance. It is possible for the free tier to expire mid-month; therefore, you may want to set up and configure a billing alert to notify you of billing events, such as when you have reached the free tier threshold. For more information, see Creating a Billing Alarm.

In addition to receiving billing alerts, you can view your current estimated bill for Amazon GameLift on the Billing and Cost Management console at https://console.aws.amazon.com/billing/. This will allow you to review your resource consumption and determine if you would like to continue using these resources beyond the free tier allowance and incur charges, or if you would like to scale down your fleet and avoid incurring charges.

To avoid incurring charges in excess of the free tier, you may want to scale down your fleet (p. 62) when not in use.

* Version information for the AWS SDK for C++ can be found in this file: aws-sdk-cpp/aws-cpp-sdk-core/include/aws/core/VersionConfig.h.
Setting Up

The topics in this section describe the basic steps needed to begin using Amazon GameLift.

Topics

- Set Up Your Amazon Lumberyard Game Project (p. 17)
- Set up an AWS Account (p. 17)
- Install the AWS CLI (p. 18)

Set Up Your Amazon Lumberyard Game Project

Get your project ready for Amazon GameLift with the following steps. If you have installed Lumberyard, you may have already completed some of these tasks.

- Install Visual Studio 2013 runtime. Run the installer from the \3rdParty\Redistributables \Visual Studio 2013 directory or download and run the installer directly from Microsoft.
- Run Setup Assistant to validate that you have installed the appropriate third-party software and SDKs, including the Amazon GameLift client. LumberyardLauncherBatch.exe is provided in \dev\Bin64.
- Configure your Lumberyard game project to ensure it compiles properly. Follow these guidelines:
  - The server and client executables must link aws-cpp-sdk-core and aws-cpp-sdk-gamelift.
  - The server executable must be built on a platform supported by Amazon GameLift. See Amazon GameLift SDKs (p. 14) for a list of allowed platforms.
  - Your project must set the AWS_CUSTOM_MEMORY_MANAGEMENT pre-processor flag to 0 or 1, depending on your use of a custom memory manager.

Set up an AWS Account

Amazon GameLift is an AWS service, and you must have an AWS account to use Amazon GameLift. Creating an AWS account is free.

For more information on what you can do with an AWS account, see Getting Started with AWS.

Set up your account for Amazon GameLift

1. Get an account. Open Amazon Web Services and choose Sign In to the Console. Follow the prompts to either create a new account or sign into an existing one.
2. Set up user groups and access permissions. Open the AWS Identity and Access Management (IAM) service console and follow these steps to define a set of users or user groups and assign access permissions to them. Permissions are extended to a user or user group by attaching an IAM policy, which specifies the set of AWS services and actions a user should have access to. For detailed instructions on using the Console (or the AWS CLI or other tools) to set up your user groups, see Creating IAM Users.
   a. Create an administrative user or user group. Administrative users include anyone who manages core Amazon GameLift resources, such as builds and fleets. To set permissions, you must create your own policy from scratch. This example (p. 18) illustrates an administrator policy for Amazon GameLift services.
b. **Create a player user.** A player user represents your game client(s). It enables access to Amazon GameLift client functionality, such as acquiring game session information and joining players to games. Your game client must use the player user credentials when communicating with the Amazon GameLift service. To set permissions, you must create your own policy from scratch. This example (p. 18) illustrates a player policy for Amazon GameLift services.

### IAM Policy Examples for Amazon GameLift

You can use the following examples to create policies and add the appropriate permissions to your IAM users or user groups.

#### Simple Policy for Administrators

This policy provides full administrative access to a user. Attach it to a user or user group to permit all Amazon GameLift actions on all Amazon GameLift resources (fleets, aliases, game sessions, player sessions, etc.).

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

#### Simple Policy for Players

This policy enables access only to functionality needed by players who are using a game client to connect to an Amazon GameLift-hosted game server. This policy allows a user to get game session information, create new game sessions, and join a game session.

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

### Install the AWS CLI

You can use the AWS command line interface (AWS CLI) tool to make calls to any action in the AWS SDK, including the Amazon GameLift API. At a minimum, you'll need to use this tool to upload your game builds to the Amazon GameLift service.

**To install the AWS CLI for Amazon GameLift**

1. **Get the tool.** Download the latest version of the AWS CLI tool and follow the instructions to install it. This tool runs on Windows, Linux, OS X, and Unix.
2. **Verify installation.** Open a command line window or terminal and type `aws gamelift help`. If the CLI is correctly installed, you will see a "Welcome to the Amazon GameLift API Reference" message, followed by a list of Amazon GameLift commands.

3. **Configure the tool.** Type `aws configure` and enter the following values at the prompts:
   - **AWS access key ID** – Half of the AWS account user credentials, which are generated using the IAM service. For help, see Get your access key ID and secret access key.
   - **AWS secret access key** – Half of the AWS account user credentials, along with the AWS access key ID.
   - **Default region name** – Name of a region, such as "us-west-2" you want to set as default. If you don’t set a default region, every command must specify a region using the `--region` parameter. See a list of available regions for Amazon GameLift at AWS Regions and Endpoints.
   - **Default output format** – Format to receive API responses. Options include "json", "text" or "table". If you don't enter a default format, the CLI processes your requests but doesn't display any results.
Integrating the Amazon GameLift SDKs into Your Games

The Amazon GameLift SDKs provide the libraries needed to enable your game clients and servers to communicate with the Amazon GameLift service. The SDKs are available as part of the Amazon Lumberyard game engine download or can be downloaded separately. For more details on the SDKs and where to get them, see Amazon GameLift SDKs (p. 14).

The topics in this section describe how game clients and servers interact with the Amazon GameLift service, and provide instruction on adding Amazon GameLift functionality to your game clients and servers.

Explore Amazon GameLift

Before you prepare your own game to use Amazon GameLift, you can experiment with the service and console tools:

- Five-Click wizard – Use this wizard to use a sample game to create a game server build, deploy it on an Amazon GameLift fleet, and connect to the server from the sample game client. From here, you can view or edit the fleet using the Amazon GameLift console tools and view Amazon GameLift's metrics and other tools in action. To access the sample wizard, sign into the Amazon GameLift console and select Sample Game in the Amazon GameLift menu.
- Amazon Lumberyard Multiplayer Sample – If you're working with Lumberyard, you can use it to deploy a game server with GameLift or explore the source code to learn about how its game client and game server are integrated with GameLift. Learn more about the Multiplayer Sample game and how to configure it for GameLift.

Topics

- Integration Plan (p. 20)
- Game Engines and Amazon GameLift (p. 22)
- Integrating your Game Server for Amazon GameLift (p. 29)
- Integrating your Game Client for Amazon GameLift (p. 33)
- Amazon GameLift and Game Client/Server Interactions (p. 37)
- Testing Your Integration (p. 41)

Integration Plan

Are you ready to start using Amazon GameLift to run your multiplayer games? Follow the steps in this workflow to integrate Amazon GameLift into your games and deploy them. If you're new to Amazon GameLift, we recommend that you read What is Amazon GameLift? (p. 1) If you're unsure whether Amazon GameLift supports your operating systems and development environments, see For Game Servers (p. 14) and Game Engines and Amazon GameLift (p. 22).

Tip

You don't need to have a game ready to start experimenting with Amazon GameLift. Learn more about available tools to get you exploring with Amazon GameLift features fast in Explore Amazon GameLift (p. 20).
1. **Get set up to use Amazon GameLift.**
   - Create and configure your AWS account for Amazon GameLift. See [Set up an AWS Account](p. 17).
   - Install the AWS Command Line Interface (AWS CLI) tool. See [Install the AWS CLI](p. 18).

2. **Prepare your game server for hosting on Amazon GameLift.**
   - Download the [Amazon GameLift Server SDK](https://aws.amazon.com/gameli/). Build the version for your preferred programming language and game engine, and add it to your game server project. If you're using the Amazon Lumberyard game engine, a version of the SDK is built in. See [Game Engines and Amazon GameLift](p. 22).
   - Add code to your game server project to enable communication with the Amazon GameLift service. At a minimum, a game server should be able to do the following: (1) Notify Amazon GameLift when it is ready to host a game session, (2) Start game sessions when prompted by Amazon GameLift, (3) Notify Amazon GameLift when players connect and disconnect from a game session, and (4) Notify when a game session is ending or it is shutting down. See [Add Amazon GameLift to Your Game Server](p. 30).

3. **Prepare your game client to connect to Amazon GameLift-hosted game sessions.**
   - Set up your game client or game service projects with the AWS SDK. These include any projects that find or create game sessions and add players to games. The AWS SDK is available in several languages. See the [Amazon GameLift SDKs For Game Clients and Game Services](p. 15).
   - Configure your game client or service projects to connect to the Amazon GameLift. Determine the region to connect to, set a target fleet and/or queue, and store AWS credentials. See [Set Up Amazon GameLift on a Client or Service](p. 34).
   - Add functionality to retrieve information on game sessions, create new game sessions, and reserve space for players on a game session. See [Get Game Sessions](p. 34).

4. **Test your Amazon GameLift integration.**
   - Use Amazon GameLift Local to test your game client and game server integration using a version of the Amazon GameLift service running locally. You can use this tool to test your integration without having to upload game builds and set up fleets. You can verify that your game components are communicating with the Amazon GameLift service, and test core functionality. See [Testing Your Integration](p. 41).

5. **Upload your game server to the Amazon GameLift service.**
   - Package the game server binaries, dependencies, install instructions, and other files needed to run your game server. See [Package Build Files](p. 46).
   - Test the game build installation process on the target operating system to check for missing dependencies.
   - Upload the build to the Amazon GameLift service. Be sure to upload your build to each region where you plan to deploy your game. See [Upload Build Files to Amazon GameLift](p. 47).

6. **Build a fleet of computing resources to host your game.**
   - Design a fleet configuration for your game. You need to decide, for example, the type of computing resources to use, which regions to deploy to, how to allocate available resources, whether to use a fleet alias, and preferred settings for a variety of features. See [Choose Computing Resources](p. 52) and [Run Multiple Processes on a Fleet](p. 70).
   - Create a fleet based on your configuration design. You can create a fleet (and edit it) from either the Amazon GameLift console or the AWS CLI tool. When you create a fleet, it takes a few minutes before it is active, that is, ready to host game sessions. See [Create a Fleet](p. 54).
   - Resolve fleet creation issues that might indicate problems with your game server integration. See [Debug Fleet Creation Issues](p. 58).
   - Experiment with your Amazon GameLift fleet configuration settings and refine as needed. For example, change a fleet's capacity allocation to optimize your use of resources and set
up auto-scaling policies to manage expected player demand. See Remotely Access Fleet Instances (p. 72).

Once you’ve fully integrated Amazon GameLift into your game components, it’s a matter of managing your game server fleets for optimal availability and performance over the long term. Use Amazon GameLift tools to track things like how quickly and efficiently players can find and connect to a game session, overall performance of your game servers over time, and player usage patterns. See Viewing Your Game Data in the Console (p. 116).

Game Engines and Amazon GameLift

You can use Amazon GameLift with most major game engines that support C++ or C# libraries, including Amazon Lumberyard, Unreal Engine, and Unity. For more information on available SDKs, supported development platforms and

In addition to the engine-specific information provided in this topic, find additional help with integrating Amazon GameLift into your game servers, clients and services in the following topics:

- Integration Plan (p. 20) – A six-step workflow for successfully integrating Amazon GameLift into your game and setting up hosting resources.
- Add Amazon GameLift to Your Game Server (p. 30) – Detailed instructions on integrating Amazon GameLift into a game server.
- Add Amazon GameLift to Your Game Client (p. 33) – Detailed instructions on integrating into a game client or service, including creating game sessions and joining players to games.

Amazon Lumberyard

Amazon GameLift SDKs and functionality are fully incorporated into the Lumberyard product.

Game servers

Prepare your game servers for hosting on Amazon GameLift using the Amazon GameLift Server SDK for C++ (p. 137). See Add Amazon GameLift to Your Game Server (p. 30) to get help with integrating the required functionality into your game server.

Game clients and services

Enable your game clients and/or game services to interact with Amazon GameLift 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 into your Lumberyard game project, see Prepare Your Game Client in Amazon Lumberyard (p. 23) and Add Amazon GameLift to Your Game Client (p. 33).

Unreal Engine

Game servers

Prepare your game servers for hosting on Amazon GameLift by adding the Amazon GameLift Server SDK for Unreal Engine (p. 166) to your project and implementing the required server functionality. For help setting up the Unreal Engine plugin and adding Amazon GameLift code, see Add Amazon GameLift to an Unreal Engine Game Server Project (p. 24).
Game clients and services

Enable your game clients and/or game services to interact with Amazon GameLift 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 into your Unreal Engine game project, see Add Amazon GameLift to Your Game Client (p. 33).

Unity

Game servers

Prepare your game servers for hosting on Amazon GameLift by adding the Amazon GameLift Server SDK for C# (p. 152) to your project and implementing the required server functionality. For help setting up with Unity and adding Amazon GameLift code, see Add Amazon GameLift to a Unity Game Server Project (p. 27).

Game clients and services

Enable your game clients and/or game services to interact with Amazon GameLift 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 into your Unity game project, see Add Amazon GameLift to Your Game Client (p. 33).

Other Engines

For a full list of the Amazon GameLift SDKs available for game servers and clients, see the section called “GameLift SDKs” (p. 14).

Prepare Your Game Client in Amazon Lumberyard

All game clients must be configured to enable communication with the Amazon GameLift service, including specifics on which fleet to use, access credentials, how to connect, etc. The simplest method is to create a batch file that sets the console variables listed as follows.

Tip
You don’t need to have a game ready to start experimenting with Amazon GameLift. Learn more about available tools to get you exploring with Amazon GameLift features fast in Explore Amazon GameLift (p. 20).

To prepare the game client

1. In your batch file, set the following console variables to launch the game client. These variables have been added to \dev\Code\CryEngine\CryNetwork\Lobby\LobbyCvars

   • gamelift_aws_access_key = part of the IAM security credentials (p. 17) for a user with "player" access in your AWS account
   • gamelift_aws_secret_key = part of the IAM security credentials (p. 17) for a user with "player" access in your AWS account
   • gamelift_fleet_id = Unique ID of an active fleet to connect to
   • gamelift_alias_id = Unique ID of an alias pointing to a fleet to connect to
   • (Optional) gamelift_endpoint = Amazon GameLift server endpoint; the default value is gamelift.us-west-2.amazonaws.com
   • (Optional) gamelift_aws_region = AWS region name; default value is us-west-2
   • (Optional) gamelift_player_id = ID that you generate to uniquely identify a player (p. 37)
Add Amazon GameLift to an Unreal Engine Game Server Project

This topic helps you set up and use the Amazon GameLift Server SDK plugin for Unreal Engine in your game server projects. If you're unsure whether Amazon GameLift supports the operating systems you're using, see For Game Servers (p. 14).

Set Up the Unreal Engine Server SDK Plugin

Follow these steps to get the Amazon GameLift Server SDK plugin for Unreal Engine ready for your game server projects.

To set up the Amazon GameLift SDK plugin for Unreal Engine

1. **Download the Amazon GameLift Server SDK.** To verify that your game system requirements are supported, see Amazon GameLift SDKs (p. 14).

2. **Build the C++ Server SDK libraries for Unreal.** The SDK download contains the source code for C++ (see GameLift-<release date>/GameLift-SDK-Release-<version>/GameLift-cpp-ServerSDK-<version>).

To build the SDK libraries, go to the directory containing the C++ source code and compile it with the flag -DBUILD_FOR_UNREAL set to true. The following instructions show how to compile using cmake.

For Linux users:

```bash
mkdir out
cd out
cmake -DBUILD_FOR_UNREAL=1 ..
make
```

The following binary files are generated:

- out/prefix/lib/aws-cpp-sdk-gamelift-server.so

For Windows users:

```bash
mkdir out
cd out
```
cmake -G "Visual Studio 14 2015 Win64" -DBUILD_FOR_UNREAL=1 ..
msbuild ALL_BUILD.vcxproj /p:Configuration=Release

The following binary files are generated:

- out\prefix\bin\aws-cpp-sdk-gamelift-server.dll
- out\prefix\lib\aws-cpp-sdk-gamelift-server.lib

For more details on building the C++ SDK, including minimum requirements and build options, see the README.md file included in the download.

3. **Add the binaries to the Amazon GameLift plugin files.** Copy the new binary files into the ThirdParty directory of the Unreal plugin, with the following paths:

   In Linux use these paths:

   - ThirdParty/GameLiftServerSDK/Linux/x86_64-unknown-linux-gnu/aws-cpp-sdk-gamelift-server.so

   In Windows use these paths:

   - ThirdParty\GameLiftServerSDK\Win64\aws-cpp-sdk-gamelift-server.dll
   - ThirdParty\GameLiftServerSDK\Win64\aws-cpp-sdk-gamelift-server.lib

4. **Import the Amazon GameLift plugin into a project.** Although you have many ways to import a plugin into Unreal Engine, the following method does not require the Unreal Editor.

   a. Add the plugin to your game project. The plugin files must contain everything in the plugin's GameLiftServerSDK directory, including the generated binary files.

   b. Add the plugin to your game's .uproject file:

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

   c. Add the plugin as a dependency to your game's ModuleRules:

   ```java
   using UnrealBuildTool;
   public class MyAwesomeGame : ModuleRules
   {
   public MyAwesomeGame(TargetInfo Target)
   {
   PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject",
   "Engine", "InputCore", "GameLiftServerSDK" });
   }
   }
   ```

**Add Amazon GameLift Code**

For more information on adding Amazon GameLift functionality, see these topics:

- **Add Amazon GameLift to Your Game Server (p. 30)**
- **Amazon GameLift Server API Reference for Unreal Engine (p. 166)**
When adding Amazon GameLift-specific code to your Unreal Engine game project, enclose the code using the preprocessor flag `WITH_GAMELIFT=1`. This flag ensures that only server builds invoke the Amazon GameLift backplane API and allows you to write code that is executed correctly regardless of the build target type you might produce with it.

Code enclosed with the `WITH_GAMELIFT=1` flag is only processed if the following are true:

- The plugin found the Amazon GameLift server SDK binary files.
- The build is a game server: `Target.Type == TargetRules.TargetType.Server`

The following code snippet illustrates how to initialize an Unreal Engine game server with Amazon GameLift.

```cpp
AGameLiftFPSGameMode::AGameLiftFPSGameMode()
    : Super()
{
    // Let's run this code only if GAMELIFT is enabled. Only with Server targets!
    #if WITH_GAMELIFT
        // Getting the module first.
        FGameLiftServerSDKModule* gameLiftSdkModule =
            &FModuleManager::LoadModuleChecked<FGameLiftServerSDKModule>(FName("GameLiftServerSDK"));

        // InitSDK establishes a local connection with GameLift's agent to enable further communication.
        gameLiftSdkModule->InitSDK();

        // When a game session is created, GameLift sends an activation request to the game server and passes along the game session object containing game properties and other settings.
        // Here is where a game server should take action based on the game session object.
        // Once the game server is ready to receive incoming player connections, it should invoke GameLiftServerAPI.ActivateGameSession()
        auto onGameSession = [=](Aws::GameLift::Server::Model::GameSession gameSession)
        {
            gameLiftSdkModule->ActivateGameSession();
        };

        FProcessParameters* params = new FProcessParameters();
        params->OnStartGameSession.BindLambda(onGameSession);

        // OnProcessTerminate callback. GameLift invokes this callback before shutting down an instance hosting this game server.
        // It gives this game server a chance to save its state, communicate with services, etc., before being shut down.
        // In this case, we simply tell GameLift we are indeed going to shut down.
        params->OnTerminate.BindLambda([=](){gameLiftSdkModule->ProcessEnding();});

        // This is the HealthCheck callback.
        // GameLift invokes 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.
    
    
    
} // "GameLiftFPS" is a sample game name, replace with file names from your own game project
#include "GameLiftFPS.h"
#include "Engine.h"
#include "EngineGlobals.h"
#include "GameLiftFPSGameMode.h"
#include "GameLiftFPSHUD.h"
#include "GameLiftFPSCharacter.h"
#include "GameLiftServerSDK.h"
```

Version 26
Add Amazon GameLift to a Unity Game Server Project

This topic helps you set up the Amazon GameLift C# Server SDK and integrate Amazon GameLift into your Unity game server projects. If you're unsure whether Amazon GameLift supports the operating systems you're using, see For Game Servers (p. 14).

Set up the C# Server SDK for Unity

Follow these steps to build the Amazon GameLift Server SDK for C# and add it to your Unity game server projects.

To set up the Amazon GameLift Server SDK for Unity

1. **Download the Amazon GameLift Server SDK.** To verify that your game system requirements are supported, see Amazon GameLift SDKs (p. 14). The Server SDK includes the following two solutions:
   - GameLiftServerSDKNet35.sln for .Net framework 3.5
   - GameLiftServerSDKNet45.sln for .Net framework 4.5

   Only the .Net 3.5 solution is compatible with Unity.

2. **Build the C# SDK libraries.** See the ReadMe.md file for the C# Server SDK for minimum requirements and additional build options. In an IDE, load the .Net 3.5 solution file. To generate the SDK libraries, restore the NuGet packages and build the solution.

3. **Check the API compatibility setting.** In the Unity Editor, go to File, Build Settings, Player Settings. In the Other Settings section, under Optimization, make sure that the API compatibility level is set to .Net 2.0.

4. **Add the Amazon GameLift libraries to Unity.** In the Unity Editor, import the following libraries into the Assets/Plugins directory:
   - EngineIoClientDotNet.dll
   - GameLiftServerSDKNet35.dll
   - log4net.dll
   - Newtonsoft.Json.dll
Add Amazon GameLift Server Code

For more information on adding Amazon GameLift functionality, see these topics:

- Add Amazon GameLift to Your Game Server (p. 30)
- Amazon GameLift Server API (C#) Reference (p. 152)

The following code example uses a `MonoBehavior` to illustrate a simple game server initialization with Amazon GameLift.

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

public class GameLiftServerExampleBehavior : MonoBehaviour
{
    // This is an example of a simple integration with GameLift server SDK that makes game
    // server processes go active on Amazon GameLift
    public void Start()
    {
        var listeningPort = 7777;
        // InitSDK establishes a local connection with Amazon GameLift's agent to enable
        // further communication.
        var initSDKOutcome = GameLiftServerAPI.InitSDK();
        if (initSDKOutcome.Success)
        {
            ProcessParameters processParameters = new ProcessParameters(
                (gameSession) =>{
                    // When a game session is created, GameLift sends an activation request
                    // to the game server and passes along the game session object containing game properties and
                    // other settings.
                    // Here is where a game server should take action based on the game
                    // session object.
                    // Once the game server is ready to receive incoming player connections,
                    // it should invoke GameLiftServerAPI.ActivateGameSession()
                    GameLiftServerAPI.ActivateGameSession();
                },
                () =>{
                    // OnProcessTerminate callback. GameLift invokes this callback before
                    // shutting down an instance hosting this game server.
                    // It gives this game server a chance to save its state, communicate
                    // with services, etc., before being shut down.
                    // In this case, we simply tell GameLift we are indeed going to shut
                    // down.
                    GameLiftServerAPI.ProcessEnding();
                },
                () =>{
                    // This is the HealthCheck callback.
                    // GameLift invokes 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.
                    GameLiftServerAPI.ProcessEnding();
                });
        }
    }
}
```
//In this case, we're always healthy!
        return true;
    },
    listeningPort, //This game server tells GameLift that it will listen on
    port 7777 for incoming player connections.
    new LogParameters(new List<string>()
    {
        //Here, the game server tells GameLift what set of files to upload when
        the game session ends.
        //GameLift uploads everything specified here for the developers to
        fetch later.
        "/local/game/logs/myserver.log"
    }));

    //Calling ProcessReady tells GameLift this game server is ready to receive
    incoming 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.Destroy() when the application quits. This
        resets the local connection with GameLift's agent.
        GameLiftServerAPI.Destroy();
    }

## Integrating your Game Server for Amazon GameLift

The topics in this section describe how to integrate Amazon GameLift into a multiplayer game server.

Adding Amazon GameLift to your game server is Step 2 on the Integration Plan (p. 20) roadmap. The integration topics in this section assume that you've created an AWS account and have an existing game server project.

Topics
- Add Amazon GameLift to Your Game Server (p. 30)
- Amazon GameLift Server API (C++) Reference (p. 137)
- Amazon GameLift Server API (C#) Reference (p. 152)
- Amazon GameLift Server API Reference for Unreal Engine (p. 166)
Add Amazon GameLift to Your Game Server

Your game server needs to interact with the Amazon GameLift service once it is deployed and running as multiple server processes on an Amazon GameLift fleet. The code you add enables each server process to communicate with the Amazon GameLift service. Server processes must be able to respond to certain events triggered by the Amazon GameLift service and keep Amazon GameLift informed about player activity and server process status. See this complete description of Amazon GameLift interactions (p. 37).

Use the Amazon GameLift Server SDK for your preferred language to add Amazon GameLift functionality to your game server. See the following Server API references for more complete information:

- C++ Server API Reference (p. 137)
- C# Server API Reference (p. 152)
- Unreal Engine Plugin API Reference (p. 166)

To integrate Amazon GameLift into your game server, add the Amazon GameLift Server SDK to your game server project and build the functionality described in this topic.

Prepare a Server Process

Add code to initialize an Amazon GameLift client and notify the Amazon GameLift service that the server is ready to host a game session. This code should run automatically before any Amazon GameLift-dependent code, such as on launch.

**Note**

Server API action names are the same in all available languages.

1. Initialize the Server SDK. Call `InitSdk()`.
2. Notify Amazon GameLift that a game server process is ready to host a game session. Each server process started on an Amazon GameLift instance must call `ProcessReady()` with the following information:

   - Port number being used by the server process. This port number, along with an IP address, is stored in a game session object, which game clients use when connecting to an active game session.
   - Location of files generated during an active game session that you want Amazon GameLift to retain, such as game session logs. Files generated by a server process when hosting a game session are saved to the Amazon GameLift instance, but lost when an instance is shut down. Amazon GameLift will upload any of these files you request; once uploaded, they are accessible through the Amazon GameLift console or by calling the Amazon GameLift API `GetGameSessionLogUrl()`. Consider using a file naming scheme that uniquely identifies game sessions if your fleet is configured to run multiple concurrent server processes per instance.
   - Names of callback functions used by Amazon GameLift to trigger certain actions on the server process. More information on implementing these functions is provided in the following sections and in `ProcessParameters (p. 150)`.
     - `onHealthCheck (required)` is called regularly to request a health status report from the server process.
     - `onStartGameSession (required)` is called when the Amazon GameLift service receives request to start a new game session (`CreateGameSession()`).
     - `onProcessTerminate (required)` is called when the Amazon GameLift service needs to force the server process to terminate, allowing the server process to shut down gracefully.
     - `onUpdateGameSession (optional)` is called when the Amazon GameLift service delivers an updated game session object to the game server or provides a status update on a match backfill.
request. This callback is required only if you're using the FlexMatch backfill (p. 12) feature. Learn more about how to implement this callback function at Update Match Data on the Game Server (p. 115).

Depending on how you configure your fleets, there may be multiple server processes running concurrently on a fleet instance. Once one server process on an instance calls ProcessReady() successfully, the Amazon GameLift service sets the instance's status to ACTIVE.

## Report Server Process Health

Add code to implement the callback function onHealthCheck(). This function is invoked by the Amazon GameLift service regularly to collect health metrics from the server process. The server process's response to a health check is a binary: healthy or unhealthy. When implementing this callback function, do the following:

- Evaluate the status of the server process using whatever measures make sense for your game. For example, you might report the server process as unhealthy if any external dependencies have failed or if metrics such as memory capacity fall outside a defined limit.
- Complete the health evaluation and respond to the callback within 60 seconds. If the Amazon GameLift service does not receive a response in that time, it will automatically consider the server process to be unhealthy.
- Return a boolean value: true for healthy, false for unhealthy.

If you do not implement a health check callback, the Amazon GameLift service considers the server process to be healthy unless the process is not responding, in which case it is considered unhealthy.

Server process health is used by Amazon GameLift to efficiently end unhealthy processes and free up resources. If a server process continues to report unhealthy or does not respond for three consecutive health checks, the Amazon GameLift service may shut down the process and start a new one. Metrics on a fleet's server process health is collected and viewable on the Amazon GameLift console.

## Start a Game Session

Add code to implement the callback function onStartGameSession. This function is invoked by the Amazon GameLift service when creating a new game session, which may occur in response to a CreateGameSession() request or as part of game session placement or matchmaking activities.

The onStartGameSession function takes a GameSession object, provided by the Amazon GameLift service, as an input parameter. This object contains the game session ID and other information that defines the requested game session. The function should accomplish the following tasks:

- Perform whatever actions are needed to create a new game session. The new game session should reflect the GameSession object, including creating slots for the specified maximum number of players and referencing the game session name and ID. The Amazon GameLift service provides the same game session information to the game client.
- Process the game property values specified by the game client in its request. The game properties are contained in the GameSession object.
- At some point after the new game session is ready to accept players, the server process must call the server API action ActivateGameSession(). In response to a successful call, the Amazon GameLift service changes the game session status to ACTIVE.
Validate a New Player

Add code to verify a player connection request with the Amazon GameLift service. This code should run whenever a new player attempts to connect to the server process and before accepting the connection.

Connection requests from a game client must reference a player session ID. This ID is issued by the Amazon GameLift service and used to reserve a player slot in the game session (in response to an AWS SDK call to `CreatePlayerSession()`). The game server must call `AcceptPlayerSession()` with the referenced player session ID to verify that the connection request is coming from the authorized player.

Once the player session ID is validated by the Amazon GameLift service, the server process can accept the connection and allow the player to join the game session. If the player session ID is not validated by the Amazon GameLift service, the server process should deny the connection.

To acquire player data associated with the game session, call `DescribePlayerSessions()`.

Report a Player Session Ending

Add code to notify the Amazon GameLift service when a player disconnects from the game session. This code should run whenever the server process detects a dropped connection.

In the code handling the dropped connection, add a call to the server API action `RemovePlayerSession()` with the player session ID associated with the dropped connection. This notification enables the Amazon GameLift service to accurately track the number of current players and available slots in the game session.

Stop a Game Session

Add code to notify the Amazon GameLift service when a game session is ending. The notification enables the Amazon GameLift service to accurately track a server process's availability for new game sessions. This code should be added to the normal game session ending process.

At the end of the code to stop a game session, add a call to the server API action `TerminateGameSession()`. On successful receipt of this notification, the Amazon GameLift service changes the game session status to TERMINATED and may immediately start a new game session.

Note

If stopping a game session will be immediately followed by shutting down the server process, you can call the server API action `ProcessEnding()`, which terminates both the game session and the server process.

Shutdown a Server Process

Shut down may be initiated by a server process or by the Amazon GameLift service. Make the following changes to your game server code to handle either scenario.

- Implement the callback function `onProcessTerminate()`. This function should call your code that shuts down the game server. The Amazon GameLift service calls this function to initiate a server process shut down, usually prior to terminating the instance that is running the server process. Reasons to invoke this call is when Amazon GameLift is shutting down an unhealthy server process, when a spot instance is interrupted, or when instance capacity is scaling down. After receiving this call, the server process usually has a few minutes (two minutes in the case of a spot instance termination) to gracefully disconnect players, preserve game state data, and perform other cleanup tasks.

- Call the server API action `GetTerminationTime()` (p. 140) from your game server shut-down code. If Amazon GameLift has issued a call to terminate the server process, `GetTerminationTime()` returns the estimated termination time, if available. You can use this information to determine which shut-
down activities can be done in the time remaining, such as saving game state data or migrating players to a new game session.

- At the start of your game server shut-down code, call the server API action `ProcessEnding()` (p. 141). This call notifies the Amazon GameLift service that the server process is shutting down. On receipt of this notification, the Amazon GameLift service changes the server process status to TERMINATED and may immediately recycle the instance's resources as needed. This also ensures that no new game sessions are sent to the process. Once `ProcessEnding()` has been invoked, it is safe for the process to shut down.

 Integrating your Game Client for Amazon GameLift

The topics in this section describe the Amazon GameLift functionality you can add to a game client or game service that handles the following tasks:

- Requests information about active game sessions from the Amazon GameLift service.
- Joins a player to an existing game session.
- Creates a new game session.
- Changes metadata about an existing game session.

Adding Amazon GameLift to your multiplayer game client is Step 5 in the Integration Plan (p. 20). The following instructions assume that you've created an AWS account, generated an Amazon GameLift-enabled game server and uploaded it to Amazon GameLift, and used Amazon GameLift tools (such as the Amazon GameLift console) to create and configure a virtual fleet to host your game sessions. When adding Amazon GameLift to your game client, you must be able to provide AWS account credentials and specify a fleet to be used with the client.

For more information on how game clients interact with the Amazon GameLift service and game servers running on Amazon GameLift, see Amazon GameLift and Game Client/Server Interactions (p. 37).

Topics

- Add Amazon GameLift to Your Game Client (p. 33)
- Create Game Sessions with Queues (p. 36)
- Generate Player IDs (p. 37)

Add Amazon GameLift to Your Game Client

You can integrate Amazon GameLift into game components that need to acquire game session information, create new game sessions, and/or join players to games. Depending on your game architecture, this functionality might be placed in the game client or in game services that handle tasks such as player authentication, matchmaking, or game placement.

To do this, use the AWS SDK with Amazon GameLift API. This SDK is available in C++, C#, and several other languages. For details on the AWS SDK, version information, and language support, see Amazon GameLift SDKs (p. 14). Most of the links in this topic go to the Amazon GameLift Service API Reference, which describes the low-level service API for Amazon GameLift-related actions and includes links to language-specific reference guides.

Note

Interested in using matchmaking to get players into games? Once you've set up Amazon GameLift on your game client or a service, add FlexMatch to match players and create custom game sessions. See more at Matchmaking with FlexMatch (p. 89).
Set Up Amazon GameLift on a Client or Service

You add code to initialize an Amazon GameLift client or game service and store some key settings for use with Amazon GameLift. This code needs to be located so that it runs before any Amazon GameLift-dependent code, such as on launch.

**Note**
To set up your game client for testing using Amazon GameLift Local, see Testing Your Integration (p. 41)

1. Decide whether to use the default client configuration or create custom settings. For custom settings, you must create a custom `ClientConfiguration` object. See AWS ClientConfiguration (C++) for object structure and the default settings.

   A client configuration specifies a target region and endpoint. The region determines which resources (fleets, queues, matchmakers, etc.) Amazon GameLift interacts with when responding to requests. The default client configuration specifies the US East (N. Virginia) region. To use any other region, create a custom configuration. See this list of AWS regions supported by Amazon GameLift for names and endpoints. If your client or service needs to make requests for multiple regions, create a separate client configuration object for each target region and each as needed. See Using Regions with the AWS SDKs for language-specific examples.

2. Initialize an Amazon GameLift client. Call `Aws::GameLift::GameLiftClient()` (C++) using either a client configuration with the default settings or a custom configuration.

3. Add a mechanism to generate a unique identifier for each player. Amazon GameLift requires a unique player ID to connect to a game session. For more details, see Generate Player IDs (p. 37).

4. Collect and store the following information to use when contacting Amazon GameLift:
   - **Target fleet** – Most Amazon GameLift API requests must specify a fleet, such as when getting information on available game sessions or managing game sessions and player sessions. How you define the optimal target fleet (for example, setting a static fleet, or choosing a fleets based on a device’s physical location). To specify a target fleet, use either a fleet ID or an alias ID that points to the target fleet. Fleet aliases are highly useful in that you can switch players from one fleet to another without issuing a game client update. The combination of target fleet and region (specified in the client configuration) uniquely identifies the fleet.
   - **Target queue** – If your game uses multi-fleet queues to place new game sessions, you can specify which queue to use. To specify a target queue, use the queue name. The queue must be configured in the region.
   - **AWS credentials** – All calls to the Amazon GameLift service must provide credentials for the AWS account that hosts the game. This is the account you used to set up your Amazon GameLift fleets, and you should have created an IAM user or user group for players with a permissions policy. You need to create an `Aws::Auth::AWSCredentials` (C++) object containing an IAM access key and secret key for the player user group. For help finding the keys, see Managing Access Keys for IAM Users.

Get Game Sessions

Add code to discover available game sessions and manage game sessions settings and metadata. See Game and Player Session Features (p. 8) for more on game session features.

**Search for active game sessions.**

Use `SearchGameSessions` to get information on 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.

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 not accepting new players: `PlayerSessionCreationPolicy = DENY_ALL`
• Exclude game sessions that are full: `CurrentPlayerSessionCount = MaximumPlayerSessionCount`
• Choose game sessions based on length of time the session has been running: Evaluate `CreationTime`
• Find game sessions based on a custom game property: `gameSessionProperties.gameMode = "brawl"

Manage game sessions.

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 whether you are deploying your game in multiple regions or in a single region.

Create a game session using a multi-region queue.

Use `StartGameSessionPlacement` to place a request for a new game session in a queue. To use this feature, you'll need to set up a queue, which determines where (on which fleets) the new game session can be placed. A queue processes a game session placement request by trying each possible fleet, in turn, until it finds one with available resources to host the new game session. For more detailed information on queues and how to use them, see Design a Game Session Queue (p. 79).

When creating a game session placement, specify the name of the queue to use, a game session name, a maximum number of concurrent players for the game, and an optional set of game properties. You can optionally provide a list of players to automatically join to the game session. If you include player latency data for relevant regions, Amazon GameLift uses this information to place the new game session on a fleet that provides the best possible gameplay experience for players.

Game session placement is an asynchronous process. Once 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` to retrieve an updated GameSessionPlacement object. When a new game session is created, the GameSessionPlacement reflects the following changes: (1) Status changes from Pending to Fulfilled; (2) New game session information is added, including game session ID and region; and (3) New player session information is added (if requested).

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. Your game should handle failures as best suits your game and players. For example, you might repeat the request until resources are freed or scaled up, or you might switch to a different fleet. Once Amazon GameLift has created the new game session and returned a `GameSession` object, you can start joining players to it.

When you use this method to create a game session, specify a fleet ID or alias ID, a session name, and a 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 in which you define the keys and a
set of values that are meaningful to your game. This information is passed to the server process hosting the new game session, to be used as designed in your game server. For example, you might use game properties to direct the game session to use a certain game map or a particular set of rules.

If you use Amazon GameLift's resource protection feature to limit the number of game sessions one player can create, you'll need to indicate the creator's player ID.

**Join a Player to a Game Session**

Add code to reserve player slots in active game sessions and connect game clients to game sessions.

1. **Reserve a player slot in a game session.**

   To reserve a player slot, create a new player session for the game session. See How Players Connect to Games (p. 8) for more on player sessions. You have two ways to create new player sessions:

   - If you're using `StartGameSessionPlacement` to create game sessions, as described in the previous section, you can reserve slots for one or more players in the new game session.
   - Reserve player slots for one or more players using `CreatePlayerSession` or `CreatePlayerSessions` with a game session ID.

   With both methods, Amazon GameLift first verifies that the game session is accepting new players and has an available player slot. If successful, Amazon GameLift reserves a slot for the player, creates the new player session, and returns a `PlayerSession` object containing the IP address and port that a game client needs to connect to the game session.

   A player session request must include a unique ID for each player. See Generate Player IDs (p. 37) for more on player IDs.

   Optionally, a player session request can include a set of custom player data. This data is stored in the newly created player session object, which can be retrieved by calling `DescribePlayerSessions()`. It is also passed from the Amazon GameLift service to the game server when the player connects directly to the game session. Player data is not used by Amazon GameLift; it is a simple string of characters that is available to your game components for interpretation. When requesting multiple player sessions, you can provide a string of player data for each player, mapped to the player ID in the request.

2. **Connect to a game session.**

   Add code to the game client to retrieve the `PlayerSession` object, which contains the game session's IP address and port. Use this information to connect directly to the server process. In order to claim the reserved player slot, the connection request must reference the player session ID. Once connected, the game client and server process communicate directly, without involving Amazon GameLift. This framework minimizes the amount of latency in gameplay.

   The server process maintains communication with the Amazon GameLift service about the player connection in order to track availability of player slots. On initial connection, it contacts the service to verify that the player session ID is valid and in a reserved status. If validated, the reservation is claimed and the player connection is accepted. Later, when the player disconnects, the server process reports the dropped connection.

**Create Game Sessions with Queues**

This set of features lets you place new game sessions more efficiently across Amazon GameLift resources, and better supports matchmaking services. Previously, new game session requests were limited to a single fleet (`CreateGameSession`), and the request failed if the fleet was at full capacity or otherwise compromised.
Use a queue to place new game sessions on any one of a group of fleets that can span regions. Increased player demand can be shifted to lesser used fleets in other regions as needed. Queues also decrease the overhead needed to monitor fleets and balance resources across multiple fleets and regions. You can manage queues and track queue performance metrics in the Amazon GameLift Console.

Create a new game session placement request and add it to a queue. A game session placement request includes standard game session properties, and also lets you add one or more players to the new game session.

When creating a game session placement request, include player latency data to help Amazon GameLift choose a fleet in a region that provides the best possible experience for all the players.

**Generate Player IDs**

Amazon GameLift uses a player session to represent a player connected to a game session. A player session must be created each time a player connects to a game session. When a player leaves a game, the player session ends and is not reused.

Amazon GameLift provides a file called *Lobby.cpp* in the Amazon Lumberyard sample project *MultiplayerSample* that demonstrates how to generate a new, random ID number for every player in every new game session. You are not required to use the sample code; we provide it as an example. You can also rewrite the code to persist your own unique, non-personally identifiable player IDs.

The following sample code in *Lobby.cpp* randomly generates unique player IDs:

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

You can view player sessions by Player ID in the AWS Management Console for Amazon GameLift. For more information on player sessions, see View Data on Game and Player Sessions (p. 122).

**Amazon GameLift and Game Client/Server Interactions**

This topic describes the interactions between a client app, a game server, and the Amazon GameLift service. See also the Amazon GameLift–Game Server/Client Interactions (p. 40) diagram.

**Setting Up a New Server Process**

1. The **Amazon GameLift service** launches a new server process on an Amazon Elastic Compute Cloud (Amazon EC2) instance.
2. The **server process**, as part of the launch process, calls these server API actions:
   - `InitSDK()` to initialize the server SDK.
   - `ProcessReady()` to communicate readiness to accept a game session and specify connection port and location of game session log files.

   It then waits for a callback from the Amazon GameLift service.
3. The **Amazon GameLift service** changes the status of the EC2 instance to ACTIVE, with 0 game sessions and 0 players.
4. The **Amazon GameLift service** begins calling the `onHealthCheck` callback regularly while the server process is active. The server process must report either healthy or not healthy within one minute.
Creating a Game Session

1. The **Client app** calls the client API action `CreateGameSession()`.
2. The **Amazon GameLift service** searches for an active server with 0 game sessions. When found, it does the following:
   - Creates a new `GameSession` object, using the port setting reported by the server process in `ProcessReady()`, and sets its status to `ACTIVATING`.
   - Responds to the client app request with the `GameSession` object.
   - Invokes the `onStartGameSession` callback on the server process, passing the `GameSession` object.
3. The **server process** runs the `onStartGameSession` callback function. When ready to accept player connections, the server process calls the server API action `ActivateGameSession()` and waits for player connections.
4. The **Amazon GameLift service** changes the `GameSession` status to `ACTIVE`.

Adding a Player to a Game Session

1. The **Client app** calls the client API action `CreatePlayerSession()` with a game session ID.
2. The **Amazon GameLift 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, it does the following:
   - Creates a new `PlayerSession` object and sets its status to `RESERVED`.
   - Responds to the client app request with the `PlayerSession` object.
3. The **Client app** connects directly to the server process with the player session ID.
4. The **server process** calls the Server API action `AcceptPlayerSession()` to validate the player session ID. If validated, the Amazon GameLift service passes the `PlayerSession` object to the server process. The server process either accepts or rejects the connection.
5. The **Amazon GameLift service** does one of the following:
   - If the connection is accepted, sets the `PlayerSession` status to `ACTIVE`.
   - If no response is received within 60 seconds of the client app's original `CreatePlayerSession()` call, changes the `PlayerSession` status to `TIMEDOUT` and reopens the player slot in the game session.

Removing a Player From a Game Session

1. The **Client app** disconnects from the server process.
2. The **server process** detects the lost connection and calls the server API action `RemovePlayerSession()`.
3. The **Amazon GameLift service** changes the `PlayerSession` status to `COMPLETED` and reopens the player slot in the game session.

Shutting down a Game Session

**Shutting down a game session**

1. The **server process** calls the server API action `TerminateGameSession()`.
2. The **Amazon GameLift service** does the following:
   - Changes the `GameSession` status to `TERMINATED`. 
Terminating a Server Process

1. The server process does the following:
   - Runs code that gracefully shuts down the server process.
   - Calls the server API action `ProcessEnding()` to inform the Amazon GameLift service.

2. The Amazon GameLift service does the following:
   - Uploads game session logs (if any) to Amazon S3.
   - Changes the server process status to TERMINATED.
   - Recycles instance resources based on the fleet's runtime configuration.

Responding to a Shutdown Request

1. The Amazon GameLift service invokes the server process's `onProcessTerminate` callback. This call is used to shut down a server process that has reported unhealthy or not responded with health status for three consecutive minutes.

2. The server process runs the `onProcessTerminate` callback function, which triggers the server's termination process, ending with a call to `ProcessEnding()`.

3. The Amazon GameLift service does the following, either in response to receiving the `ProcessEnding()` call or after five minutes:
   - Uploads game session logs (if any) to Amazon S3.
   - Changes the server process status to TERMINATED.
   - Recycles instance resources based on the fleet's runtime configuration.
Amazon GameLift–Game Server/Client Interactions

### Launch server process

- **Client App**
  - Request new game session

- **AWS SDK**
  - Initialize()
  - SetTargetFleet() or SetTargetAliasId()

- **Game Server**
  - On launch:
    - Initialize GameLift SDK
    - Signal ready for game sessions
  - Evaluate server health
  - Health status (bool)

### Start game

- **Client App**
  - On launch:
    - Initialize GameLift SDK
    - Store settings

- **Game Server**
  - Set up game session
  - Signal ready for players

### Add player

- **Client App**
  - Request new player session

- **Game Server**
  - AcceptPlayerSession()

### Drop player

- **Client App**
  - Disconnect from server

- **Game Server**
  - Notify GameLift when lost connection is detected

---

*Version 40*
Testing Your Integration

Use Amazon GameLift Local to run a limited version of the Amazon GameLift 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 and configuring a fleet to host your game—can take 30 minutes or more each time.

With Amazon GameLift Local, you can verify the following:

- Your game server is correctly integrated with the Server SDK and is properly communicating with the Amazon GameLift 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 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 Local is a command-line tool that starts a self-contained version of the Amazon GameLift service. Amazon GameLift Local also provides a running event log of server process initialization, health checks, and API calls and responses. Amazon GameLift Local recognizes a subset of the AWS SDK actions for Amazon GameLift. 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 web service.

Set Up Amazon GameLift Local

Amazon GameLift 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-supported language.

Before running Local, you must also have the following installed.

- A build of the Amazon GameLift Server SDK version 3.1.5 or higher
- 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 Local service. This verifies that your game server is performing as expected with the following:

- The game server launches properly and initializes the Amazon GameLift Server SDK.
- As part of the launch process, the game server notifies Amazon GameLift that the server is ready to host game sessions.
- The game server sends health status to Amazon GameLift every minute while running.
- The game server responds to requests to start a new game session.

1. Start Amazon GameLift 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-3.1.6.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-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 Local service. This means that your game server successfully initialized the Amazon GameLift Server SDK (with \texttt{InitSDK()}). It has called \texttt{ProcessReady()} with the log paths shown and, if successful, is ready to host a game session. While the game server is running, Amazon GameLift 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!
16:50:53,541 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: \texttt{<process ID>}! Was InitSDK() invoked?”
- Warning: “Process state already exists for process with pID: \texttt{<process ID>}! Is ProcessReady(...) invoked more than once?”

3. **Start the AWS CLI.**

Once your game server successfully calls \texttt{ProcessReady()}, you can start making client calls. Open another command prompt window and start the AWS CLI tool. If you haven't already set it up, see \textit{Install the AWS CLI (p. 18)}. The AWS CLI by default uses the Amazon GameLift web service endpoint. You must override this with the Local endpoint in every request using the \texttt{--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, calls to AWS SDK APIs result in responses as documented in the .

4. **Create a game session.**

With the AWS CLI, submit a \texttt{CreateGameSession()} request. The request should follow the expected syntax. For Local, the \texttt{FleetId} parameter can be set to any valid string (\texttt{^fleet-\S+}).
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 Local. You can verify the following actions:

- The game client is successfully making AWS SDK requests to the Amazon GameLift 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 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.

   ```bash
   ./gamelist-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-integrated game server locally. See [Test a Game Server](p. 41) 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 Local service, you must make the following changes to your game client's setup, as described in [Set Up Amazon GameLift on a Client or Service](p. 34):

   - 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 service.

4. **Test game client calls to the Amazon GameLift 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 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 service.**

For Amazon GameLift to manage player demand and correctly report metrics, your game server must report various statuses back to Amazon GameLift. 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 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 Local log messages should show that your game server calls TerminateGameSession(). An AWS CLI call to DescribeGameSessions() should reflect a status change from Active to Terminated (or Terminating).

- **Server process is terminated** – Amazon GameLift Local log messages should show that your game server calls ProcessEnding().

**Variations with Local**

When using Amazon GameLift Local, keep in mind the following:

- Unlike the Amazon GameLift 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
```
Uploading Your Game to Amazon GameLift

Before setting up computing resources to host your Amazon GameLift-enabled multiplayer game, you first need to create a game build and upload it to the Amazon GameLift service. A game build includes all the server executables and dependent files needed to run server processes and host game sessions. Once you’ve uploaded a build to Amazon GameLift, you can then create a fleet of computing resources to operate your game.

The topics in this section describe how to package your game build and how to use the AWS CLI tool to upload it to Amazon GameLift.

Tip
You don’t need to have a game ready to start experimenting with Amazon GameLift. Learn more about available tools to get you exploring with Amazon GameLift features fast in Explore Amazon GameLift (p. 20).

Topics
- Package Build Files (p. 46)
- Upload Build Files to Amazon GameLift (p. 47)

Package Build Files

Before uploading your Amazon GameLift-enabled game server to the Amazon GameLift service for hosting, you need to package all game server files into a build directory. This directory must include all components required to run your game servers, including the following:

- **Game server binaries** – The binary files required to run the game server. A build can include binaries for multiple game servers, as long as they are built to run on the same platform (see supported platforms (p. 14)).
- **Dependencies** – Any dependent files required by your game server executables to run. Examples include assets, configuration files, and dependent libraries.
- **Install script** – Script file to handle tasks that are required to fully install your game build onto Amazon GameLift hosting servers. This file must be placed at the root of the build directory. An install script is run once as part of fleet creation.

Once you’ve packaged your build files, make sure your game server can run on a clean installation of your target operating system (not one that’s been used for development). This step helps ensure that you include all required dependencies in your package and that your install script is accurate.

To observe the build packaging process in action, see the Amazon GameDev tutorials for Amazon GameLift.

**Note**
If you’re storing your game build files in an Amazon S3 bucket for uploading, place all build files into a .zip file.
Creating an Install Script

Create an install script for the operating system of your game build:

- Windows: create a batch file named "install.bat".
- Linux: create a shell script file named "install.sh".

When creating an install script, keep in mind the following:

- The script cannot take any user input.
- A build is installed on a hosting server in the following locations. File directories in your build package are recreated.
  - For Windows fleets: C:\game
  - For Linux fleets: /local/game
- During the installation process, the run-as user has limited access. It has full rights to the directory where your build files are installed. If your install script takes actions that require administrator permissions, you'll need to specify admin access (sudo for Linux, runas for Windows). Permissions failures related to the install script generate an event message indicating a problem with the script.
- On Linux, common shell interpreter languages such as bash are supported. Add a shebang (such as 
  #!/bin/bash) to the top of your install script. If you need to verify support for your preferred shell commands, you can remotely access an active Linux instance and open a shell prompt. Learn more at Remotely Access Fleet Instances (p. 72).

Example scripts

These examples illustrate common script usage for Windows and Linux.

Windows

This example install.bat file installs Visual C++ runtime components required for the game server and writes the results to a log file. The component file is included in the build package at the root.

```bat
vcredist_x64.exe /install /quiet /norestart /log c:\game\vcredist_2013_x64.log
```

Linux

This example install.sh file illustrates using bash in your install script and writing results to a log file.

```sh
#!/bin/bash
echo 'Hello World' > install.log
```

Upload Build Files to Amazon GameLift

Once your game server files are packaged (p. 46), you must deliver them to Amazon GameLift for hosting. This is done by creating an Amazon GameLift build. You have two options for creating a build:

- Create a build with game build files that are stored in a directory. This is the simplest method.
- Create a build with game build files that are stored in an Amazon S3 account.

When you create a build, a new game build record is created. This record contains a unique build ID (example: build-75bf99cd-2dd8-2039-8074-ab24da1f80e4), creation time stamp, uploaded file size, status, and other metadata. The build is immediately placed in **Initialized** status and remains there.
until Amazon GameLift acquires the build files. Once the game server files are successfully acquired, the build moves to **Ready** status.

Once a build is in **Ready** status, you can deploy it with a new Amazon GameLift fleet. When you create a fleet for the build, Amazon GameLift sets up new fleet instances and installs the game server build on each instance. Build files are installed in the following locations:

- For Windows fleets: C:\game
- For Linux fleets: /local/game

To troubleshoot fleet activation problems that might be related to build installation, you can remotely access a fleet instance for debugging. See **Remotely Access Fleet Instances (p. 72)**.

---

### Create a Build with Files in a Directory

To create a game build with packaged game server files stored in any location, including a local directory, use the AWS Command Line Interface (AWS CLI) command `upload-build`. This command creates a new build record in Amazon GameLift and uploads files from a location you specify.

1. **Send an upload request.** In a command line window, type the following command and parameters.

   ```bash
   aws gamelift upload-build --operating-system [supported OS] --build-root [build path] 
   --name [user-defined name of build] --build-version [user-defined build number] --region [region name]
   --build-root
   The directory path of your build files.
   --operating-system
   The operating system that all server executables in the build run on. All fleets created with this build automatically use this OS. Valid values are WINDOWS_2012 and AMAZON_LINUX. This parameter is optional. If an operating system is not specified, Amazon GameLift uses the default value (WINDOWS_2012). This value cannot be changed later.
   --region
   Name of the region that you want to create your build in. You must create the build in the region that you want to deploy fleets in. To deploy in multiple regions, you must create a build and upload files for each region. If you configured your AWS CLI with a default region, you can omit this parameter.
   **Note**
   If you work in multiple regions, it is always a good idea to check your current default region. Use the AWS CLI command `configure get` (`aws configure get region`) to see your current default region. Use the command `configure set` (`aws configure set region [region name]`) to set a default region.
   --name and --build-version
   Use these parameters to describe your build. These metadata values can be changed later using `update-build` (or the AWS SDK operation `UpdateBuild`).

   In response to your upload request, the Amazon GameLift service provides upload progress, and on a successful upload returns the new build record ID. Upload time depends on the size of your game files and the connection speed.

   **Examples:**

   ```bash
   aws gamelift upload-build --operating-system WINDOWS_2012 --build-root /path/to/build 
   --name MyBuild --build-version 1.0 --region us-east-1
   ```
Create a Build with Files in Amazon S3

To create a game build with packaged game server files stored in an Amazon S3 bucket in your AWS account, use the AWS CLI command create-build. This operation creates a new build in Amazon GameLift and acquires your build files from the Amazon S3 bucket that you specify.

1. **Store your build files in Amazon S3.** Create a .zip file containing the packaged build files and upload it to an Amazon 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 build.

2. **Give Amazon GameLift access to your build files.** Using the AWS Identity and Access Management (IAM) service, set up a new role that allows Amazon GameLift to access to your build files. This step requires the following tasks: (1) create the role and give it a name, (2) attach a trust relationship policy that allows Amazon GameLift to assume the role, and (3) attach an access policy that limits the role's access. You can limit access as tightly as you need to, even to one specific file in a bucket. Once you've created the role, take note of the new role's Amazon Resource Name (ARN), which you'll need when creating a build.

   **Note**
   You can use the IAM console or AWS CLI tool to set up the role. If you use the console Create Role wizard, the easiest method is to create a simple AWS service role (select any service and do not attach a policy). Then add the trust relationship and access policy (shown as follows) to the newly created role. See the IAM user guide topic Creating a Role for help using the IAM console or AWS CLI tool.

   Attach the following policies to the role:

   - Trust relationship limited to Amazon GameLift:

     ```json
     {
         "Version": "2012-10-17",
         ...  // Trust policy for Amazon GameLift
     }
     ```

2. **Check build status.** View the new build record, including current status, using describe-build (or DescribeBuild). You can also view status on the Amazon GameLift console.

   In a command line window, type the following command and parameters.

   ```bash
   aws gamelift describe-build --build-id [build ID returned with the upload request] --region [region name]
   ```

   **Example:**

   ```bash
   aws gamelift describe-build --build-id "build-75bf99cd-2dd8-2039-8074-ab24da1f80e4" --region us-west-2
   ```

   In response to your request, the Amazon GameLift service returns the requested build record. This record contains a set of build metadata including status, size of the uploaded files, and a creation time stamp.
Create a Build with Files in Amazon S3

- Inline policy limiting access to read-only rights for a specified S3 bucket:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Action": [
            "s3:GetObject",
            "s3:GetObjectVersion",
            "s3:GetObjectMetadata"
         ],
         "Resource": "arn:aws:s3:::[BucketName]/*",
         "Effect": "Allow"
      }
   ]
}
```

3. **Send a request to create a new build.** Use the AWS CLI command `create-build` (or the AWS SDK operation `CreateBuild`) to create a new build record and tell Amazon GameLift where your build files are stored. In this request, you must specify an Amazon S3 location, including the following information (which you collected when setting up your bucket and access role):

- **Bucket:** The name of the bucket that contains your build. Example: "my_build_files".

- **Key:** The name of the .zip file that contains your build files. Example: "mygame_build_7.0.1, 7.0.2".

- **Role ARN:** The ARN assigned to the access role you created. Example: "arn:aws:iam::111122223333:role/GameLiftAccess".

In a command line window, type the following command and parameters.

```
aws gamelift create-build --operating-system [supported OS] --storage-location "Bucket=[S3 bucket label],Keys=[Build zip file name],RoleArn=[Access role ARN]" --name [user-defined name of build] --build-version [user-defined build number] --region [region name]
```

**Example:**

```
aws gamelift create-build --operating-system WINDOWS_2012 --storage-location "Bucket=gamelift-builds,Key=MyGame.zip,RoleArn=arn:aws:iam::401680102694:role/gamelift" --name "My Game Nightly Build" --build-version "build 255" --region us-west-2
```
In response to your request, the Amazon GameLift service returns the newly created build record.

**Update Your Build Files**

Once an Amazon GameLift build has been created, the build files associated with it cannot be changed. Instead, you must create a new Amazon GameLift build for each new set of files. If you provide build files using the `upload-build` command, you don't need to do anything special because Amazon GameLift automatically creates a new build record for each request. If you provide build files using the `create-build` command, upload a new build `.zip` file with a different name to Amazon S3 and create a build by referencing the new file name.

Try these tips for deploying updated builds:

- **Use aliases to seamlessly transfer players to a new game build.** When integrating your game client with Amazon GameLift, specify a fleet alias instead of a fleet ID. That way you can move players to a new build in just three steps: (1) Create the new build. (2) Create a fleet to deploy the new build. (3) Change the alias target from the old fleet to the new fleet. For more information, see Working with Aliases (p. 77).

- **Set up automated build updates.** Follow the GameDev blog post Automating Deployments to Amazon GameLift, with sample scripts, to incorporate Amazon GameLift deployments into your build system.
Working with Fleets

The topics in this section provide detailed help with setting up and managing your game server fleets. Fleets are, quite simply, your hosting resources in the form of EC2 instances. To host a game server on Amazon GameLift, you need to deploy a fleet with your game server build. The size of a fleet depends on the number of instances you give it, and you can adjust fleet size to meet player demand either manually or by auto-scaling.

Most games in production require multiple fleets. Each fleet deploys one game server build to a single AWS region. As a result, if you want to host players in more than one region, or if you have two or more versions of your game server build (such as free and premium versions), you'll need to have multiple fleets. There is a wide range of resources available; Learn more about AWS resources and Amazon GameLift service limits (p. 52) per AWS account and how to request more.

Tip
You don't need to have a game ready to start experimenting with Amazon GameLift. Learn more about available tools to get you exploring with Amazon GameLift features fast in Explore Amazon GameLift (p. 20).

Topics
- Choose Computing Resources (p. 52)
- Create a Fleet (p. 54)
- Update a Fleet (p. 59)
- Delete a Fleet (p. 60)
- Manage Fleet Capacity (p. 60)
- Spot Fleet Integration Guide (p. 69)
- Run Multiple Processes on a Fleet (p. 70)
- Remotely Access Fleet Instances (p. 72)
- Set Up VPC Peering (p. 74)

Choose Computing Resources

Amazon GameLift uses Amazon Elastic Compute Cloud (Amazon EC2) resources, called instances, to deploy your game servers and host game sessions for your players. When setting up a new fleet, you decide what type of instances your game needs and how to run game server processes on them (via the runtime configuration). Once a fleet is active and ready to host game sessions, you can add or remove instances at any time to accommodate more or fewer players. All instances in a fleet use the same type of resources and the same runtime configuration. You can edit a fleet's runtime configuration, but the type of resources cannot be changed.

When choosing resources for a fleet, you'll need to consider several factors, including game operating system, instance type (the computing hardware), and whether to use on-demand or spot instances. Each of these issues is discussed further in this topic. Keep in mind that hosting costs with Amazon GameLift primarily depend on the type of resources you use. Learn more about Amazon GameLift pricing.

Operating Systems

Amazon GameLift supports game server builds that run on either Microsoft Windows or Amazon Linux (see supported game server operating systems (p. 14)). When uploading a game build to Amazon
GameLift, you specify the operating system for the game. When you create a fleet to deploy the game build, Amazon GameLift automatically sets up instances with the build's operating system.

The cost of resources depends on the operating system in use. Learn more about the resources available for the supported operating systems:

- Microsoft Windows
- Amazon Linux

**Instance Types**

A fleet's instance type determines the kind of hardware that will be used for every instance in the fleet. Instance types offer different combinations of computing power, memory, storage, and networking capabilities. With Amazon GameLift you have a wide range of instance type options to choose from. View a list of available instance types or open the Service limits page in the Amazon GameLift console to view instance types, current usage, and usage limits. To learn more about the capabilities of each instance type, see Amazon EC2 Instance Types. Note that the instance types offered may vary depending on the region.

When choosing an instance type for your game, consider the following: (1) the computing requirements of your game server build, and (2) the number of server processes that you plan to run on each instance. You may be able to run multiple server processes on each instance by using a larger instance type, which can reduce the number of instances you need to meet player demand. However, larger instance types also cost more. Learn more about running multiple processes on a fleet (p. 70).

**On-Demand versus Spot Instances**

When creating a new fleet, you designate the fleet type as using either on-demand or spot instances. On-demand and spot instances offer exactly the same hardware and performance, based on the instance type chosen, and are configured in exactly the same way. They differ in available supply and in cost.

**On-demand instances**

On-demand instances are simply that: you request an instance and it is created for you. You can always acquire an on-demand instance when you need it and keep it as long as you want. On-demand instances have a fixed cost; you pay for the amount of time that you use them and there are no long-term commitments.

**Spot instances**

Spot instances offer a highly cost-efficient alternative to on-demand instances. Spot instances use spare AWS computing capacity, so spot instances can cost significantly less than on-demand instances. There is one drawback to using spot instances: unlike on-demand instances, spot prices fluctuate based on the current supply and demand for each instance type in each region. As a result, spot instances may be interrupted by AWS with a two-minute notification when AWS needs the capacity back.

With Amazon GameLift's FleetIQ, however, you can achieve cost savings while maintaining high game-server availability. When placing new game sessions, FleetIQ prioritizes the use of spot instances to locate the best available game server based on (1) instances with the highest cost savings, and (2) instances with historically low interruption rates.

You can evaluate a spot fleet's performance using a set of metrics generated by FleetIQ, including interruption rates for instances and game sessions. Learn more about Amazon GameLift metrics (p. 125). You can also view pricing history for any instance type by using the Amazon GameLift console. The Spot history page graphs on-demand and spot pricing and calculates the relative cost savings with spot. Use the controls to select an instance type, operating system, and a time range.

Learn more about how to use spot instances in the Spot Fleet Integration Guide (p. 69).
Instance Service Limits

AWS places limits how many Amazon EC2 instances (on-demand or spot) can be used by an AWS account. Each instance type has a maximum number allowed per account, and this limit varies by AWS region. Each account also is limited in the total number of instances used regardless of type of instance. You can access information on limits in several ways:

- Find general limits for Amazon GameLift, as well as all other AWS services on the AWS Service Limits page.
- See limits for a specific region in the Amazon GameLift console: Select a region and choose Service limits from the Amazon GameLift menu. You can also view the total number of instances currently in use in the region.
- Retrieve the maximum number of instances per AWS account (by region) by using the Amazon GameLift API action DescribeEC2InstanceLimits. This action also returns the number for instances currently active in the region.

If you need more instances than allowed by AWS service limits, you can request an increase on the Amazon GameLift Service limits page of the AWS Management Console.

Create a Fleet

You can create a new fleet to host game servers for any game build that has been uploaded to the Amazon GameLift service and is in a Ready status. Use either the Amazon GameLift console or the AWS Command Line Interface (CLI) to create a fleet. You can change a fleet's configuration by editing a fleet (p. 59).

Create a Fleet (Console)

To create a fleet with the Amazon GameLift console:

1. Open the Amazon GameLift console at https://console.aws.amazon.com/gamelift/.
2. On the Builds page, find the build that you want to create a fleet for and verify that its status is Ready.
3. Select the build (use the option button to the left of the build status) and click Create fleet from build to open the Create fleet page.
4. On the Create fleet page, complete the Fleet Details section:
   - Name – Create a meaningful fleet name so you can easily identify it in a list and in metrics.
   - Description – (Optional) Add a short description for this fleet to further aid identification.
   - Fleet type – Choose whether to use on-demand or spot instances for this fleet. Learn more about fleet types in Choose Computing Resources (p. 52).
   - Metric group – (Optional) Enter the name of a new or existing fleet metric group. When using Amazon CloudWatch to track your Amazon GameLift metrics, you can aggregate the metrics for multiple fleets by adding them to the same metric group.
   - Build – If you Note that the build information, including name, ID and operating system, is automatically filled in.
5. In the Instance type section, select an Amazon EC2 instance type from the list. The instance types listed vary depending several factors, including the current region, the operating system of the selected game build, and the fleet type (on-demand or spot). Learn more about choosing an instance type in Choose Computing Resources (p. 52). Once this fleet is created, you cannot change the instance type.
Amazon GameLift provides a free tier for one year with a c4.large instance. The free tier instance type is available for on-demand fleets only. When using a free-tier instance type, you may want to set up billing alerts or turn on safe scaling for the fleet to avoid incurring charges in excess of the free tier. For more information, see Free Tier and Billing Alerts (p. 16).

6. In the Process management section, configure how you want server processes to run on each instance.

   a. **Server process allocation:**

      Specify the type and number of game server processes you want to run on each instance. Each fleet must have at least one server process configuration defined and can have multiple configurations. For example, if you have multiple server executables in your game build, you must have a configuration for each executable.

      - **Launch path** – Type the path to the game executable in your build. All launch paths must start with the game server location, which varies based on the operating system in use. On Windows instances, game servers are built to the path `C:\game`. On Linux instances, game servers are built to `/local/game`, so all launch paths must start with this location. Examples: `C:\game\MyGame\server.exe` or `/local/game/MyGame/server.exe`.

      - **Launch parameters** – (Optional) You can pass information to your game executable at launch time. Type the information as a set of command line parameters here. Example: `+sv_port 33435 +start_lobby`.

      - **Concurrent processes** – Indicate how many server processes with this configuration to run concurrently on each instance in the fleet. Check Amazon GameLift’s limits on number of concurrent server processes; they depend on which SDK your game server uses.

      Once you enter a server process configuration, click the green checkmark button to save the configuration. To add additional server process configurations, click Add configuration.

      Limits on concurrent server processes per instance apply to the total of concurrent processes for all configurations. If you’re limited to one process, you can have only one configuration, and concurrent processes must be set to 1. If you configure the fleet to exceed the limit, the fleet will not be activated.

      The collection of server process configurations is called the fleet's runtime configuration. It describes all of the server processes that will be running on each instance in this fleet at any given time.

   b. **Game session activation:**

      Set the following limits to determine how new game sessions are activated on the instances in this fleet:

      - **Max concurrent game session activation** – Limit the number of game sessions on an instance that can be activating at the same time. This limit is useful when launching multiple new game sessions may have an impact on the performance of other game sessions running on the instance.

      - **New activation timeout** – This setting limits the amount of time Amazon GameLift allows for a new game session activate. If the game session does not complete activation an move to status ACTIVE, the game session activation is terminated.

    7. In the Safe scaling policy section, turn on this auto-scaling policy to ensure that your fleet scales down to zero instances when there is no activity. Once the fleet is created, you can turn this policy on or off or set custom auto-scaling policies on the fleet's detail page (see the Scaling tab).

    8. In the EC2 port settings section, click Add port settings to define access permissions for inbound traffic connecting to server processes deployed on this fleet. You can create multiple port settings.
for a fleet. At least one port setting must be set for the fleet before access is allowed. If you don't specify port settings at this time, you can edit the fleet later.

- **Port range** – Specify a range of port numbers that your game servers can use to allow inbound connections. A port range must use the format nnnnn[–nnnnn], with values between 1025 and 60000. Example: 1500 or 1500–20000.
- **Protocol** – Select the type of communication protocol for the fleet to use.
- **IP address range** – Specify a range of IP addresses valid for instances in this fleet. Use CIDR notation. Example: 0.0.0.0/0 (This example allows access to anyone trying to connect.)

9. In the **Resource creation limit** section, click **Add resource creation limits** to set up a policy that limits the number of game sessions any one player can create in a specified period of time. This limit protects your available fleet resources from excessive consumption. To use this feature, requests for new game sessions must specify a creator.

- **Game sessions per policy period** – Specify the number of game sessions one player (based on player ID) is allowed to create during the policy period.
- **Policy period** – Specify the amount of time, in minutes, over which to limit game session creation per player. Amazon GameLift evaluates each new game session request to determine whether the creator has exceeded the creation limit in the most recent span of time.

10. In the **Protection policy** section, choose whether to apply game session protection to instances in this fleet. Instances with protection are not terminated during a scale down event if they are hosting an active game session. You can also set protection for individual game sessions. Once the fleet is created, you can edit the fleet to change the fleet-wide protection policy.

11. Once you've finished setting the configuration for your new fleet, click **Initialize fleet**. Amazon GameLift assigns an ID to the new fleet and begins the fleet activation process. You can view the new fleet's status on the **Fleets** page. Once the fleet is active, you can change the fleet's capacity (p. 62), runtime configuration, and other configuration settings as needed.

---

**Create a Fleet (AWS CLI)**

To create a fleet with the AWS CLI, open a command line window and use the `create-fleet` command to define a new fleet. See complete documentation on this command in the AWS CLI Command Reference. If you haven't yet installed the AWS CLI, see the topic Install the AWS CLI (p. 18).

The example `create-fleet` request shown below creates a new fleet with the following characteristics:

- The fleet will use `c4.large` on-demand instances with the operating system required for the selected game build.
- It will deploy the specified game server build, which must be in a **Ready** status.
- Each instance in the fleet will run ten identical processes of the game server concurrently, enabling each instance to host up to ten game sessions simultaneously.
- On each instance, Amazon GameLift will allow only two new game sessions to be activating at the same time. It will also terminate any activating game session if it is not ready to host players within 300 seconds.
- All game sessions hosted on instances in this fleet have game session protection turned on. It can be turned off for individual game sessions.
- Individual players can create three new game sessions within a 15-minute period.
- Each game session hosted on this fleet will have a connection point that falls within the specified IP address and port ranges.
- Metrics for this fleet will be added to the EMEA fleets metric group, which (in this example) combines metrics for all fleets in EMEA regions.
Create a Fleet (AWS CLI)

```bash
$ aws gamelift create-fleet
   --name "SampleFleet123"
   --description "The sample test fleet"
   --ec2-instance-type "c4.large"
   --fleet-type "ON_DEMAND"
   --build-id "build-92f061ed-27c9-4a02-b1f4-6f85b2385620"
   --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"
   --MetricGroups "EMEAfleets"
```

Copiable version:

```bash
aws gamelift create-fleet --name "SampleFleet123" --description "The sample test fleet" --fleet-type "ON_DEMAND" --MetricGroups "EMEAfleets" --build-id "build-92f061ed-27c9-4a02-b1f4-6f85b2385620" --ec2-instance-type "c4.large" --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"
```

If the create-fleet request is successful, Amazon GameLift returns a set of fleet attributes that includes the configuration settings you requested and a new fleet ID. Amazon GameLift immediately initiates the fleet activation process and sets the fleet status 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`

Once the fleet is active, 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 a VPC Peering Connection with a New Fleet

You can create a new fleet and request a VPC peering connection for the new fleet at the same time. For more information on using VPC peering with Amazon GameLift, see Set Up VPC Peering (p. 74).
Before requesting a peering connection for a new fleet, you must first take the following steps: (1) Create a VPC for a set of resources on your AWS account (if you don't already have one) and get the VPC ID; and (2) call the Amazon GameLift command `create-vpc-peering-authorization` to pre-authorize the peering request from Amazon GameLift.

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 request's status is set to `initiating-request`. You can track the success or failure of the peering request by calling `DescribeVpcPeeringConnections()`. Please note that the VPC peering connection is not completed until the fleet is ready to become active; as a result, it is not available when the game server build is being installed on the new fleet.

When requesting a VPC peering connection with a new fleet, both actions either succeed or fail. If a fleet fails during the fleet creation process, the VPC peering connection will not complete successfully. Likewise, if a VPC peering connection fails for any reason, the new fleet will fail to move from status `Activating` to `Active`.

Using the example from the previous section, add the following parameters for VPC on your AWS account that you've authorized. This generates a VPC peering connection between the VPC that is created for the new fleet and the VPC on your AWS account. The combination of your AWS account ID and VPC ID uniquely identifies the VPC to peer with. This VPC must be in the same region as the fleet you're creating.

```
--peer-vpc-aws-account-id "111122223333"
--peer-vpc-id "vpc-a12bc345"
```

### Debug Fleet Creation Issues

Understanding how Amazon GameLift sets up a new fleet can help you resolve fleet activation errors.

When a new fleet is created, the Amazon GameLift service performs a series of tasks to prepare the first instance for hosting. Once that first instance is set up, it is used to quickly start new instances as needed. The fleet status indicates which phase it is in. Problems during fleet creation will cause the fleet status to go to `Error` with meaningful error messaging (such as an incorrect build path or a service error). These phases are:

- **New** – Create a fleet record and designate resources for the first instance.
- **Downloading** – Download the game build files to the instance.
- **Validating** – Validate that the build files were successfully downloaded.
- **Building** – Install the game server build onto the instance, including running an install script.
- **Activating** – Start server process(es) as specified in the fleet's runtime configuration, and verify processes are communicating with the Amazon GameLift service by reporting readiness to host a game session.

As soon as one server process notifies Amazon GameLift that it is ready, the fleet status moves to `Active`.

The most common problem that game developers experience when creating a new fleet is having a fleet get stuck in an `Activating` status. Common reasons that a fleet might fail to activate include:

- **Server processes start but fail quickly or report poor health.** Other than issues with your game build, this 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 version.
configuration using either the Amazon GameLift console (edit the fleet's capacity allocation settings) or by calling the AWS CLI command `update-runtime-configuration`.

- **Server processes fail to start.** Assuming there are no issues with your game build and/or installation script, check that you've correctly set the launch path and optional parameters in the fleet's runtime configuration. You can view the fleet's current runtime configuration using either the Amazon GameLift console (see the Fleet detail page, Capacity Allocation tab) or by calling the AWS CLI command `describe-runtime-configuration`.

- **Server processes start but fleet fails to activate.** If server processes start and continue to run successfully, but the fleet does not move to **Active** status, a likely cause is that the server process is failing to notify Amazon GameLift that it is ready to host game sessions. Check that your game server is correctly calling the Server API action `ProcessReady()` (see Prepare a Server Process).

For additional troubleshooting, you can remotely access a fleet instance. See Remotely Access Fleet Instances.

**Update a Fleet**

Use the **Edit fleet** page in the Amazon GameLift console to change a fleet's configuration. All fleet properties can be changed except for the build ID and the instance type. To change scaling settings, see Auto-Scale Fleet Capacity.

**Note**

If a fleet's build has been deleted or an error occurred while retrieving the build, you may see one of the following **Build** statuses:

- **Deleted** – The build for this fleet was deleted. Your fleet will still run properly despite the build having been deleted.
- **Error** – An **error** occurred while attempting to retrieve build information for the fleet.

**To update a fleet configuration**

1. Open the Amazon GameLift console at [https://console.aws.amazon.com/gamelift/](https://console.aws.amazon.com/gamelift/).
2. Choose **Fleets** from the menu bar to open the **Fleets** page, which displays a list of all fleets created for this account.
3. On the **Fleets** page, click the name of the fleet you want to edit. Only fleets in status **ACTIVE** can be edited.
4. On the selected fleet page, under **Actions**, choose **Edit fleet**. The selected fleet's current configuration is displayed.
5. On the **Edit fleet** page, you can change the following fields:

   - **Name** – Change the friendly name for your fleet.
   - **Description** – Add or change the description of the fleet.
   - **Metric group** – Add or change a metric group name. Names can be new or existing metric groups. Metric groups are used in Amazon CloudWatch to track aggregated Amazon GameLift metrics for all fleets in a metric group.
   - **Process management** – Change how you want server processes to run on each instance in the fleet.
     - Server process allocation – Add, change, or remove the type and number of game server processes you want to run on each instance.
     - Game session activation – Reset the limits on game session activations for each instance in the fleet.
Delete a Fleet

Delete a Fleet

In the process of testing or running multiple versions of your game, you might create dozens of fleets that eventually become unnecessary to maintain. You can delete fleets that you no longer need. Deleting a fleet also permanently removes associated game sessions and collected data.

Note
You can delete a fleet only after you have scaled it down to 0. Set the desired number of EC2 instances to 0 and then wait for the scaled down state to take effect before you delete the fleet.

To delete a fleet

1. In the Amazon GameLift console, choose Fleets from the menu bar.
2. On the Fleets page, select the fleet you want to delete.
3. On the selected fleet page, for Actions, choose Terminate fleet.
4. In the Terminate fleet dialog box, confirm the deletion by typing the name of the fleet.
5. Click Delete.

Manage Fleet Capacity

Fleet capacity, measured in instances, determines the number of game sessions (and players) the fleet can host. One of the most challenging tasks with game hosting is to strike a balance between maintaining enough capacity to accommodate every new player and not wasting money on unnecessary resources. Learn more about how capacity scaling works (p. 3) in Amazon GameLift.

You have full control over scaling a fleet. You can set capacity to a specific number of instances, or you can take advantage of auto-scaling to adjust capacity based on actual player demand. We recommended that you start by turning on the auto-scaling option Target Tracking. Target tracking is an effective and easy to use tool for that helps you maintain just enough hosting resource to accommodate current player demand and handle sudden spikes. For the vast majority games, target tracking will be the only solution you need.

The topics in this section provide detailed help with the following tasks:

- Set minimum and maximum limits for capacity scaling (p. 61)
- Manually set capacity levels (p. 62)
- Turn on auto-scaling with Target Tracking (p. 64)
- Manage rule-based auto-scaling (advanced feature) (p. 66)
- Temporarily disable auto-scaling (p. 63)
Set Fleet Capacity Limits

Fleet size is determined by the number of instances it contains. Each fleet has a defined minimum and maximum limit, which you can set as needed for your game. All requests to change fleet capacity (either by auto-scaling or manual adjustment) must fall within the current limits. By default, limits on new fleets are set to a minimum of 0 instances and a maximum of 1 instance. Before scaling up a fleet, you'll need to adjust the fleet's limits.

If you're auto-scaling a fleet, the maximum limit lets Amazon GameLift scale up the fleet as needed to accommodate player demand while also preventing runaway hosting costs, such as might occur during a DDOS attack. Set up CloudWatch to alarm when capacity approaches the maximum limit, so you can evaluate the situation and manually adjust as needed. (You can also set up a billing alert to monitor AWS costs.) The minimum limit is useful when you want to ensure that you always have some hosting availability at all times.

Limits also apply to manually scaled fleets. Before you can adjust fleet capacity to a value outside the limit range, you must change the limits. Since fleet capacity has such a significant effect on game availability and player experience, the limits feature provides an additional layer of control over capacity.

You can set a fleet's capacity limits in the Amazon GameLift console or by using the AWS CLI. The fleet's status must be Active.

To Set Capacity Limits (Console)

1. Open the Amazon GameLift console at https://console.aws.amazon.com/gamelift/.
2. On the Fleets page, click the name of an active fleet to open the fleet's detail page. (You can also access a fleet's detail page via the Dashboard.)
3. Open the Scaling tab to view historical scaling metrics and to view or change the current settings. Scaling settings are located below the metrics graph.
4. Under Instance Limits, set minimum and maximum instance counts. For each control, commit your changes by clicking the checkmark button.

If the fleet's current desired instance count is outside the new limit range, you'll get an error. In this case, you must first adjust the fleet's desired instance count so that it falls inside the new limit range.
range. This can be done on the Scaling tab. If the fleet uses auto-scaling, you’ll need to disable auto-scaling, manually adjust the desired instance count and set the new limit range, and then re-enable auto-scaling.

The new limits are immediately reflected in the graph at the top of the Scaling tab.

**To Set Capacity Limits (AWS CLI)**

1. **Check current capacity settings.** In a command line window, use the `describe-fleet-capacity` command with the fleet ID of the fleet you want to change capacity for. This command returns a `FleetCapacity` object that includes the current instance count and capacity limits. Determine whether the new instance limits will accommodate the current desired instances setting.

   ```bash
   aws gamelift describe-fleet-capacity --fleet-id <unique fleet identifier>
   ```

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.

   ```bash
   --fleet-id <unique fleet identifier>
   --max-size <maximum capacity for auto-scaling>
   --min-size <minimum capacity for auto-scaling>
   --desired-instances <fleet capacity as an integer> [Optional]
   ```

   Example:

   ```bash
   aws gamelift update-fleet-capacity
   --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa
   --max-size 10
   --min-size 1
   --desired-instances 10
   ```

   Copyable version:

   ```bash
   aws gamelift update-fleet-capacity --fleet-id
   fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa --max-size 10 --min-size 1 --desired-instances 10
   ```

   If your request is successful, the fleet ID is returned. If the new `max-size` or `min-size` value conflicts with the current `desired-instances` setting, an error is returned.

**Manually Set Fleet Capacity**

When you create a new fleet, fleet capacity is automatically set to one instance. In response, Amazon GameLift starts one new instance with game server processes as configured. To change fleet capacity, you can either turn on auto-scaling or you can manually set the number of instances you want for the fleet. Learn more about Managing Capacity and Scaling (p. 3).

Setting a fleet’s capacity manually can be useful when auto-scaling is not needed or when you need to hold capacity at an arbitrary number of instances temporarily or permanently. When manually setting a desired capacity, keep in mind that this action will affect actual fleet capacity only if (1) there are no auto-scaling policies for the fleet, or (2) auto-scaling has been disabled. If auto-scaling is enabled, it will immediately reset the desired capacity based on its own scaling rules.
You can manually set fleet capacity in the Amazon GameLift console or by using the AWS CLI. The fleet's status must be **Active**.

### Disabling Auto-Scaling

Disabling auto-scaling allows you to turn off all auto-scaling for a fleet and return to manual scaling. While you can always delete a fleet's autoscaling policy, this feature lets you temporarily turn off auto-scaling but retain the policies for future use. For example, if you want to scale up in preparation for an upcoming major event, you can disable auto-scaling, manually set a desired fleet capacity, and then, once the event is in progress, re-enable auto-scaling. When auto-scaling is disabled, all auto-scaling activities for the fleet are stopped, including all currently active policies and any policies that might be created in the future. You can enable/disable auto-scaling in the Amazon GameLift console (see Step 4 in "To manually set capacity").

### To Manually Set Capacity (Console)

1. Open the Amazon GameLift console at [https://console.aws.amazon.com/gamelift/](https://console.aws.amazon.com/gamelift/).
2. On the **Fleets** page, click the name of an active fleet to open the fleet's detail page. (You can also access a fleet's detail page via the **Dashboard**.)
3. Open the **Scaling** tab to view historical scaling metrics and to view or change the current settings. Scaling settings are located below the metrics graph.
4. Under **Auto-Scaling Policies**, check the box "Disable all scaling policies" and commit your change by clicking the check mark button. This setting stops all auto-scaling actions for the fleet. It is a good idea to do this even if the fleet currently has no policies, as it will prevent any newly created policies from taking effect. Once you commit this change, the table listing active scaling policies indicates that all policies are "disabled".
5. Under **Auto-Scaling Policies**, for the option "Manually adjust the desired instance count to...", specify the number of instances for the fleet. This value tells Amazon GameLift how many instances to maintain in an active state, ready to host game sessions. Commit the change by clicking the checkmark button.

   If the new desired instances value is outside the fleet's capacity limits, you'll get an error. In this case, you must first adjust the fleet's instance limits to allow for the new desired instance count. Instance Limits can also be set on the **Scaling** tab.

As soon as you commit changes to instance limits and manual scaling levels, the new values are reflected in the graph at the top of the **Scaling** tab. Amazon GameLift immediately begins responding to the changes by deploying additional instances or shutting down unneeded ones. As this process is completed, the number of **Active** instances change to match the newly updated desired value. This process may take a little time.

### To Manually Set Capacity (AWS CLI)

1. **Check current capacity settings.** In a command line window, use the `describe-fleet-capacity` command with the fleet ID of the fleet you want to change capacity for. This command returns a **FleetCapacity** object that includes the current instance count and capacity limits. Determine whether the new instance count falls between the minimum and maximum limits.

   ```bash
   aws gamelift describe-fleet-capacity --fleet-id <unique fleet identifier>
   ```

2. **Update desired capacity.** Use the `update-fleet-capacity` command with the fleet ID and a new **desired-instances** value. If this value falls outside the current limit range, include adjust limit values in the same command.

   ```bash
   --fleet-id <unique fleet identifier>
   ```
Auto-Scale Fleet Capacity

Use auto-scaling 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 and fast player experience. Learn more about how auto-scaling (p. 3) works in Amazon GameLift.

Auto-scaling is done by creating scaling policies that provide instructions to Amazon GameLift for scaling up or down. There are two types of scaling policies, target-based and rule-based. The target-based approach—target tracking—offers a complete solution; it is recommended as the simplest and most effective option. Rule-based scaling policies, which require you to define each aspect of the auto-scaling decision-making process, is useful for addressing specific issues. It works best as a supplement to target-based auto-scaling.

Target-based auto-scaling can be managed using either the Amazon GameLift Console or the AWS CLI or AWS SDK. Rule-based auto-scaling is managed using the AWS CLI or AWS SDK only, although you can view rule-based scaling policies in the Console.

### Topics

- Auto-Scale with Target Tracking (p. 64)
- Auto-Scale with Rule-Based Policies (p. 66)

### Auto-Scale with Target Tracking

Target tracking adjusts capacity levels based on a single key fleet metric: "percent available game sessions". This metric measures the number of available game session slots at current capacity—additional game sessions that could be started immediately. In effect, this metric represents the fleet's buffer against 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, the time it takes to get new players into game sessions can be measured in seconds. If no resources are available, players must wait for existing game sessions to end or for new resources to become available. The time required to start up new instances and server processes can be minutes.
When setting up target tracking, you simply specify the size of the buffer you want the fleet to maintain. Since the metric "percent available game sessions" measures the percentage of available resources, the actual buffer size is a percentage of the total fleet capacity. Amazon GameLift adds or removes as many instances as are needed to maintain the target buffer size. Choosing a buffer size depends on how you want to prioritize minimizing player wait time against controlling hosting costs. With a large buffer, you minimize wait time but you also pay for extra resources that may not get used. If your players are more tolerant of wait times, you can lower costs by setting a small buffer.

**Set Target Tracking (Console)**

1. Open the Amazon GameLift console at https://console.aws.amazon.com/gamelift/.
2. On the **Fleets** page, click the name of an active fleet to open the fleet's detail page. (You can also access a fleet's detail page via the **Dashboard**.)
3. Open the **Scaling** tab. This tab displays the fleet's historical scaling metrics and contains controls for adjusting current scaling settings. Scaling settings are located below the metrics graph.
4. Under **Instance Limits**, check that the minimum and maximum limits are appropriate for the fleet. With auto-scaling enabled, capacity may adjust to any level between these two limits.
5. Under **Auto-Scaling Policies**, check the option to **Maintain a buffer of X percent game session availability**. Set a buffer size, and click the checkmark button to save the auto-scaling settings. Once you've saved the settings, a new target-based policy is added to the **Scaling policies** table.
6. To turn on auto-scaling for the fleet, verify that the option to **Disable all scaling policies in the fleet** is unchecked. If this option is checked, all policies, including the new target-tracking policy, are disabled. This state is reflected in the **Scaling policies** table.

**Set Target Tracking (AWS CLI)**

1. **Set capacity limits.** Set either or both limit values using the `update-fleet-capacity` command. For help, see To Set Capacity Limits (AWS CLI) (p. 62).
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.

   ```bash
   --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"
   
   Copyable version:
   
   $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 provide fine-grained control when auto-scaling a fleet's capacity in response to player activity. For each policy, you can link fleet scaling to one of several available fleet metrics, identify a trigger point, and customize the responding scale-up or scale-down event. Rule-based policies are particularly useful for supplementing target-based scaling to handle special circumstances.

A rule-based policy makes the following statement: "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 the AWS SDK or AWS CLI with the Amazon GameLift Service API. You can view all active policies in the Amazon GameLift console.

To temporarily disable 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 help, see To Set Capacity Limits (AWS CLI) (p. 62).

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.

```
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
```

Copyable version:

```
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:

```bash
aws gamelift delete-scaling-policy
--fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa
--name "Scale up when AGS<50"
```

Copyable version:

```bash
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 rule-based scaling policy statement, you must 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 triggers a scale-up event whenever a fleet's extra capacity (available hosting resources not currently in use) is less than what is needed to handle 50 new game sessions:

If AvailableGameSessions remains at less than 50 for 15 minutes, then change fleet capacity using ChangeInCapacity by 10 instances.

**Metric name**

To trigger a scaling event, link an auto-scaling policy to one of the following fleet-specific metrics. See Amazon GameLift Metrics for Fleets (p. 125) for more complete metric descriptions.

- 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

The following metrics may be used if the fleet is included in a game session queue:

- Queue depth (fleet specific) – The number of pending game session requests for which this fleet is the best available hosting location.
- Wait time (fleet specific) – Fleet-specific wait time. The length of time that the oldest pending game session request has been waiting to be fulfilled. As with queue depth, this metric reflects only game session requests for which this fleet is the best available hosting location. A fleet's wait time is equal to the oldest current request's time in queue.

**Comparison operator**

This variable tells Amazon GameLift how to compare the metric data to the threshold value. Valid comparison operators include greater than (>), less than (<), greater than or equal (>=), or less than or equal (<=).
Threshold value

When the specified metric value meets or crosses the threshold value, it can trigger a scaling event. Depending on the metric selected, it may indicate an amount of player sessions, game sessions, instances, or game session requests. 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 triggering 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 should adjust the fleet's capacity when a scaling event is triggered. Choose from three possible types of adjustments:

- **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, an value of "-10" will scale 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 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" will add ten instances to the fleet.

- **Exact capacity** – Set desired instances 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 in force 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. However, there are no limits on using multiple policies.

Multiple policies behave independently. Keep in mind that there is no way to control the sequence of scaling events. For example, if you have multiple policies driving scaling up, it is possible that player activity could trigger multiple scaling events simultaneously. For example, the effects of two scale up policies can easily be compounded if it is possible for player activity to trigger both metrics. Also watch for policies that trigger each other. For example, you could create an infinite loop if you create scale up and scale down policies that sets capacity beyond the threshold of each other.

**Set maximum and minimum capacity**

Each fleet has a maximum and minimum capacity limit. This feature is particularly important when using auto-scaling. Auto-scaling will never set 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, you must increase the maximum value.

Fleet capacity is also constrained by limits on the fleet's instance type and on your AWS account. You cannot set a minimum and maximum that is outside the service and account limits.

**Track metrics after a change in capacity**

After changing capacity in response to an auto-scaling policy, Amazon GameLift waits ten minutes before responding to triggers from the same policy. This wait allows Amazon GameLift 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 continues to evaluate the policy against the metric and track the policy's evaluation period, which restarts once a scaling event is triggered. This means that a scaling policy could trigger another scaling event immediately after the wait time is over.

There is no wait time between scaling events triggered by different auto-scaling policies.

Spot Fleet Integration Guide

When setting up fleets of hosting resources, you have the option of using both spot instances and on-demand instances. Learn more about how Amazon GameLift uses spot instances in On-Demand versus Spot Instances (p. 53). If you choose to use spot fleets, you'll need to make a few adjustments to your game integration.

Are you using FlexMatch for matchmaking? You can use spot fleets with your existing queues for matchmaking placements.

To set up game and hosting resources for spot fleets:

1. **Design a queue with a "spot-optimized" configuration.** Managing fleet usage with a queue is always a best practice, but it's required when using spot instances. For this step, you'll need to identify which regions to cover in the queue, select the right instance types (hardware) for your game, and decide how you want FleetIQ to prioritize game session placement. As a best practice, each region supported by your queue should have more than one spot fleet and one on-demand fleet. The spot fleets should use different instance types, ideally in the same family (C4.large, C4.xlarge, etc.). This design increases the resiliency of your game servers. See Design a Queue for Spot Instances (p. 82) for help designing a spot-optimized queue.

   Once you've created the fleets for your queue, you can create the queue, add fleet destinations, and set fleet prioritization. See Create a Queue (p. 83) for help creating and configuring the new queue.

2. **Create the fleets for your spot-optimized queue.** Before creating the queue, you need to create the fleets for it. Spot fleets and on-demand fleets can have identical configurations, with the only difference being the fleet type designation. See Create a Fleet (p. 54) for help creating and configuring new fleets.

   Note
   We recommend that you include the fleet type (spot or on-demand) in the fleet name. This will make it easy to identify your spot fleets at a glance in a list of fleet names, such as when adding fleets to a queue.

3. **Enable your game client or client service to place new game sessions using the spot-optimized queue.** If you're not already doing so, update your game client code to use game session placement with a queue rather than creating game sessions on a single specific fleet. By using the queue to place game sessions, you're enabling FleetIQ to select the best available fleet based on cost, interruption rate, and player latency. You're also providing fallback hosting resources in case a preferred fleet is temporarily not available for use. For help implementing game session placements in your game client, see Create Game Sessions (p. 35).

4. **Enable your game server to handle a spot interruption.** Spot instances can be interrupted with two minutes' notification when AWS needs the capacity back. You'll want your game server to handle an interruption gracefully, if it happens, and minimize the player impact. When AWS is about to reclaim a spot instance, it sends termination notification up to two minutes before, which Amazon GameLift passes on to all affected server processes. This is done by invoking the Server SDK callback function onProcessTerminate(). Implementing this callback function is always a best practice, but it is particularly important when using spot instances. The implementation of this callback can take action to either gracefully end the game sessions or find a way to move the game sessions
Run Multiple Processes

This topic provides additional information on how multiple processes per instance are managed on a fleet and how you can make use of this feature for your games. Depending on how you configure your fleet, running multiple processes gives you greater control over how efficiently you use your Amazon GameLift resources, which can potentially reduce overall operating costs for your game.

Optimizing for Multiple Processes

At a minimum, you must do the following to enable multiple processes:

- Create a build (p. 46) that contains all of the game server executables that you want to deploy to a fleet and upload it to Amazon GameLift. All game servers in a build must run on the same platform and be integrated with Amazon GameLift using the Amazon GameLift Server SDK for C++, version 3.0.7 or later.
- Create a runtime configuration with one or more server process configurations and multiple concurrent processes.
- Game clients connecting to games hosted on this fleet must be integrated using the AWS SDK, version 2016-08-04 or later.

In addition, implement the following in your game servers to optimize fleet performance:

- Handle server process shutdown scenarios to ensure that Amazon GameLift can recycle processes efficiently. If you don't do this, server processes can't be shut down until they fail.
- Add a shutdown procedure to your game server code, ending with the server API call to `ProcessEnding()`.
• Implement the callback function `OnProcessTerminate()` in your game server code to gracefully handle termination requests from Amazon GameLift.

• Make sure that “unhealthy” server processes are shut down and relaunched quickly by defining what “healthy” and “unhealthy” mean and reporting this status back to Amazon GameLift. You do this by implementing the `OnHealthCheck()` callback function in your game server code. Amazon GameLift automatically shuts down server processes that are reported unhealthy for three consecutive minutes. If you don’t implement `OnHealthCheck()`, Amazon GameLift assumes a server process is healthy unless it fails to respond. As a result, poorly performing server processes can continue to exist, using up resources until they finally fail.

How a Fleet Manages Multiple Processes

Amazon GameLift uses a fleet runtime configuration to manage what processes to maintain on each instance in the fleet. A runtime configuration is actually made up of one or multiple server process configurations, each of which identifies the following:

• The path and file name of a server executable in the game build deployed on the fleet
• (Optional) Parameters to pass to the server process on launch
• The number of this server process to maintain concurrently on the instance

When an instance is started in the fleet, the instance immediately begins launching server processes called for in the runtime configuration. Server process launches on an instance are staggered by a few seconds, so depending on the total number of server processes configured for an instance, it may take a few minutes to achieve full capacity.

Over time, server processes end, either by self-terminating (calling the Server SDK `ProcessEnding()` ) or by being terminated by Amazon GameLift. An instance regularly checks that it is running the number and type of server processes specified in the runtime configuration. If not, the instance automatically launches server processes as needed to meet the runtime configuration requirements. As a result, as server processes end, their resources are continually recycled to support new server processes, and instances generally maintain the expected complement of active server processes.

You can change a fleet’s runtime configuration at any time by adding, changing, or removing server process configurations. Here’s how Amazon GameLift adopts runtime configuration changes.

1. Before an instance checks that it is running the correct type and number of server processes, it first requests the latest version of the fleet’s runtime configuration from the Amazon GameLift service. If you’ve changed the runtime configuration, the instance acquires the new version and implements it.

2. The instance checks its active processes against the current runtime configuration and handles discrepancies as follows:
   • The updated runtime configuration removes a server process type. Active server processes that no longer match the runtime configuration continue to run until they end.
   • The updated runtime configuration decreases the number of concurrent processes for a server process type. Excess server processes of that type continue to run until they end.
   • The updated runtime configuration adds a new server process type or increases the concurrent processes setting for an existing type. New server processes are launched immediately to match the updated runtime configuration, unless the instance is already running the maximum number of server processes. In this case, new server processes are launched only when existing processes end.

Choosing the Number of Processes per Instance

There are effectively three limits to keep in mind when deciding on the number of concurrent processes:
• Amazon GameLift limits each instance to a maximum number of concurrent processes. Whether your runtime configuration has one or multiple server process configurations specified, the sum of all concurrent processes for a fleet's server process configurations can't exceed this limit.

• The Amazon EC2 instance type that you choose may limit the number of processes that can run concurrently while offering acceptable performance levels. You need to test different configurations for your game to find the optimal number of processes for your preferred instance type. Factors that may affect your choice include the resource requirements of your game server, the number of players to be hosted in each game session, and player performance expectations.

• When changing a fleet's runtime configuration, keep in mind that Amazon GameLift will never run more concurrent processes than the total number configured. This means that the transition from old runtime configuration to new may happen gradually, with new processes starting only as existing processes end. Here's an example: You have a fleet that is configured to run 10 concurrent processes of your server executable, `myGame.exe`, with launch parameters set to `-loglevel=1`. You update the configuration to continue running 10 concurrent processes of `myGame.exe` but change the launch parameters to `-loglevel=4`. Since instances in the fleet are already running 10 processes, Amazon GameLift waits to start a process with the new launch parameters until a process with the old launch parameters ends.

Remotely Access Fleet Instances

You can remotely access any fleet instance that is currently running in an Amazon GameLift fleet. This capability is useful for troubleshooting fleet activation issues. You can also use this feature to get real-time game server activity, such as track log updates or run benchmarking tools using actual player traffic.

When remotely accessing individual Amazon GameLift instances, keep the following in mind:

• The Amazon GameLift service continues to manage fleet activity and capacity. Establishing a remote connection to an instance does not affect how Amazon GameLift manages it in any way. As a result, the instance continues to execute the fleet runtime configuration, stop and start server processes, create and terminate game sessions, and allow player connections. In addition, the Amazon GameLift service may terminate the instance at any time as part of a scale down event.

• Making local changes to an instance that is hosting active game sessions and has live players connected may significantly affect player experience. For example, your local changes have the potential to drop individual players, crash game sessions or even shut down the entire instance with multiple game sessions and players affected.

For more information on how games are deployed and managed on Amazon GameLift instances, see the following topics:

• How Amazon GameLift Works (p. 2)
• Debug Fleet Creation Issues (p. 58)
• How a Fleet Manages Multiple Processes (p. 71)

Connect to an Instance

You can access remote instances that are running either Windows or Linux. To connect to a Windows instance, use a remote desktop protocol (RDP) client. To connect to a Linux instance, use an SSH client.

Use the AWS CLI get the information you need to access a remote instance. For help, see the AWS CLI Command Reference. If you haven't yet installed the AWS CLI, see the topic Install the AWS CLI (p. 18).
Connect to an Instance

1. **Find the ID of the instance you want to connect to.** When requesting access, you must specify an instance ID. You can get information on all fleet instances using the command `describe-instances` with a fleet ID. This example retrieves the first three instances in a fleet:

   ```bash
   # aws gamelift describe-instances
   --fleet-id "fleet-a7abc071-5537-4f0f-b5ee-1b5c1187565f"
   --limit 3
   ```

2. **Request access credentials for the instance.** Once you have an instance ID, use the command `get-instance-access` to request access credentials and other information. If successful, Amazon GameLift 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:

   ```bash
   # aws gamelift get-instance-access
   --fleet-id "fleet-a7abc071-5537-4f0f-b5ee-1b5c1187565f"
   --instance-id "i-01463992e435d836c"
   ```

   **For Linux instances** – To connect to a Linux instance, SSH requires a user name and private key. Amazon GameLift 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, you must (1) convert the string to a .pem file, and (2) set permissions for the new file.

   - To convert the string to a properly formatted .pem file, add special parameters to your `get-instance-access` request, as shown in the following example. This example automatically outputs the returned value of Secret to a text file named MyPrivateKey.pem, and replaces all the \n characters with line breaks.

     ```bash
     # aws gamelift get-instance-access
     --fleet-id "fleet-a7abc071-5537-4f0f-b5ee-1b5c1187565f"
     --instance-id "i-01463992e435d836c"
     --query "InstanceAccess.Credentials.Secret"
     --output text > MyPrivateKey.pem
     ```

   - To set permissions on the new file, run the following command

     ```bash
     # chmod 400 MyPrivateKey.pem
     ```

3. **Open a port for the remote connection.** Instances in Amazon GameLift fleets can only be accessed through ports 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. 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:

   ```bash
   # aws gamelift update-fleet-port-settings
   --fleet-id "fleet-a7abc071-5537-4f0f-b5ee-1b5c1187565f"
   --inbound-permission-authorizations
   "FromPort=22,ToPort=22,IpRange=54.186.139.221/32,Protocol=TCP"
   ```
4. **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.

## View and Update Remote Instances

When connected to an instance remotely, you have full user and administrative access. This means you also have the ability to cause errors and failures in game hosting. If the instance is hosting games with active players, you run the risk of crashing game sessions and dropping players, as well as disrupting game shutdown processes and causing errors in saved game data and logs.

Hosting resources on an instance can be found in the following locations:

- **Game build files.** These are the files included in the game build you uploaded to Amazon GameLift. They include one or more game server executables, assets and dependencies. These files are located in a root directory called `game`:
  - On Windows: `c:\game`
  - On Linux: `/local/game`

- **Game log files.** Any log files your game server generates are stored in the `game` root directory at whatever directory path you designated.

- **Amazon GameLift hosting resources.** Files used by the Amazon GameLift service to manage game hosting are located in a root directory called `Whitewater`. These files should not be changed for any reason.

- **Runtime configuration.** The fleet runtime configuration is not accessible for individual instances. To test changes to a runtime configuration (launch path, launch parameters, maximum number of concurrent processes), you must update the fleet-wide runtime configuration (see the AWS SDK action `UpdateRuntimeConfiguration` or the AWS CLI `update-runtime-configuration`).

## Set Up VPC Peering

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. This topic provides guidance on setting up a peering connection between the VPC for your Amazon GameLift game servers and a VPC for non-Amazon GameLift resources. For more information on how VPC peering works with Amazon GameLift, see [Networking With AWS Resources](#).

If you’re already familiar with Amazon VPCs and VPC peering, please note that setting up peering with Amazon GameLift game servers is somewhat different. Since you don’t have access to the VPC for your game server processes (it is controlled by the Amazon GameLift service), you can’t create a VPC peering connection request for it. Instead, you first pre-authorize a VPC with non-Amazon GameLift resources to accept a peering request from Amazon GameLift. You then have Amazon GameLift request the VPC peering that you just authorized. Amazon GameLift automatically creates the peering connection, sets up the route tables, and configures the connection as needed.

**To set up a VPC peering connection:**

1. **Get identifiers for each VPC.**

   Get the following information for the two VPCs to be peered:

   - VPC for your Amazon GameLift game servers – Your game server processes are deployed in Amazon GameLift as a fleet of EC2 instances. Each fleet is automatically placed in its own VPC, which is managed by the Amazon GameLift service. Since you don’t have access to the VPC for
your game server processes, you identify a VPC by it’s associated fleet. To identify the set of game server processes you want to establish a VPC peering for, you need the Amazon GameLift fleet ID.

- VPC for your non-Amazon GameLift 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 for help creating a VPC and adding resources to it. 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 requesting a peering, both VPCs must exist in the same region. The VPC for your Amazon GameLift fleet game server processes is in the same region as the fleet.

2. **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 is used to manage your Amazon GameLift game servers.
- AWS account that is used to manage your non-Amazon GameLift resources.

If you’re using the same account for Amazon GameLift and non-Amazon GameLift resources, you’ll need the ID and credentials for only one account.

3. **Authorize a VPC peering with non-Amazon GameLift resources.**

In this step, you are accepting a future request from Amazon GameLift to peer to your VPC for non-Amazon GameLift resources. For this action, use credentials for the account that manages your non-Amazon GameLift resources.

To authorize the VPC peering, call the Amazon GameLift service API `CreateVpcPeeringAuthorization()` or use the AWS CLI command `create-vpc-peering-authorization`. Identify the following information:

- Peer VPC ID – This is for the VPC with the non-Amazon GameLift resources.
- Amazon GameLift AWS account ID – This is for the account that you use to manage your Amazon GameLift 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`).

4. **Request a peering between VPCs for an Amazon GameLift fleet and the non-Amazon GameLift resources.**

Once you have a valid authorization for a VPC peering, you can trigger Amazon GameLift to request the peering. For this action, use credentials for the account that manages your Amazon GameLift game servers.

To request a VPC peering, call the Amazon GameLift service API `CreateVpcPeeringConnection()` or use the AWS CLI command `create-vpc-peering-connection`. Identify the following information, which identifies the two VPCs to peer:

- Peer VPC ID and AWS account ID – This is the VPC for your non-Amazon GameLift resources and the account that you use to manage them. The VPC ID used must match one on a valid authorization.
• Fleet ID – This identifies the VPC for your Amazon GameLift game server processes.

You can manage your VPC peering connections using the following operations:

- DescribeVpcPeeringConnections() (AWS CLI describe-vpc-peering-connections).
- DeleteVpcPeeringConnection() (AWS CLI delete-vpc-peering-connection).

You can request a VPC peering connection when you create a new fleet. For more information, see Create a VPC Peering Connection with a New Fleet (p. 57).

5. Track your VPC peering connections.

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.

Common reasons a connection request fails:

- An authorization for the requested connection was not found. This may mean an existing authorization is no longer valid, or never existed. A common cause for this issue is a region mix-up. Verify that your authorization and your request are using the same region.
- Overlapping CIDR blocks (see Invalid VPC Peering Connection Configurations). The IPv4 CIDR blocks that are assigned to peered VPCs cannot overlap. The CIDR block of the VPC for your Amazon GameLift fleet is automatically assigned and can't be changed. You can look up this CIDR block by calling DescribeVpcPeeringConnections(). To resolve this issue, you'll need to change the CIDR block of the VPC for your non-Amazon GameLift resources to a non-overlapping range.
- The fleet did not activate. If you requested a VPC peering connection as part of a CreateFleet() request, the new fleet may have failed to progress to Active status. In this scenario, the peering connection cannot succeed.
Working with Aliases

Amazon GameLift uses the term alias to refer to the redirecting of player sessions to a fleet that you specify. You can use aliases to direct player sessions to fleets that you’re testing for launch or to support multiple game client sessions.

Amazon GameLift uses the term alias to refer to the redirect connection between a fleet and players. You can decide which fleet you want an alias to resolve to, and you can embed the alias ID into your game client to control the flow of players to the designated fleet.

There are two types of routing strategies for aliases:

- **Simple** – A simple alias routes player traffic to the associated fleet. You can update the fleet to which the alias resolves at any time.
- **Terminal** – A terminal alias does not resolve to a fleet. Instead, it passes a message back to the client. For example, you may want to notify players to upgrade their game client versions if they are attempting to connect to a fleet that is no longer supported.

Create an Alias

You can create a new alias to resolve to a fleet.

**To create a new alias**

1. Open the Amazon GameLift console at https://console.aws.amazon.com/gamelift/.
2. Choose Aliases from the menu bar.
3. On the Aliases page, click Create alias.
4. On the Create alias page, in the Alias details section, do the following:
   - **Alias name** – Type a friendly name so you can easily identify the alias in the catalog.
   - **Description** – (Optional) Type a short description for your alias to add further identification.
5. In the Routing options section, for Type, choose Simple or Terminal:
   - If you choose Simple, select an available fleet to associate with your alias. A simple alias routes player traffic to the associated fleet.
   - If you select Terminal, type a message that will be displayed to players. A terminal alias does not resolve to a fleet but only passes your message to the client.
6. Click Configure alias.

Edit an Alias

Use the Edit alias page in the Amazon GameLift console to update the information for your alias.

**To edit an alias**

1. Open the Amazon GameLift console at https://console.aws.amazon.com/gamelift/.
2. Choose Aliases from the menu bar.
3. On the Aliases page, click the name of the alias you want to edit.
4. On the selected alias page, for **Actions**, choose **Edit alias**.

5. On the **Edit alias** page, you can edit the following:
   - **Alias name** – Friendly name for your alias.
   - **Description** – Short description for your alias.
   - **Type** – Routing strategy for player traffic. Select **Simple** to change the associated fleet or select **Terminal** to edit the termination message.

6. Click **Submit**.
Working with Queues

Amazon GameLift uses the queues feature to efficiently place new game sessions on any fleet in a group of fleets that can span multiple regions. When a game client or game service requests a new game session, it specifies a queue to process the request. You can have multiple queues per account. Use the AWS Command Line Interface (CLI) or the Amazon GameLift console to create, edit, or track metrics for a queue.

Topics
- Design a Game Session Queue (p. 79)
- Create a Queue (p. 83)
- View Your Queues (p. 86)
- View Queue Details (p. 86)

Design a Game Session Queue

Amazon GameLift uses queues to process requests for new game session placements and to find the "best" available hosting resources. The way you design your queue determines (1) where Amazon GameLift looks for available resources, and (2) how Amazon GameLift evaluates available resources to find the "best" resource to host a requested new game session. Use queues to deliver the best possible experience to your players and to ensure that the hosting resources you're paying for are utilized efficiently.

Queues are required with these Amazon GameLift features:
- Matchmaking with FlexMatch (see Design a FlexMatch Matchmaker (p. 89))
- Spot fleets (see Spot Fleet Integration Guide (p. 69))

Learn more about queues in the section called “Placing Game Sessions” (p. 3). For information on creating a queue, see Create a Queue (p. 83). For information on creating new game sessions with queues, see Create Game Sessions (p. 35).

Why Use Queues?

We highly recommend using queues for game session placement even if you're not using an Amazon GameLift feature that requires them. While you always have the option to manually create a game session on a specific fleet (CreateGameSession), queues—particularly those that span multiple regions—can offer critical benefits for you and your players.

- Minimize latency for a better player experience. When a game session placement request includes player latency data, Amazon GameLift's FleetIQ ensures that players get the lowest possible latency to the game server. Set additional rules to minimize latency differences between players and to strike a balance between getting players into games fast and getting players into the best possible gaming experience.
- Take advantage of lower-priced spot fleets. You can add spot fleets, which offer significantly lower hosting costs, to a queue at any time. When spot fleets are available, FleetIQ places new game sessions in fleets with the lowest possible latency and the lowest spot price.
- Place new games faster at high capacity. A single fleet has limited capacity, and once that limit is reached, players must wait until additional instances are scaled up before new game sessions can
start. A queue, however, can fall back to immediately make placements on other fleets if the preferred fleet is full. In addition, with auto-scaling, each fleet in the queue scales up as it nears full capacity, so it's unlikely that all fleets in a queue will be full at the same time. As a result, wait time for players is impacted less during surges in player demand.

- **Make game availability more resilient.** Fleet and region-level outages can happen. When using a multi-region queue, a slowdown or outage doesn't have to affect player access to your game. Instead, if one or more preferred fleets are unavailable, Amazon GameLift can place new game sessions with the next best fleet. Auto-scaling then adjusts for this temporary shift in fleet activity until the preferred fleet(s) are available again.

- **Use extra fleet capacity more efficiently.** To handle unexpected surges in player demand, it makes sense to have quick access to extra hosting capacity. When relying on a single fleet to support player access to your game, you need to maintain unused capacity just in case. In contrast, the fleets in a queue can act as fall backs, with increased player demand shifting to lesser used fleets in other regions as needed. For example, when demand is high in Asia, demand is typically low in Europe; your European fleets can provide additional capacity to support surges in Asia, even when they're scaled down during low demand.

- **Get metrics on game session placements and queue performance.** Amazon GameLift emits queue-specific 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 console or in CloudWatch.

**Tips on Setting Queue Destinations**

A queue contains a list of destinations where game session placement requests can be fulfilled. In most cases a destination is a fleet, which you can specify by either a fleet ID or alias ID. When choosing destinations for the queue, consider the following guidelines and best practices:

- You can add any existing fleet or alias from any region. Add a fleet in each region where you want to support players, especially if you're using player latency for placement.
- If you assign aliases to your fleets (as recommended), it is a good idea to use the alias names when setting destinations in your queue.
- All destinations in a queue must be running game builds that are compatible with the game clients that use the queue. Keep in mind that new game session requests processed by the queue might be placed on any destination in the queue.
- A queue should have at least two fleets and span at least two regions. This design improves hosting resiliency by decreasing the impact of fleet or regional slowdowns and more efficiently managing unexpected changes in player demand.
- The list order of destinations in a queue matters. If you include latency data in a game session placement request, Amazon GameLift re-prioritizes the destinations to find available resources with (1) minimal player latency and (2) lowest spot price (if relevant). If you don't provide latency data, Amazon GameLift follows the destination list order. In this situation, game sessions will usually be hosted on the first fleet listed and will only be placed on back-up fleets when needed.
- You need to decide where to create your queue (in which region). Ideally you're making game session placement requests through a game client service, such as a session directory service. If so, we recommend that you create your queue in a region that is geographically near where the client service is deployed. This arrangement minimizes latency when submitting game session placement requests.

**Design a Multi-Region Queue**

A multi-region design is recommended for all queues, whether or not you're using special features such as FlexMatch matchmaking or spot fleets. This design can improve placement speed and hosting resiliency, and is critical when working with player latency data.
This topic steps through the process of designing a basic multi-region queue. For this example, we'll use a game server build that we want to distribute to players on the east coast of North America. We've chosen to host the our game in the following regions: us-east-1 and ca-central-1. We've uploaded the game build and created a fleet in each region.

1. Pick a region to create the queue in. Our queue's location may not be significant, but if we're making placement requests through a client service, we can minimize request latency by placing the queue in a region near where the client service is deployed.

2. Create a new queue and add our fleets as queue destinations.

3. Decide on a default list order for the destinations. GameLift may evaluate fleets in this order when searching for available hosting resources. If a placement request contains player latency data, Amazon GameLift reorders our destinations to prioritize the lowest latency rates. If no latency data is provided, Amazon GameLift uses the default order.

   • If we expect a lot of requests without latency data, we'll want to pay more attention to our default list order and the fleets we list first. Without latency data, Amazon GameLift will always place new game sessions on the first fleet listed when available, only using the remaining fleets as backup. As a result, we'll want a fleet in the top spot that (1) is high-powered enough to handle most of our player base, and (2) is likely to deliver the highest availability and lowest latency to the majority of our players.

      In this scenario, we expect that the us-east-1 fleet will best serve most of our players, so we choose a high-performance instance type, configure it for multiple concurrent game sessions, scale for high player volume, and list it first in our queue. Our backup ca-central-1 fleet uses a smaller, less expensive instance type and maintains minimal capacity. Additional backup fleets (if added) do not need to be ordered, but we might de-prioritize regions that are likely to have high latency rates.

   • If we expect most requests to have latency data, the list order is less important because FleetIQ will re-prioritize it. We'll want to have fleets in all regions that are close to our target players to minimize latency rate. Generally, we'll want fleets that are configured to handle the expected amount of player demand. We'll want to make sure that the fleet in the top spot is a good choice to handle the requests that don't have latency data.

      In this scenario, we expect that both the us-east-1 and ca-central-1 fleets will provide low latency rates and be heavily used. Both fleets use high-performance instance types and are configured for multiple concurrent game sessions. Demand for the ca-central-1 fleet will be somewhat lower, so it will maintain a lower capacity.

As our queue is put into action, we can use metric data to determine how well the design is holding up. With queues, we make fleet changes as needed, either by reconfiguring existing fleets or by removing and adding new fleets that better fit our hosting needs.

**Design Player Latency Policies**

If your game session placement requests include player latency data, Amazon GameLift uses FleetIQ to ensure that players are put into game sessions with the lowest possible latency. FleetIQ prioritizes a queue's destinations based on the average regional latency for all players in the request.

You can set up player latency policies to affect how Amazon GameLift uses player latency data. You can set player latency policies to:

   • Set maximum latency for individual players. Amazon GameLift by default places game sessions based on average latency across all players in the request. This policy ensures that a game session is not placed where any one player will experience latency over the maximum.

   • Use multiple policies to relax maximum latency over time. Setting a maximum latency protects players from high-latency games, but they also increase the risk that a game session request with high latency values will never be fulfilled. You can use a series of policies to gradually raise the latency maximum
over time. Use this approach to balance how you provide great game experiences against getting players into games with a minimum of wait time.

For example, you might define the following policies for a queue with a 5-minute timeout. Multiple policies are enforced consecutively, starting with the policy that has the lowest maximum latency value. This set of policies starts with a maximum latency of 50 milliseconds and increases it over time to 200 milliseconds. Your last policy sets the absolute maximum latency allowed for any player. If you want to ensure that all game sessions are placed regardless of latency, you can set last policy's maximum to a very high value.

1. Spend 120 seconds searching for a destination where all player latencies are less than 50 milliseconds, then...
2. Spend 120 seconds searching for a destination where all player latencies are less than 100 milliseconds, then...
3. Spend the remaining queue timeout searching for a destination where all player latencies are less than 200 milliseconds.

In this example, the first policy is in force for the first two minutes, the second policy is in force for the third and fourth minute, and the third policy is in force for the fifth minute until the placement request times out.

Design a Queue for Spot Instances

If you plan to use spot instances for your fleets, you'll need to set up a queue. Ideally, you want to set up a resilient queue that takes advantage of cost savings with spot fleets while minimizing the chance of game session interruptions. Use the following best practices:

- Include fleets from more than one region. A multi-region queue boosts resiliency by ensuring that there are always fleets available to host new game sessions.
- In each region, include at least one spot fleet and one on-demand fleet. This design ensures that game sessions can be placed in any region even if no spot fleets are currently viable in that region.
- If you include multiple spot fleets in each region, use different instance types for each fleet, preferably in the same instance family (C4.large, C4.xlarge, etc.). This design lessens the likelihood that multiple spot fleets will be unavailable or interrupted at the same time. Check the historical pricing data in the Amazon GameLift console to verify that your preferred instance types typically deliver significant cost savings with spot. This data displays pricing for spot and on-demand instances and provides estimated spot saving per instance.
- To optimize FleetIQ’s ability to select the lowest priced spot instance, plan to include player latency data for all regions. As described in How Perfect Placement Works (p. 82), FleetIQ works more effectively when destinations are re-prioritized by region, which is done when latency data is provided.
- If you're not providing player latency data in game session placement requests, order your destinations by preference. For example, you might list destinations by region preference (spot fleet followed by on-demand fleet). Alternatively, you might list all your spot fleets first.

How Perfect Placement Works

Amazon GameLift’s FleetIQ uses the following decisionmaking process when searching for the best possible placement for a new game session.

1. FleetIQ filters the queue's destinations to remove any of the following fleets:
   - If the request includes player latency data, FleetIQ evaluates the queue's player latency policies and remove any fleets in regions where any player's latency exceeds a policy's maximum limit.
Create a Queue

Queues are used to place new game sessions with the best available hosting resources across multiple fleets and regions. To learn more about building queues for your game, see Design a Game Session Queue (p. 79).

In a game client, new game sessions are started with queues by using placement requests. Learn more about game session placement in Create Game Sessions (p. 35).

To create a queue, you can use either the Amazon GameLift console or the AWS Command Line Interface (p. 18) (CLI).

Create a Queue (Console)

Create a queue from the AWS Management Console.

To create a queue

1. Open the Amazon GameLift console at https://console.aws.amazon.com/gamelift/, and choose the region you want to create your queue in.
2. From the Amazon GameLift menu, choose Create a queue.
3. On the Create queue page, complete the Queue Details section:
   - **Queue Name** – Create a meaningful queue name so you can identify it in a list and in metrics. Requests for new a game session (using StartGameSessionPlacement) must specify a queue by this name. Spaces and special characters are not allowed.
   - **Queue Timeout** – Specify how long you want Amazon GameLift to try to place a new game session before stopping. Amazon GameLift continues to search for available resources on any fleet until the request times out.
4. Under Player latency policies, define zero or more policies for the queue. For each placement request, Amazon GameLift automatically minimizes the average latency across all players. You can also create a latency policies to set a maximum limit for each individual player. Player latency policies are evaluated only when player latency data is provided in the placement request. You can opt to enforce one limit throughout the placement process, or you can gradually relax the limit over time. Learn more at Design Player Latency Policies (p. 81).
a. To add a first policy, choose Add player latency policy. Enter a maximum player latency value for this policy (default is 150 milliseconds). As indicated in the policy language, this first policy will be enforced either for the entire placement process or—if you create additional policies—for any remaining time after the other policies have expired.

b. To add another player latency policy, choose Add player latency policy again. For additional policies, set the maximum player latency value and a length of time (in seconds) to enforce it. Maximum latency values for these policies must be lower than the first policy.

As you add policies, the console automatically reorders the policies based on the maximum player latency value, lowest value listed first. This is the order in which the policies are enforced during a game session placement effort.

5. Under Destinations, add one or more destinations to the queue. A queue can contain fleets from multiple regions and with both on-demand and spot fleets. The only requirement is that all fleets must be running game builds that are compatible with the game clients that will use the queue, such as fleets in multiple regions that are running the same game build. Fleets and aliases must exist before you can add them as destinations.

a. Choose Add destination.

b. Use the columns to specify the region and type (fleet or alias) for your destination. From the resulting list of fleet or alias names, select the one you want to add.

c. To save the destination, choose the green checkmark icon. You must save each destination before adding another one, changing the default order, or saving the queue.

d. If you have multiple destinations, set the default order by using the arrow icons in the Priority (default) column. This order is used by Amazon GameLift when searching destinations for available resources to place a new game session. (The default order is overridden if a game session placement request includes player latency data.)

6. Once you've finished configuring your new queue, choose Create queue. Your new queue is saved and the Queues page shows the new queue and any other queues that exist. You can choose a queue from this page to view detailed information, including queue metrics. You can edit a queue configuration at any time.

Create a Queue (AWS CLI)

You can use the AWS Command Line Interface (AWS CLI) to create a queue. If you haven't yet installed the AWS CLI, see the topic Install the AWS CLI (p. 18).

To create a queue

- Open a command line window and use the create-game-session-queue command to define a new queue. For more information, see the AWS CLI Command Reference.

The following example creates a queue for placing new game sessions on any of several fleets before timing out at five minutes. Fleets are listed as destinations and identified by either a fleet ARN or alias ARN. All fleets and aliases must already exist. Amazon GameLift tries to place new game sessions on fleets in the order they are listed here unless the order is overridden by individual game session placement requests.

Note
You can get fleet and alias ARN values by calling either describe-fleet-attributes or describe-alias with the fleet or alias ID. For more information on ARN (Amazon Resource Name) formats, see ARNs and AWS Service Namespaces.

```
$ aws gamelift create-game-session-queue
--name "Sample test queue"
```
Create a Queue (AWS CLI)

```
--timeout-in-seconds 300
--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
```

Copiable version:

```
aws gamelift create-game-session-queue --name "Sample test queue" --timeout-in-seconds 300 --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
```

If the `create-game-session-queue` request is successful, Amazon GameLift returns a `GameSessionQueue` object with the new queue configuration. You can now submit requests to the queue using `StartGameSessionPlacement`.

To create a queue with player latency policies

- Open a command line window and use the `create-game-session-queue` command to define a new queue. For more information, see the AWS CLI Command Reference.

The following example creates a queue with a 10-minute timeout, three destinations, and a set of player latency policies. In this example, the first player latency policy is in force for the first two minutes, the second policy is in force for the third and fourth minute, and the third policy is in force for six minutes until the placement request times out.

```
$ 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"
```

Copiable version:

```
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 returns a `GameSessionQueue` object with the new queue configuration.
View Your Queues

You can view information on all existing game session placement queues (p. 79). Queues shown include only those created in the selected region. From the Queues page, you can create a new queue, delete existing queues, or open a details page for a selected queue. A Queue detail page contains the queue's configuration and metrics data. You can also edit or delete the queue.

To view the Queues page

1. Choose Queues from the Amazon GameLift console's menu.

   The Queues page displays the following summary information for each queue:

   - **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. New game sessions can be placed on any fleet in the queue.

2. To view details for a queue, including metrics, choose the queue name. For more information on the queue details page, see View Queue Details (p. 86).

View Queue Details

You can access detailed information on any queue, including the queue configuration and metrics. To open a Queue detail page, go to the main 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 Actions, Edit queue.
- Remove a queue. After a queue is removed, all requests for new game sessions that reference that queue name will fail. (Note that deleted queues can be restored by simply creating a queue with the deleted queue's name.) Choose Actions, Delete queue.

Summary

The summary table includes the following information:

- **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. New game sessions can be placed on any fleet in the queue.

Destinations

The Destinations tab shows all fleets or aliases listed for the queue. Fleets are identified by either a fleet ARN or an alias ARN, which specifies the fleet or alias ID and the region.
When Amazon GameLift searches the destinations for available resources to host a new game session, it searches in the order listed here. As long as there is capacity on the first destination listed, new game sessions are placed there. This is the default order for destinations. You can have individual game session placement requests override the default order by providing player latency data. This data tells Amazon GameLift to search for an available destination with the lowest average player latency.

To add, edit, or delete destinations, choose Actions, Edit queue.

Player Latency Policies

The Player latency policies tab shows all policies that have been defined for the queue. Policies are listed in the order they are enforced during a game session placement effort.

To add, edit, or delete player latency policies, choose Actions, Edit queue.

Queue Metrics

The Metrics tab shows a graphical representation of queue metrics over time.

To display metrics information in the graph

1. Click one or more of the metric names that are listed to the left of the graph area. Metric names shown in black are displayed in the graph, while those shown in gray are turned off. Use the color key to identify which graphed line matches a selected metric. The following metrics are available:

   • Queue – These metrics show game session placement activity across the entire queue.
   • Average wait time – Average length of time that the requests currently in the queue have been waiting for placement.
   • Queue depth – Total number of requests that are currently in the queue.
   • Placements started – Number of new requests that were added to the queue.
   • Placements succeeded – Number of requests that resulted in successfully starting a new game session.
   • Placements cancelled – Number of requests that were cancelled before succeeding.
   • Placements time out – Number of requests that waited for the full timeout period without resulting in a new game session.
   • Placements failed – Number of requests that failed for other reasons.

   • Fleet-specific (listed by ARN) – These metrics show game session placement activity broken down by hosting location. Each placement request in a queue is assigned a "preferred" fleet, which is based on either the default queue order (the first available fleet listed) or on the request's player latency data. These metrics show how successfully the queue is handling the task of placing new game sessions on the best available fleet.
   • Queue depth – Number of game session placement requests that are waiting to be hosted on this fleet.
   • Average wait time – Average length of time that a game session request is currently spending in the queue.

2. Use the following filters, shown above the graph area, to change how metric data is displayed:

   • Data and Period – Offers two options for selecting a date range:
     • Use Relative to select a period of time relative to the current time, such as Last hour, Last day, Last week.
     • Use Absolute to specify a period with an arbitrary start and end date/time.
   • Granularity – Select a length of time to aggregate data points.
• **Refresh rate** – Select how often you want the graph display to be updated.
• **Format** – Select which time format to use in the graph display: **UTC** (universal coordinated time) or **Browser time** (local time).
• **Show points** – Toggle to display discrete data points as circles or display lines only.
Matchmaking with FlexMatch

Use Amazon GameLift's FlexMatch feature to add player matchmaking to your games. FlexMatch pairs the matchmaking service with a customizable rules engine. This lets you design how to match players together based on player attributes and game modes that make sense for your game, and rely on FlexMatch to manage the nuts and bolts of forming player groups and placing them into games.

In addition, FlexMatch builds on the Queues feature. Once a match is created, FlexMatch hands the match details to a queue of your choice. The queue then searches for available hosting resources on your Amazon GameLift fleets and starts a new game session for the match.

For more information on FlexMatch's key features and how it works, see Amazon GameLift FlexMatch (p. 10).

Topics

- FlexMatch Integration Guide (p. 89)
- Design a FlexMatch Matchmaker (p. 90)
- Build a FlexMatch Rule Set (p. 93)
- Set up FlexMatch Event Notification (p. 105)
- Add FlexMatch to a Game Client (p. 106)
- Add FlexMatch to a Game Server (p. 110)
- Backfill Existing Games with FlexMatch (p. 112)

FlexMatch Integration Guide

To add FlexMatch matchmaking to your game, you need to complete the following tasks.

- **Set up a matchmaker.** A matchmaker receives matchmaking requests from players and processes them. It groups players based on a set of defined rules and, for each successful match, creates a new game sessions and player sessions. Follow these steps to set up a matchmaker:
  - **Create a rule set.** A rule set tells the matchmaker how to construct a valid match. It specifies team structure and specifies how to evaluate players for inclusion in a match. See these topics:
    - Build a FlexMatch Rule Set (p. 93)
    - FlexMatch Rule Set Examples (p. 95)
  - **Create a game session queue.** A queue locates the best region for each match and creates a new game session in that region. You can use an existing queue or create a new one for matchmaking. See this topic:
    - Create a Queue (p. 83)
  - **Set up notifications (optional).** Since matchmaking requests are asynchronous, you need a way to track the status of requests. Notifications is the preferred option. See this topic:
    - Set up FlexMatch Event Notification (p. 105)
  - **Configure a matchmaker.** Once you have a rule set, queue, and notifications target, create the configuration for your matchmaker. See these topics:
    - Design a FlexMatch Matchmaker (p. 90)
    - Create a Matchmaking Configuration (p. 91)
  - **Integrate FlexMatch into your game client service.** Initiate new game sessions with matchmaking. Add functionality to your game client service to make matchmaking requests and provide player attributes for matching. Matchmaking requests identify the matchmaker to use, which determines the rule set and queue that are used to fulfill requests. See this topic:
    - Add FlexMatch to a Game Client (p. 106)
• **Integrate FlexMatch into your game server.** Enable your game server to start new game sessions for matches in response to requests from the Amazon GameLift service. Requests include game, player, and match information, which your game server uses to construct a game session for the match. See this topic:
  • Add FlexMatch to a Game Server (p. 110)
• **Set up FlexMatch backfill (optional).** Find player matches to fill open player slots in existing games. You can initiate match backfill requests directly from your game server processes, or from a client-side service, such as a session directory service. See this topic:
  • Backfill Existing Games with FlexMatch (p. 112)

Design a FlexMatch Matchmaker

A FlexMatch matchmaker process does the work of building a game match. It manages the pool of matchmaking requests received, forms teams for a match, processes and selects players to find the best possible player groups, and initiates the process of placing and starting a game session for the match. This topic describes the key aspects of a matchmaker and how to configure one customized for your game.

To create a matchmaker, define a matchmaking configuration using either the Amazon GameLift console or the AWS CLI (see Create a Matchmaking Configuration (p. 91)). You can configure as many matchmakers as you need for your game.

For a detailed description of how a FlexMatch matchmaker processes the matchmaking requests it receives, see How Amazon GameLift FlexMatch Works (p. 11).

**Matchmaker Configurations**

At a minimum, a matchmaker needs three elements:

• The **rule set** determines the size and scope of teams for a match and defines a set of rules to use when evaluating players for a match. Each matchmaker is configured to use one rule set. For more information on creating rule sets, see Build a FlexMatch Rule Set (p. 93).

• The **game session queue** determines where matches created by this matchmaker will be hosted. The matchmaker uses the queue to find available resources and to place a game session for the match. A queue specifies which regions to use when placing a game session. For more information on creating queues, see Create a Queue (p. 83).

• The **request timeout** determines how long matchmaking requests can remain in the request pool and be evaluated for potential matches. Once a request has timed out, it has failed to make a match and is removed from the pool.

In addition to these minimum requirements, you can configure your matchmaker with the following additional options.

**Player Acceptance**

You can configure a matchmaker to require that all players selected for a match must accept participation. If acceptance is required, all players must be given the option to accept or reject a proposed match. A match must receive acceptances from all players in the proposed match before it can be completed. If any player rejects or fails to accept a match, the proposed match is discarded. The matchmaking requests are handled as follows: requests where all players accepted the match are returned to the matchmaking pool for continued processing; requests where at least one player rejected the match or failed to respond fail and are no longer processed. Player acceptance requires a time limit, which you can define; all players must accept a proposed match within the time limit for the match to continue.
Matchmaking Notification

This feature emits all matchmaking-related events to a designated Amazon Simple Notification Service (SNS) topic. Since matchmaking requests are asynchronous, all games must have a way to track the status of matchmaking requests, and setting up notifications is simple and efficient. See more on options for tracking requests in Add FlexMatch to a Game Client (p. 106). To use notifications, you first need to set up an SNS topic, and then provide the topic ARN as the notification target in the matchmaking configuration. See more information on setting up notifications at Set up FlexMatch Event Notification (p. 105).

Game Properties

When requesting a new game session, you can pass additional information to your game server about the type of game session to create. Game properties can be delivered with any type of game session request—with a placement request or with a matchmaking request. With matchmaking, game properties are included in the matchmaker configuration. All game sessions created using a matchmaker use the same game properties. If you need to vary game properties, create separate matchmakers and send matchmaking requests to the matchmaker with the appropriate game properties.

Reserved Player Slots

You can designate that certain player slots in each match be reserved and filled at a later time. This is done by configuring the “additional player count” property of a matchmaking configuration.

Custom Event Data

Use this property to include a set of custom information in all matchmaking-related events for the matchmaker. This feature can be useful for tracking certain activity unique to your game, including tracking performance of your matchmakers.

Create a Matchmaking Configuration

To set up a FlexMatch matchmaker, you create a new matchmaking configuration. Use either the Amazon GameLift console or the AWS Command Line Interface (AWS CLI). For more detailed information on configuring a matchmaker, see the section called “Design a Matchmaker” (p. 90).

Note

Before you can create a matchmaking configuration, you must have already created the rule set and queue that you want to use with the matchmaker.

Choosing a Region for the Matchmaker

A matchmaker is hosted in the region where its configuration is created. It's worth considering how the location of a matchmaker might affect its performance and how to refine the match experience for its intended players. We recommend that you place a matchmaker in a region close to the clients or client service that will be requesting matchmaking. As a best practice, we also recommend putting the matchmaker and the queue it uses in the same region. This helps to minimize communication latency between the matchmaker and queue.

Create a Matchmaking Configuration (Console)

Follow these steps to create a matchmaking configuration from the AWS Management Console.

To create a matchmaking configuration with the Amazon GameLift console:

1. Open the Amazon GameLift console at https://console.aws.amazon.com/gamelift/home.
2. Switch to the region where you want to place your matchmaker.
3. From the main menu, choose Create matchmaking configuration. Fill in the matchmaking configuration details.
• **Name** – Create a meaningful matchmaker name so you can easily identify it in a list and in metrics. The matchmaker name must be unique within a region. Matchmaking requests identify which matchmaker to use by its name and region.

• **Description** – (Optional) Add a description of the matchmaker. The description is used for identification only; it is not used in the matchmaking process.

• **Queue** – Choose the game session queue to use with this matchmaker. To find the queue, first choose the region where the queue is configured. Then choose the queue you want from the list of available queues in that region.

• **Request timeout** – Type the maximum amount of time, in seconds, for the matchmaker to complete a match for each request. Matchmaking requests that exceed this time are terminated.

• **Acceptance required** – (Optional) Indicate whether to require each player in a proposed match to actively accept participation in the match. If you chose yes, indicate how long you want the matchmaker to wait for player acceptances before canceling the match.

• **Rule set name** – Choose the rule set to use with this matchmaker. The list contains all rule sets that have been created in the current region.

• **Notification target** – (Optional) Type the ARN of an SNS topic for receiving matchmaking event notifications. If you haven't set one up yet, you can add this information later by editing the matchmaking configuration. See [Set up FlexMatch Event Notification (p. 105)](#).

• **Additional players** – (Optional) Specify the number of player slots to remain unfilled in each new match. These slots can be filled with players in the future.

• **Custom event data** – (Optional) Specify any data you want to associate with this matchmaker in event messaging. This data is included in every event that is associated with the matchmaker.

4. Once you've finished configuring a matchmaker, click **Create**. If the creation is successful, the matchmaker is immediately ready to accept matching requests.

## Create a Matchmaking Configuration (AWS CLI)

To create a matchmaking configuration with the AWS CLI, open a command line window and use the `create-matchmaking-configuration` command to define a new matchmaker. See complete documentation on this command in the [AWS CLI Command Reference](#). If you haven't yet installed the AWS CLI, see [Install the AWS CLI (p. 18)](#).

This example creates a new matchmaking configuration that requires player acceptance and uses notifications to track the status of matchmaking requests. It also reserves two player slots for additional players to be added later.

```bash
$ aws gamelift create-matchmaking-configuration
  --name "SampleMatchmaker123"
  --description "The sample test matchmaker with acceptance"
  --rule-set-name "My_Rule_Set_One"
  --request-timeout-seconds "120"
  --acceptance-required "true"
  --acceptance-timeout-seconds "30"
  --additional-player-count "2"
  --game-session-data "key=map,value=winter444"
```

Copiable version:

```bash
copyableversion:
```

Version
92
If the matchmaking configuration creation request is successful, Amazon GameLift returns a `MatchmakingConfiguration` object with the settings that you requested for the matchmaker. The new matchmaker is immediately ready to accept matchmaking requests.

### Build a FlexMatch Rule Set

Every FlexMatch matchmaker must have a rule set. The rule set determines the two key elements of a match: (1) team structure and size, and (2) how to group players for the best possible match and gameplay experience.

For example, a rule set might describe a match like this: Create a match with two teams of five players each, one team is the defenders and the other team the invaders. All players in both teams must be equally (or nearly) skilled. All players must be able to play with a latency of 20 milliseconds or less. If no match is found after 30 seconds, gradually relax the skill and latency requirements.

To create a matchmaking rule set, you can use either the Amazon GameLift console or the AWS CLI. You can create multiple rule sets and use them in different matchmakers, or use the same rule set in more than one matchmaker.

#### Topics

- Create a Matchmaking Rule Set (p. 94)
- FlexMatch Rule Set Examples (p. 95)
- FlexMatch Rules Reference (p. 172)

### Rule Set Components

Each rule set contains the following components:

- **Name** (optional) – This is a descriptive label within the rule set syntax and is not used by Amazon GameLift in any meaningful way. Do not confuse this value with the rule set name, which is set, along with the rule set syntax, when you create a rule set.

- **Rule language version** (required) – This is the version of the property expression language used to create FlexMatch rules. The value must be equal to "1.0".

- **Player attributes** (optional) – Rules may choose players for matches based on individual player characteristics. If you create rules that rely on player attributes, they must be declared in this section. Values for these player attribute must be included in every matchmaking request that is sent a matchmaker using this rule set. When declaring a player attribute, include the following information:
  - **Name** (required) – This value must be unique to the rule set.
  - **Type** (required) – This is the data type of the attribute value. Valid data types are number, string, or string map.
  - **Default** (optional) – Enter a default value to use if none is provided for a player. If no default is declared and a player does not provide a value, the player cannot be matched.

- **Teams** (required) – This section defines the structure and size of teams for the match.
  - You can define as many teams as you want.
  - Each team must have a unique name.
Create a Matchmaking Rule Set

Create matchmaking rule sets for your FlexMatch matchmakers. Use either the Amazon GameLift console or the AWS Command Line Interface (CLI). For more detailed information on configuring a matcher, see the section called “Build a Rule Set” (p. 93).

Once created, matchmaking rule sets cannot be changed or deleted, so we recommend checking the rule set syntax before creating the rule set. Both the console and the AWS CLI provide a validation option.

Create a Matchmaking Rule Set (Console)

To create a rule set with the Amazon GameLift console:

1. Open the Amazon GameLift console at https://console.aws.amazon.com/gamelift/.
2. Switch to the region where you want to place your rule set. Rule sets must be defined in the same region as the matchmaking configuration they will be used with.
3. From the Amazon GameLift main menu, choose Create matchmaking rule set and fill in the rule set details.

   • **Rule set name** – Create a meaningful name so you can easily identify it in a list and in events and metrics. The rule set name must be unique within a region. Matching configurations identify which rule set to use by its name. Note: This is not the same as the "name" field in the rule set body, which is not currently used.
   
   • **Rule set** – Enter the JSON text of a rule set body. Create your own rule set body or copy from the FlexMatch Rule Set Examples (p. 95) page.
4. Since rule sets can’t be edited once they’re created, it’s a good idea to validate your rule set first. Click **Validate rule set** to verify that the syntax of your rule set body is correct.

5. Once you’ve finished configuring a matchmaker, click **Create rule set**. If creation is successful, the rule set can be used by a matchmaker.

### Create a Matchmaking Rule Set (AWS CLI)

To create a matchmaking rule set with the AWS CLI, open a command line window and use the `create-matchmaking-rule-set` command to define a new rule set. See complete documentation on this command in the [AWS CLI Command Reference](https://docs.aws.amazon.com/cli/latest/reference/gamelift/create-matchmaking-rule-set.html). If you haven’t yet installed the AWS CLI, see [Install the AWS CLI](https://docs.aws.amazon.com/cli/latest/userguide/cli-install-linux.html) (p. 18).

This example creates a simple matchmaking rule set that sets up a single team. Be sure to create the rule set in the same region as the matchmaking configurations that will reference it.

```bash
$ aws gamelift create-matchmaking-rule-set
--name "SampleRuleSet123"
--rule-set-body '{"name": "aliens_vs_cowboys",
    "ruleLanguageVersion": "1.0",
    "teams": [{
        "name": "cowboys",
        "maxPlayers": 8,
        "minPlayers": 4}]}'
```

Copiable version:

```bash
aws gamelift create-matchmaking-rule-set --name "SampleRuleSet123" --rule-set-body
'{"name": "aliens_vs_cowboys", "ruleLanguageVersion": "1.0", "teams": [{"name": "cowboys", "maxPlayers": 8, "minPlayers": 4}]}'
```

If the creation request is successful, Amazon GameLift returns a **MatchmakingRuleSet** object that includes the settings you specified. The new rule set can now be used by a matchmaker.

### FlexMatch Rule Set Examples

FlexMatch rule sets can cover a variety of matchmaking scenarios. The following examples conform to the FlexMatch configuration structure and property expression language. Copy these rule sets in their entirety or choose components as needed.

For more information on using FlexMatch rules and rule sets, see the following topics:

- [Build a FlexMatch Rule Set](https://docs.aws.amazon.com/gamelift/latest/developerguide/gamematchflexmatch.html) (p. 93)
- [FlexMatch Rules Reference](https://docs.aws.amazon.com/gamelift/latest/developerguide/gamematchflexmatchapi.html) (p. 172)

**Note**

When evaluating a matchmaking ticket that includes multiple players, all players in the request must meet the match requirements.

### Example 1: Create Two Teams with Evenly Matched Players

This example illustrates how to set up two equally matched teams of players with the following instructions.

- Create two teams of players.
- Include between four and eight players in each team.
• Final teams must have the same number of players.
• Include a player’s skill level (if not provided, default to 10).
• Choose players based on whether their skill level is similar to other players. Ensure that both teams have an average player skill within 10 points of each other.
• If the match is not filled quickly, relax the player skill requirement to complete a match in reasonable time.
  • After 5 seconds, expand the search to allow teams with average player skills within 50 points.
  • After 15 seconds, expand the search to allow teams with average player skills within 100 points.

Notes on using this rule set:

• This example allows for teams to be any size between four and eight players (although they must be the same size). For teams with a range of valid sizes, the matchmaker makes a best-effort attempt to match the maximum number of allowed players.
• The FairTeamSkill rule ensures that teams are evenly matched based on player skill. To evaluate this rule for each new prospective player, FlexMatch tentatively adds the player to a team and calculates the averages. If rule fails, the prospective player is not added to the match.

```json
{
  "name": "aliens_vs_cowboys",
  "ruleLanguageVersion": "1.0",
  "playerAttributes": [{
    "name": "skill",
    "type": "number",
    "default": 10
  }],
  "teams": [{
    "name": "cowboys",
    "maxPlayers": 8,
    "minPlayers": 4
  }, {
    "name": "aliens",
    "maxPlayers": 8,
    "minPlayers": 4
  }],
  "rules": [{
    "name": "FairTeamSkill",
    "description": "The average skill of players in each team is within 10 points from the average skill of all players in the match",
    "type": "distance",
    // get skill values for players in each team and average separately to produce list of two numbers
    "measurements": [ "avg(teams[*].players.attributes[skill])" ],
    // get skill values for players in each team, flatten into a single list, and average to produce an overall average
    "referenceValue": "avg(flatten(teams[*].players.attributes[skill]))",
    "maxDistance": 10 // minDistance would achieve the opposite result
  }, {
    "name": "EqualTeamSizes",
    "description": "Only launch a game when the number of players in each team matches, e.g. 4v4, 5v5, 6v6, 7v7, 8v8",
    "type": "comparison",
    "measurements": [ "count(teams[cowboys].players)" ],
    "referenceValue": "count(teams[aliens].players)",
    "operation": "=" // other operations: !=, <, <=, >, >=
  }],
  "expansions": [{
    "target": "rules[FairTeamSkill].maxDistance",
    "steps": [{
```

Version
96
Example 2: Create Uneven Teams (Hunters vs. Monster)

This example describes a game mode in which a group of players hunt a single monster. People choose either a hunter or a monster role. Hunters specify the minimum skill level for the monster that they want to face. The minimum size of the hunter team can be relaxed over time to complete the match. This scenario sets out the following instructions:

- Create one team of exactly five hunters.
- Create a separate team of exactly one monster.
- Include the following player attributes:
  - A player's skill level (if not provided, default to 10).
  - A player's preferred monster skill level (if not provided, default to 10).
  - Whether the player wants to be the monster (if not provided, default to 0 or false).
- Choose a player to be the monster based on the following criteria:
  - Player must request the monster role.
  - Player must meet or exceed the highest skill level preferred by the players who are already added to the hunter team.
- Choose players for the hunter team based on the following criteria:
  - Players who request a monster role cannot join the hunter team.
  - If the monster role is already filled, player must want a monster skill level that is lower than the skill of the proposed monster.
- If a match is not filled quickly, relax the hunter team's minimum size as follows:
  - After 30 seconds, allow a game to start with only four players in the hunter team.
  - After 60 seconds, allow a game to start with only three people in the hunter team.

Notes on using this rule set:

- By using two separate teams for hunters and monster, you can evaluate membership based on different sets of criteria.
Example 3: Set Team Requirements and Latency Limits

This example illustrates how to set up three equally matched teams of players and apply a set of rules to a team instead of each individual player. It also establishes a maximum latency for all players. Latency maximums can be relaxed over time to complete the match. This scenario sets out the following instructions:

- Create three teams of players.
- Include between three and five players in each team.
- Final teams must contain the same or nearly the same number of players (within one).
- Include the following player attributes:
  - A player’s skill level (if not provided, default to 10).
  - A player’s character role (if not provided, default to “peasant”).
- Choose players based on whether their skill level is similar to other players in the match.
- Ensure that each team has an average player skill within 10 points of each other.
• Limit teams to the following number of “medic” characters:
  • An entire match can have a maximum of five medics.
  • Only match players with 50ms latency at most.
• If a match is not filled quickly, relax the player latency requirement as follows:
  • After 10 seconds, allow player latency values up to 100 ms.
  • After 20 seconds, allow player latency values up to 150 ms.

Notes on using this rule set:
• The rule set ensures that teams are evenly matched based on player skill. To evaluate the FairTeamSkill rule, FlexMatch tentatively adds the prospective player to a team, calculates the average skill of players in the team, and compares it against the average skill of players in both teams. If rule fails, the prospective player is not added to the match.
• The team- and match-level requirements (total number of medics) are achieved through a collection rule. This rule type takes a list of character attributes for all players and checks against the maximum counts. Use flatten to create a list for all players in all teams.
• When evaluating based on latency, note the following:
  • Latency data is provided in the matchmaking request as part of the Player object. It is not a player attribute, so it does not need to be listed as one.
  • The matchmaker evaluates latency by region. Any region with a latency higher than the maximum is ignored. To be accepted for a match, a player must have at least one region with a latency below the maximum.
  • If a matchmaking request omits latency data one or more players, the request is rejected for all matches.

```json
{
  "name": "three_team_game",
  "ruleLanguageVersion": "1.0",
  "playerAttributes": [{
    "name": "skill",
    "type": "number",
    "default": 10
  }, {
    "name": "character",
    "type": "string_list",
    "default": [ "peasant" ]
  }],
  "teams": [{
    "name": "red",
    "minPlayers": 3,
    "maxPlayers": 5
  }, {
    "name": "blue",
    "minPlayers": 3,
    "maxPlayers": 5
  }, {
    "name": "green",
    "minPlayers": 3,
    "maxPlayers": 5
  }],
  "rules": [{
    "name": "FairTeamSkill",
    "description": "The average skill of players in each team is within 10 points from the average skill of players in the match",
    "type": "distance",
    // get players for each team, and average separately to produce list of 3
    "measurements": [ "avg(teams[*].players.attributes[skill])" ]
  }]
}
```
Example 4: Use Explicit Sorting to Find Best Matches

This example sets up a simple match with two teams of three players. It illustrates how to use explicit sorting rules to help find the best possible matches as quickly as possible. These rules presort all active matchmaking tickets to create the best matches based on certain key requirements. This scenario is implemented with the following instructions:

- Create two teams of players.
- Include exactly three players in each team.
- Include the following player attributes:
  - Experience level (if not provided, default to 50).
  - Preferred game modes (can list multiple values) (if not provided, default to “coop” and “deathmatch”).
  - Preferred game maps, including map name and preference weighting (if not provided, default to “defaultMap” with a weight of 100).
- Set up presorting:
  - Sort players based on their preference for the same game map as the anchor player. Players can have multiple favorite game maps, so this example uses a preference value.
  - Sort players based on how closely their experience level matches the anchor player. With this sort, all players in all teams will have experience levels that are as close as possible.
• All players across all teams must have selected at least one game mode in common.
• All players across all teams must have selected at least one game map in common.

Notes on using this rule set:
• The game map sort uses an absolute sort that compares the mapPreference attribute value. It is placed first in the rule set to prioritize it.
• The experience sort uses a distance sort that compares the distance of a prospective player from the anchor player.
• The order of the sorting rules determines the order that each sort is performed. In this scenario, players are sorted by game map preference, and then by experience level.

```json
{
    "name": "multi_map_game",
    "ruleLanguageVersion": "1.0",
    "playerAttributes": [{
        "name": "experience",
        "type": "number",
        "default": 50
    }, {
        "name": "gameMode",
        "type": "string_list",
        "default": [ "deathmatch", "coop" ]
    }, {
        "name": "mapPreference",
        "type": "string_number_map",
        "default": { "defaultMap": 100 }
    }, {
        "name": "acceptableMaps",
        "type": "string_list",
        "default": [ "defaultMap" ]
    }],
    "teams": [{
        "name": "red",
        "maxPlayers": 3,
        "minPlayers": 3
    }, {
        "name": "blue",
        "maxPlayers": 3,
        "minPlayers": 3
    }],
    "rules": [{
        // We placed this rule first since we want to prioritize players preferring the same map
        "name": "MapPreference",
        "description": "Favor grouping players that have the highest map preference aligned with the anchor’s favorite",
        // This rule is just for sorting potential matches. We sort by the absolute value of a field.
        "type": "absoluteSort",
        // Highest values go first
        "sortDirection": "descending",
        // Sort is based on the mapPreference attribute.
        "sortAttribute": "mapPreference",
        // We find the key in the anchor's mapPreference attribute that has the highest value.
        // That’s the key that we use for all players when sorting.
        "mapKey": "mapKey",
    }]
}
```
Example 5: Find intersections across multiple player attributes

This example illustrates how to use a collection rule to find intersections in two or more player attributes. When working with collections, you can use the `intersection` operation for a single attribute, and the `reference_intersection_count` operation for multiple attributes.

To illustrate this approach, this example evaluates players in a match based on their character preferences. The example game is a 'free-for-all' style in which all players in a match are opponents. Each player is asked to (1) choose a character for themselves, and (2) choose characters they want to play against. We need a rule that ensures that every player in a match is using a character that is on all other players' preferred opponents list.

The example rule set describes a match with the following characteristics:

- Team structure: One team of five players
- Player attributes:
  - `myCharacter`: The player's chosen character.
  - `preferredOpponents`: List of characters that the player wants to play against.
- Match rules: A potential match is acceptable if each character in use is on every player's preferred opponents list.

To implement the match rule, this example uses a collection rule with the following property values:

- Operation – Uses `reference_intersection_count` operation to evaluate how each string list in the measurement value intersects with the string list in the reference value.
- Measurement – Uses the `flatten` property expression to create a list of string lists, with each string list containing one player's `myCharacter` attribute value.
- Reference value – Uses the `set_intersection` property expression to create a string list of all `preferredOpponents` attribute values that are common to every player in the match.
- Restrictions – `minCount` is set to 1 to ensure that each player's chosen character (a string list in the measurement) matches at least one of the preferred opponents common to all players. (a string in the reference value).
- Expansion – If a match is not filled within 15 seconds, relax the minimum intersection requirement.
The process flow for this rule is as follows:

1. A player is added to the prospective match. The reference value (a string list) is recalculated to include intersections with the new player's preferred opponents list. The measurement value (a list of string lists) is recalculated to add the new player's chosen character as a new string list.

2. Amazon GameLift verifies that each string list in the measurement value (the players' chosen characters) intersects with at least one string in the reference value (the players' preferred opponents). Since in this example each string list in the measurement contains only one value, the intersection is either 0 or 1.

3. If any string list in the measurement does not intersect with the reference value string list, the rule fails and the new player is removed from the prospective match.

4. If a match is not filled within 15 seconds, drop the opponent match requirement to fill the remaining player slots in the match.

```json
{
   "name": "preferred_characters",
   "ruleLanguageVersion": "1.0",
   "playerAttributes": [{
      "name": "myCharacter",
      "type": "string_list"
   }, {
      "name": "preferredOpponents",
      "type": "string_list"
   }],
   "teams": [{
      "name": "red",
      "minPlayers": 5,
      "maxPlayers": 5
   }],
   "rules": [{
      "description": "Make sure that all players in the match are using a character that is on all other players' preferred opponents list.",
      "name": "OpponentMatch",
      "type": "collection",
      "operation": "reference_intersection_count",
      "measurements": ["flatten(teams[*].players.attributes[myCharacter])"],
      "referenceValue": "set_intersection(flatten(teams[*].players.attributes[preferredOpponents]))",
      "minCount": 1
   }],
   "expansions": [{
      "target": "rules[OpponentMatch].minCount",
      "steps": [{
         "waitTimeSeconds": 15,
         "value": 0
      }]
   }]
}
```

**Example 6: Compare Attributes Across All Players**

This example illustrates how to compare player attributes across a group of players.

The example rule set describes a match with the following characteristics:

- Team structure: Two teams of one player each
• Player attributes:
  • **gameMode**: Type of game chosen by the player (if not provided, default to “turn-based”).
  • **gameMap**: Game world chosen by the player (if not provided, default to 1).
  • **character**: Character chosen by the player (no default value means players must specify a character).

• Match rules: Matched players must meet the following requirements:
  • Players must choose the same game mode.
  • Players must choose the same game map.
  • Players must choose different characters.

To implement the match rule, this example uses comparison rules to check all players’ attribute values. For game mode and map, the rule verifies that the values are the same. For character, the rule verifies that the values are different.

```json
{  
  "name": "",
  "ruleLanguageVersion": "1.0",

  "playerAttributes": [{
    "name": "gameMode",
    "type": "string",
    "default": "turn-based"
  }, {
    "name": "gameMap",
    "type": "number",
    "default": 1
  }, {
    "name": "character",
    "type": "number"
  }],

  "teams": [{
    "name": "red",
    "minPlayers": 1,
    "maxPlayers": 1
  }, {
    "name": "blue",
    "minPlayers": 1,
    "maxPlayers": 1
  }],

  "rules": [{
    "name": "SameGameMode",
    "description": "Only match players when they choose the same game type",
    "type": "comparison",
    "operation": "=",
    "measurements": ["flatten(teams[*].players.attributes[gameMode])"]
  }, {
    "name": "SameGameMap",
    "description": "Only match players when they're in the same map",
    "type": "comparison",
    "operation": "=",
    "measurements": ["flatten(teams[*].players.attributes[gameMap])"]
  }, {
    "name": "DifferentCharacter",
    "description": "Only match players when they're using different characters",
    "type": "comparison",
    "operation": "!=",
    "measurements": ["flatten(teams[*].players.attributes[character])"]
  }]
}
```
Set up FlexMatch Event Notification

If you're using FlexMatch matchmaking in your game, you need a way to track the status of individual matchmaking requests and take action as appropriate. Sometimes, such as when players are required to accept a proposed match, these actions are time sensitive. The simplest method for tracking each request's status is to continuously poll using DescribeMatchmaking. However, for a faster solution, we recommend using notifications to track matchmaking events. Event notifications are simple to set up and are significantly faster and more efficient.

There are two options for setting up event notifications. You can use Amazon CloudWatch Events, which has a suite of tools available for managing events and taking action on them. Alternatively, you can set up your own SNS topic(s) and attach them to your matchmaker to receive matchmaking event notifications directly.

Set up CloudWatch Events

Amazon GameLift automatically posts all matchmaking events to CloudWatch Events. With CloudWatch Events you can set up rules to have matchmaking events routed to a range of targets, including SNS topics and other AWS services for processing. For example, you might set a rule to route the event "PotentialMatchCreated" to an AWS Lambda function that handles player acceptances. Learn more about how to use CloudWatch Events in the Getting Started guide, which includes a collection of tutorials.

If you plan to use CloudWatch Events, when configuring your matchmakers, you can leave the notification target field empty, or reference an SNS topic if you want to use both options.

To access Amazon GameLift matchmaking events in CloudWatch Events, go to the Amazon CloudWatch console and open Events. Be sure that you're in the region where you've set up your matchmaking configuration. For more information about getting account credentials to access CloudWatch Events, see Sign in to the Amazon CloudWatch Console. Each matchmaking event is identified by the service (GameLift), the matchmaking name, and the matchmaking ticket.

Set up an SNS Topic

You can ask Amazon GameLift to publish all events generated by a FlexMatch matchmaker to an Amazon Simple Notification Service (SNS) topic. When configuring the matchmaker, set the notification target field to an SNS topic ARN.

To set up an SNS topic for Amazon GameLift event notification

1. Go to the Amazon Simple Notification Service console.
2. Create a topic. From the SNS dashboard, choose Create topic and follow the instructions to create your topic. When the topic is created, the console automatically opens the Topic details page for the new topic.
3. Allow Amazon GameLift to publish to the topic. If you're not already in the Topic details page for your topic, choose Topics from the navigation bar and click the topics ARN to open it. Choose the topic action Edit topic policy, and go to the Advanced view tab.

Add the bolded syntax below to the end of your existing policy. (The entire policy is shown for clarity.)

```json
{
    "Version": "2008-10-17",
    "Id": "_default_policy_ID",
    "Statement": [
    {
```
Add FlexMatch to a Game Client

This topic describes how to initiate the FlexMatch matchmaking process for your players. To learn more about FlexMatch and how to set it up for your games, see the these topics:

- Amazon GameLift FlexMatch (p. 10)
- FlexMatch Integration Guide (p. 89)

**Note**
When requesting matchmaking for your players, we highly recommend making these requests through a client-side game service, and not directly from your game client. By using a trusted source, you can more easily protect against hacking attempts and also avoid the use of fake player data that can ruin the game experience for your players. If your game uses a session directory service, this is a good option for handling matchmaking requests.

As with other client functionality, your game service enables matchmaking by using the AWS SDK with the Amazon GameLift API. This SDK is available in C++, C#, and several other languages. For a general description of client APIs, see the Amazon GameLift Service API Reference, which describes the low-level service API for Amazon GameLift-related actions and includes links to language-specific reference guides.

The information in this topic assumes that you've (1) created a matchmaking configuration (see Design a FlexMatch Matchmaker (p. 90)); and (2) successfully integrated Amazon GameLift into the game service (see Add Amazon GameLift to Your Game Client (p. 33)). To test FlexMatch in your game, you also
need to set up your game server to properly start game sessions for matches (see Add FlexMatch to a Game Server (p. 110).

To enable FlexMatch for your game clients, you’ll need to add the following functionality:

- Request matchmaking for one or more players.
- Track the status of matchmaking requests.
- Request player acceptance for a proposed match.
- Once a game session is created for the new match, get player connection information and join the game.

When using FlexMatch, you may want to set up backfilling for your games. Match backfill helps keep game sessions filled as players leave the game, and is particularly useful in games with longer life spans. It ensures that new players who enter a game meet the same match criteria as existing players. Learn more about match backfill in How Amazon GameLift FlexMatch Works (p. 11). For help setting it up, see Backfill Existing Games with FlexMatch (p. 112).

**Request Matchmaking for Players**

Add code to create and send a matchmaking request to a FlexMatch matchmaker. To manage matchmaking requests, use these APIs:

- StartMatchmaking
- StopMatchmaking

To create a matchmaking request, call StartMatchmaking with the following information.

**Matchmaker**

Specify the name of the matchmaking configuration to use for the request. FlexMatch places each request into the request pool for the specified matchmaker, and the request will be processed according to the matchmaker's configuration and rule set.

**Ticket ID**

Specify a ticket ID. All matchmaking requests must be given a unique ticket ID. You are responsible for generating, assigning, and managing ticket IDs. You can use any string format, up to a maximum of 128 characters. Ticket IDs are attached to all requests, events and notifications related to matchmaking.

**Player data**

Specify one or more players you want to create a match for. If you include multiple players in a single request, FlexMatch tries to create a match for all the players and place them into the same team. All players in a request must pass matchmaking rules for the request to be matched.

**Note**

A player (identified by their player ID) can only be included in one active matchmaking request at a time. If you create a new request for a player, any active matchmaking tickets with the same player ID are automatically cancelled.

- **Player ID** – Each player must have a unique player ID, generated by you. For more information, see Generate Player IDs (p. 37).
- **Player attributes** – For each player in a request, there should be a set of player attribute values as required by the matchmaker; these are defined in the matchmaker's rule set and support rules that evaluate potential matches based on player attributes like individual skill level. The attribute name and data type in the matchmaking request must match that defined in the matchmaker's rule set. Attributes in a rule set can be configured with a default value to be used when a matchmaking request does not provide one. However, if an attribute has no default value, and the matchmaking request
request does not provide one, the player cannot be matched (and the matchmaking request will never succeed). For more information on matchmaker rule sets and player attributes, see the section called “Build a Rule Set” (p. 93).

- **Player latencies** – Player latency data contains reported latency values for one or more regions. Player latency data may be used to find matches between players who report similar latencies; when provided, it is also used to place a match's game session in a region that offers the lowest possible latency for all players in the match.

If a matchmaker has a rule that evaluates player latency, players must report latency to be matched. In this case, players can only be matched in regions that a latency value is provided for.

Additional factors that affect a matchmaking request are set in the matchmaking configuration, including a time limit for requests, and optional player acceptance of matches. All matchmaking requests sent to a matchmaker are processed as configured. For more information on configuring a matchmaker and setting match rules, see Design a FlexMatch Matchmaker (p. 90).

A matchmaking request can be cancelled at any time using **StopMatchmaking**.

### Track Matchmaking Request Status

Add code to game service to track the status of all matchmaking requests and respond as needed. There are a couple of options available for tracking status.

#### Event notifications

Set up notifications to track events that Amazon GameLift emits for matchmaking processes. This is the recommended method; it is simple to set up and an efficient use of resources. You can set up notifications either directly, by creating an SNS topic, or by using CloudWatch Events. For more information on setting up notifications, see Set up FlexMatch Event Notification (p. 105). Once you've set up notifications, you need to add a listener on your game service to detect the events and respond as needed. It's also a good idea to poll for status updates if you haven't received a notification after 30 seconds.

#### Continuous polling

Retrieve a matchmaking request ticket, including current status, by calling DescribeMatchmaking. This call returns complete information about the matchmaking request. Once a request has been successfully completed, the ticket also contains the information needed to connect a player to the match. We recommend polling no more than once every 10 seconds.

### Request Player Acceptance

For requests that use a matchmaker with player acceptance turned on, your game service must be able to do the following:

1. **Detect when player acceptance is required.** Your game service must be able to detect when a matchmaking ticket's status is **REQUIRES_ACCEPTANCE**. If you're monitoring notifications, a change to this status triggers the FlexMatch event **MatchmakingRequiresAcceptance**.

2. **Get acceptances from all players.** Your game service must have a mechanism to present the match details to each player in the matchmaking ticket and solicit an acceptance or rejection. You can retrieve match details by calling DescribeMatchmaking.

3. **Report player responses to FlexMatch.** Report each player acceptance by calling **AcceptMatch** with either accept or reject. All players in a matchmaking request must accept the match for it to go forward.

4. **Handle tickets with failed acceptances.** Your game service should have a way to handle matchmaking requests that fail player acceptance. This includes requests where any player in the request either rejects the match or fails to respond by the acceptance time limit.
# Connect to a Match

Add code to your game service handle to a successfully completed match (status `COMPLETED` or event `MatchmakingSucceeded`) as needed. This includes handing off the match's connection information to game clients. When a matchmaking request is completed, connection information is added to the matchmaking ticket. Connection information (see `GameSessionConnectionInfo`) includes the game session IP address and port, as well as a set of player IDs and player session IDs for all players in the match. Retrieve a completed matchmaking ticket by calling `DescribeMatchmaking`.

Your game client uses this information to connect directly to the server process that is hosting the match. A connection request for a matched game session must include a player session ID and a player ID. As with all game session connections, the player session ID is used to validate the player's connection request and claim a reserved slot in the game session. For matched game sessions, the player ID associates the connected player to a set of matchmaker data, including team assignments, that describe the game session (see `GameSession`).

# Sample StartMatchmaking Requests

The following code example builds matchmaking requests for a variety of matchmakers. As discussed, each matchmaker has its own requirements for player attributes, and matchmaking requests must match those requirements.

```python
# Uses matchmaker for two-team game mode based on player skill level
def start_matchmaking_for_cowboys_vs_aliens(config_name, ticket_id, player_id, skill, team):
    response = gamelift.start_matchmaking(
        ConfigurationName=config_name,
        Players=[{
            "PlayerAttributes": {
                "skill": {"N": skill}
            },
            "PlayerId": player_id,
            "Team": team
        }],
        TicketId=ticket_id)

# Uses matchmaker for monster hunter game mode based on player skill level
def start_matchmaking_for_players_vs_monster(config_name, ticket_id, player_id, skill, is_monster):
    response = gamelift.start_matchmaking(
        ConfigurationName=config_name,
        Players=[{
            "PlayerAttributes": {
                "skill": {"N": skill},
                "desiredSkillOfMonster": {"N": skill},
                "wantsToBeMonster": {"N": int(is_monster)}
            },
            "PlayerId": player_id
        }],
        TicketId=ticket_id)

# Uses matchmaker for brawler game mode with latency
def start_matchmaking_for_three_team_brawler(config_name, ticket_id, player_id, skill, role):
    response = gamelift.start_matchmaking(
        ConfigurationName=config_name,
        Players=[{
            "PlayerAttributes": {
                "skill": {"N": skill},
                "character": {"S": [role]}
            }
        }],
        TicketId=ticket_id)
```
Add FlexMatch to a Game Server

This topic describes how to integrate Amazon GameLift FlexMatch into your game server. This is one of several tasks required to add FlexMatch features to your game; see FlexMatch Integration Guide (p. 89) for the full process. For a detailed description of FlexMatch's customized matchmaking, see Amazon GameLift FlexMatch (p. 10).

The information in this topic assumes that you've successfully integrated the Amazon GameLift Server SDK into your game server project, as described in Add Amazon GameLift to Your Game Server (p. 30). With this work completed, you have most of the mechanisms you need. The sections in this topic cover the remaining work needed to handle games that are set up with FlexMatch.

Set Up Your Game Server for Matchmaking

To set up your game server to handle matched games, complete the following tasks.

1. **Start game sessions created with matchmaking.** To request a new game session, Amazon GameLift sends an `onStartGameSession()` request to your game server with a game session object (see GameSession). Your game server uses the game session information, including customized game data, to start the requested game session. For more details, see Start a Game Session (p. 31).

   For matched games, the game session object contains a set of matchmaker data, described later in this topic. Matchmaker data includes information that your game server needs to set up a new game session for the match. This includes the match's team structure, team assignments, and certain player attributes. For example, you might use the average skill level of the players to unlock certain features or levels. As another example, you might allow players to choose multiple map preferences and then build matches with just one common map choice. In this scenario, you would use the player attribute data to determine the right map for the game session. See Work with Matchmaker Data (p. 111) for more details.

2. **Handle player connections.** When connecting to a matched game, a game client references a player ID as well as the player session ID used for validation (see Validate a New Player (p. 32)). Your code must use the player ID to associate an incoming player with player information in the matchmaker data. Matchmaker data identifies each player's team assignment, and may provide other information required to correctly represent the player in the game. For example, player attributes might determine the player's game character, location or status in the game, etc.

3. **Report when players leave a game.** Ensure that your game server is calling the Server API `RemovePlayerSession()` to report dropped players (see Report a Player Session Ending (p. 32)).
This step is particularly important if you're using FlexMatch backfill to fill empty slots in existing games, and critical when initiating backfill requests in a client-side game service. For more on implementing FlexMatch backfill, see Backfill Existing Games with FlexMatch (p. 112).

4. **Request new players for existing matched game sessions (optional).** If you intend to use the match backfill feature to add new players to existing games, you may want to send backfill requests directly from the game server. For more details, including request options, see Backfill Existing Games with FlexMatch (p. 112). Learn more about the match backfill feature in How Amazon GameLift FlexMatch Works (p. 11).

## Work with Matchmaker Data

Your game server must be able to recognize and use the game information in a `GameSession` object. The Amazon GameLift service passes these objects to your game server whenever a game session is started or updated. Core game session information includes game session ID and name, maximum player count, connection information, and custom game data (if provided).

For game sessions that are created using FlexMatch, the `GameSession` object also contains a set of matchmaker data that was generated as part of the matchmaking process. In addition to a unique match ID, it identifies the matchmaker that was used to create the match and describes the teams, team assignments, and players. It includes the player attributes from the original matchmaking request (see the `Player` object). It doesn't include the player latency; if you need latency data on current players, such as for match backfill, we recommend getting fresh data.

**Note**

Matchmaker data specifies the full matchmaking configuration ARN, which identifies the configuration name, AWS account, and region. When requesting match backfill, you need either the full configuration ARN (when requesting from a game server) or just the configuration name (when requesting from a game client or client service). You can extract the configuration name from an ARN value by parsing out the string that follows ":matchmakingconfiguration/". In the example shown, the matchmaking configuration name is "MyMatchmakerConfig".

The following JSON shows a typical set of matchmaker data. This example describes a two-player game, with players matched based on skill ratings and highest level attained. The matchmaker also matched based on character, and ensured that matched players have at least one map preference in common. In this example scenario, the game server should be able to detect the common map—or select from multiple common preferences—to determine which map to assign for the game session.

```json
{
    "matchId":"111aaaaa-22bb-33cc-44dd-5555eeeee66ff",
    "teams": [
        {
            "name":"attacker",
            "players": [
                {
                    "playerId":"4444dddd-55ee-66ff-77aa-8888bbbb99cc",
                    "attributes": {
                        "skills": {
                            "attributeType": "STRING_DOUBLE_MAP",
                            "valueAttribute": {"Body": 10.0, "Mind": 12.0, "Heart": 15.0, "Soul": 33.0} 
                        }
                    }
                },
                {
                    "playerId":"3333cccc-44dd-55ee-66ff-7777aaaa88bb",
                    "attributes": {
                        "skills": {
                            "attributeType": "STRING_DOUBLE_MAP",
                            "valueAttribute": {"Body": 11.0, "Mind": 12.0, "Heart": 11.0, "Soul": 40.0} 
                        }
                    }
                }
            ]
        },
        {
            "name": "defender",
            "players": [
                {
                    "playerId": "3333cccc-44dd-55ee-66ff-7777aaaa88bb",
                    "attributes": {
                        "skills": {
                            "attributeType": "STRING_DOUBLE_MAP",
                            "valueAttribute": {"Body": 11.0, "Mind": 12.0, "Heart": 11.0, "Soul": 40.0} 
                        }
                    }
                }
            ]
        }
    ]
}
```
Backfill Existing Games with FlexMatch

Match backfill uses your FlexMatch mechanisms to match new players to existing matched game sessions. Although you can always add players to any game (see Join a Player to a Game Session (p. 36)), match backfill ensures that new players meet the same match criteria as current players. In addition, match backfill assigns the new players to teams, manages player acceptance, and sends updated match information to the game server process that is running the game session. Learn more about match backfill in How Amazon GameLift FlexMatch Works (p. 11).

This topic describes how to set up match backfill for your game. It assumes that you’ve already built the necessary FlexMatch components and successfully added matchmaking processes to your game server and a client-side game service. For more details on setting up FlexMatch, see FlexMatch Integration Guide (p. 89).

To enable match backfill for your game, you'll need to add the following functionality:

- Send matchmaking backfill requests to a matchmaker and track the status of requests.
- Update match information for the game session.

Send Backfill Requests (From a Game Server)

You can initiate match backfill requests directly from the game server process that is hosting the game session. The server process has the most up-to-date information on current players connected to the game and the status of empty player slots.

As with other server functionality, a game server uses the Amazon GameLift Server SDK. This SDK is available in C++ and C#. For a general description of server APIs, see the Server API references (p. 137).

To make match backfill requests from your game server, complete the following tasks.

1. **Trigger a match backfill request.** Generally, you want to initiate a backfill request whenever a matched game has one or more empty player slots. You may want to tie backfill requests to specific circumstances, such as to fill critical character roles or balance out teams. You'll likely also want to limit backfilling activity based on a game session's age.

2. **Create a backfill request.** Add code to create and send match backfill requests to a FlexMatch matchmaker. Backfill requests are handled using these server APIs:
   - `StartMatchBackfill()` (p. 145)
   - `StopMatchBackfill()` (p. 146)

To create a backfill request, call `StartMatchBackfill` with the following information. To cancel a backfill request, call `StopMatchBackfill` with the backfill request's ticket ID.

- **Ticket ID** — Provide a matchmaking ticket ID (or opt to have them autogenerated). You can use the same mechanism to assign ticket IDs to both matchmaking and backfill requests. Tickets for matchmaking and backfilling are processed the same way.
- **Matchmaker** — Identify which matchmaker to use for the backfill request. Generally, you'll want to use the same matchmaker that was used to create the original match. This request...
takes a matchmaking configuration ARN. This information is stored in the game session object
(GameSession), which was provided to the server process by Amazon GameLift when activating
the game session. The matchmaking configuration ARN is included in the MatchmakerData
property.

• **Game session ARN** — Identify the game session being backfilled. You can get the game session
ARN by calling the server API GetGameSessionId() (p. 139). During the matchmaking process,
tickets for new requests do not have a game session ID, while tickets for backfill requests do. The
presence of at game session ID is one way to tell the difference between tickets for new matches
and tickets for backfills.

• **Player data** — Include player information (Player) for all current players in the game session you
are backfilling. This information allows the matchmaker to locate the best possible player matches
for the players currently in the game session. If your game server has been accurately reporting
player connection status, you should be able to acquire this data as follows:

1. The server process hosting the game session should have the most up-to-date information
which players are currently connected to the game session.

2. To get player IDs, attributes, and team assignments, pull player data from the game session
object (GameSession), MatchmakerData property (see Work with Matchmaker Data (p. 111). The
matchmaker data includes all players who were matched to the game session, so you'll
need to pull the player data for only the currently connected players.

3. For player latency, if the matchmaker calls for latency data, collect new latency values from
all current players and include it in each Player object. If latency data is omitted and the
matchmaker has a latency rule, the request will not be successfully matched. Backfill requests
require latency data only for the region that the game session is currently in. You can get a
game session's region from the GameSessionId property of the GameSession object; this
value is an ARN, which includes the region.

3. **Track the status of a backfill request.** Amazon GameLift updates your game server about the
status of backfill requests using the Server SDK callback function onUpdateGameSession (see
Prepare a Server Process (p. 30)). Add code to handle the status messages—as well as updated
game session objects as a result of successful backfill requests—at Update Match Data on the Game
Server (p. 115).

A matchmaker can process only one match backfill request from a game session at a time. If you
need to cancel a request, call StopMatchBackfill() (p. 146). If you need to change a request, call
StopMatchBackfill and then submit an updated request.

**Send Backfill Requests (From a Client Service)**

As an alternative to sending backfill requests from a game server, you may want to send them from a
client-side game service. To use this option, the client-side service must have access to current data on
game session activity and player connections; if your game uses a session directory service, this might be
a good choice.

As with other client functionality, a client-side game service uses the AWS SDK with Amazon GameLift
API. This SDK is available in C++, C#, and several other languages. For a general description of client APIs,
see the Amazon GameLift Service API Reference, which describes the low-level service API for Amazon
GameLift-related actions and includes links to language-specific reference guides.

To set up a client-side game service to backfill matched games, complete the following tasks.

1. **Trigger a request for backfilling.** Generally, a game initiates a backfill request whenever a
matched game has one or more empty player slots. You may want to tie backfill requests to specific
circumstances, such as to fill critical character roles or balance out teams. You'll likely also want
to limit backfilling based on a game session's age. Whatever you use for a trigger, at a minimum
you'll need to the following information. You can get this information from the game session object
(GameSession) by calling DescribeGameSessions with a game session ID.
• **Number of currently empty player slots.** This value can be calculated from a game session’s maximum player limit and the current player count. Current player count is updated whenever your game server contacts the Amazon GameLift service to validate a new player connection or to report a dropped player.

• **Creation policy.** This setting indicates whether the game session is currently accepting new players.

The game session object contains other potentially useful information, including game session start time, custom game properties, and matchmaker data.

2. **Create a backfill request.** Add code to create and send match backfill requests to a FlexMatch matchmaker. Backfill requests are handled using these client APIs:

- **StartMatchBackfill**
- **StopMatchmaking**

To create a backfill request, call `StartMatchBackfill` with the following information. A backfill request is similar to a matchmaking request (see Request Matchmaking for Players (p. 107)), but also identifies the existing game session. To cancel a backfill request, call `StopMatchmaking` with the backfill request’s ticket ID.

- **Ticket ID** — Provide a matchmaking ticket ID (or opt to have them autogenerated). You can use the same mechanism to assign ticket IDs to both matchmaking and backfill requests. Tickets for matchmaking and backfilling are processed the same way.

- **Matchmaker** — Identify the name of the matchmaking configuration to use. Generally, you’ll want to use the same matchmaker for backfilling that was used to create the original match. This information is in a game session object (`GameSession`), `MatchmakerData` property, under the matchmaking configuration ARN. The name value is the string following ""matchmakingconfiguration/". (For example, in the ARN value "arn:aws:gamelift:us-west-2:111122223333:matchmakingconfiguration/MM-4v4", the matchmaking configuration name is "MM-4v4").

- **Game session ARN** — Specify the game session being backfilled. Use the `GameSessionId` property from the game session object; this ID uses the ARN value that you need. Matchmaking tickets (`MatchmakingTicket`) for backfill requests have the game session ID while being processed; tickets for new matchmaking requests do not get a game session ID until the match is placed; the presence of a game session ID is one way to tell the difference between tickets for new matches and tickets for backfills.

- **Player data** — Include player information (`Player`) for all current players in the game session you are backfilling. This information allows to matchmaker to locate the best possible player matches for the players currently in the game session. If your game server has been accurately reporting player connection status, you should be able to acquire this data as follows:

  1. Call `DescribePlayerSessions()` with the game session ID to discover all players who are currently connected to the game session. Each player session includes a player ID. You can add a status filter to retrieve active player sessions only.

  2. Pull player data from the game session object (`GameSession`), `MatchmakerData` property (see Work with Matchmaker Data (p. 111)). Use the player IDs acquired in the previous step to get data for currently connected players only. Since matchmaker data is not updated when players drop out, you will need extract the data for current players only.

  3. For player latency, if the matchmaker calls for latency data, collect new latency values from all current players and include it in the `Player` object. If latency data is omitted and the matchmaker has a latency rule, the request will not be successfully matched. Backfill requests require latency data only for the region that the game session is currently in. You can get a
game session's region from the GameSessionId property of the GameSession object; this value is an ARN, which includes the region.

3. **Track status of backfill request.** Add code to listen for matchmaking ticket status updates. You can use the mechanism set up to track tickets for new matchmaking requests (see Track Matchmaking Request Status (p. 108)) using event notification (preferred) or polling. Although you don't need to trigger player acceptance activity with backfill requests, and player information is updated on the game server, you still need to monitor ticket status to handle request failures and resubmissions.

   A matchmaker can process only one match backfill request from a game session at a time. If you need to cancel a request, call StopMatchmaking. If you need to change a request, call StopMatchmaking and then submit an updated request.

   Once a match backfill request is successful, your game server receives an updated GameSession object and handles the tasks needed to join new players to the game session. See more at Update Match Data on the Game Server (p. 115).

### Update Match Data on the Game Server

No matter how you initiate match backfill requests in your game, your game server must be able to handle the game session updates that Amazon GameLift delivers as a result of match backfill requests.

When Amazon GameLift completes a match backfill request—successfully or not—it calls your game server using the callback function onUpdateGameSession. This call has three input parameters: a match backfill ticket ID, a status message, and a GameSession object containing the most up-to-date matchmaking data including player information. You need to add the following code to your game server as part of your game server integration:

1. Implement the onUpdateGameSession function. This function must be able to handle the following status messages (updateReason):
   - MATCHMAKING_DATA_UPDATED – New players were successfully matched to the game session. The GameSession object contains updated matchmaker data, including player data on existing players and newly matched players.
   - BACKFILL_FAILED – The match backfill attempt failed due to an internal error. The GameSession object is unchanged.
   - BACKFILL_TIMED_OUT – The matchmaker failed to find a backfill match within the time limit. The GameSession object is unchanged.
   - BACKFILL_CANCELLED – The match backfill request was cancelled by a call to StopMatchmaking (client) or StopMatchBackfill (server). The GameSession object is unchanged.

2. For successful backfill matches, use the updated matchmaker data to handle the new players when they connect to the game session. At a minimum, you'll need to use the team assignments for the new player(s), as well as other player attributes that are required to get the player started in the game.

3. In your game server's call to the Server SDK action ProcessReady() (p. 142), add the onUpdateGameSession callback method name as a process parameter.
Viewing Your Game Data in the Console

Amazon GameLift continually collects data for active games to help you understand player behavior and performance. With the Amazon GameLift console, you can view, manage, and analyze this information for your builds, fleets, game sessions, and player sessions.

**Topics**
- View Your Current Amazon GameLift Status (p. 116)
- View Your Builds (p. 117)
- View Your Fleets (p. 118)
- View Fleet Details (p. 118)
- View Data on Game and Player Sessions (p. 122)
- View Your Aliases (p. 123)

**View Your Current Amazon GameLift Status**

The **Dashboard** provides a grid view showing the following:

- Uploaded builds
- Created fleets in all statuses
- Created aliases and the fleets they point to (if any)

**To open the Amazon GameLift dashboard**

- In the Amazon GameLift console, choose **Dashboard** from the menu bar.

From the dashboard you can take the following actions:

- Create a new fleet or alias.
- View relationships among items. Click anywhere inside an item box to show the relationships between that item and others on the dashboard. For example, click a build to display all fleets that were created with that build. Click a fleet to see the build it was created with and the alias it points to. To reset the dashboard, click the **Reset overview** button.
- View details on a build, fleet, or alias. Click the ID number for an item to open the details page.
View Your Builds

You can view information about all the game server builds you have uploaded to Amazon GameLift and take actions on them. Builds shown include only those uploaded for the selected region.

Build Catalog

Uploaded builds are shown on the Builds page. To view this page, choose Builds from the Amazon GameLift console menu bar.

The Builds page provides the following summary information for all builds:

- **Status** – Displays one of three possible status messages:
  - **Initialized** – The build has been created, but the upload has not yet started or the upload is still in progress.
  - **Ready** – The build has been successfully received and is ready for fleet creation.
  - **Failed** – The build timed out before the binaries were received.
- **Build name** – Name associated with the uploaded build. A build name is provided when uploading the build to Amazon GameLift, and can be changed using the AWS SDK action UpdateBuild.
- **Build ID** – Unique ID assigned to the build on upload.
- **Version** – Version label associated with the uploaded build. A build name is provided when uploading the build to Amazon GameLift, and can be changed using the AWS SDK action UpdateBuild.
- **OS** – Operating system that the build runs on. The build OS determines what operating system is installed on a fleet's instances.
- **Size** – Size, in megabytes (MB) of the build file uploaded to Amazon GameLift.
- **Date created** – Date and time that the build was uploaded to Amazon GameLift.
- **Fleets** – Number of fleets currently deployed with this build.
From this page you can do any of the following:

- Create a new fleet from a build. Select a build and click **Create fleet from build**.
- Delete a fleet. Select a build and click **Delete build**.
- Filter and sort the build list. Use the controls at the top of the table.
- View build details. Click a build name to open the build detail page.

## Build Detail

Access a build's detail page from either the console dashboard or the **Builds** page by clicking the build name. The **Build** detail page displays the same build summary information as the **Builds** page. It also shows a list of fleets created with the build. This list is essentially the fleets catalog, filtered by build. It includes the same summary information as the **Fleets** page (p. 118).

## View Your Fleets

You can view information on all the fleets created to host your games on Amazon GameLift under your AWS account. Fleets shown include only those created in the selected region. From the **Fleets** page, you can create a new fleet or view additional detail on one selected fleet. A **Fleet** detail page contains usage information and metrics; it also lets you view and edit fleet configuration settings, create or remove the fleet, and access the fleet's game session and player session data.

To view the **Fleets** page, choose **Fleets** from the Amazon GameLift console's menu bar.

The **Fleets** page displays the following summary information:

- **Status** – The status of the fleet, which can be one of these states: New, Downloading, Building, and Active. A fleet must be in Active status to be able to host game sessions and allow player connections.
- **Fleet name** – Friendly name given to the fleet.
- **EC2 type** – The Amazon EC2 instance type, which determines the computing capacity of fleet's instances.
- **OS** – Operating system on each instances in the fleet. A fleet's OS is determined by the build deployed to it.
- **Active** – The number of EC2 instances in use for the fleet.
- **Desired** – The number of EC2 instances Amazon GameLift should maintain in the fleet. This value is configurable (within service limits). Amazon GameLift starts or terminates instances as needed to maintain the desired number of instances.
- **Active game sessions** – The number of game sessions currently running in the fleet. The data is delayed five minutes.
- **Player sessions** – The number of players connected to game sessions in the fleet. The data is delayed five minutes.
- **Uptime** – The total length of time the fleet has been running.
- **Date created** – The date and time the fleet was created.

## View Fleet Details

You can access detailed information on any fleet, including configuration settings, scaling settings, metrics, and game and player data. Access a **Fleet** detail page from either the console dashboard or the **Fleets** page by clicking the fleet name.
The fleet detail page displays a summary table and tabs containing additional information. On this page you can do the following:

- Update the fleet's metadata and runtime configuration. Choose **Actions: Edit fleet**.
- Change fleet capacity settings. On the Scaling page, edit values from **Minimum**, **Maximum**, and **Desired** instances.
- Set or change auto-scaling policies. On the Scaling page, add or edit a policy.
- Shut down a fleet. Choose **Actions: Terminate fleet**.

**Summary**

The summary table includes the following information:

- **Status** – Current status of the fleet, which may be **New**, **Downloading**, **Building**, and **Active**. A fleet must be active before it can host game sessions or accept player connections.
- **Fleet ID** – Unique identifier assigned to the fleet.
- **EC2 type** – Amazon EC2 instance type selected for the fleet when it was created. A fleet's instance type specifies the computing hardware and capacity used for each instance in the fleet and determines the instance limits for the fleet.
- **OS** – Operating system on each instances in the fleet. A fleet's OS is determined by the build deployed to it.
- **Active instances** – Number of instances in **Active** status, which means they are currently running game sessions or are ready to run game sessions.
- **Active servers** – Number of server processes currently in an **Active** status in the fleet. The data has a five-minute delay.
- **Active game sessions** – Number of game sessions currently running on instances in the fleet. The data has a five-minute delay.
- **Current player sessions** – Number of players currently connected along with the total number of player slots in active game sessions across the fleet. For example: 25 (connected players) of 100 (possible players) means the fleet can support 75 additional players. The data has a five-minute delay.
- **Protection** – Current setting for **game session protection** (p. 3) for the fleet.
- **Uptime** – Total length of time the fleet has been active.
- **Date created** – Date and time indicating when the fleet was created.

**Metrics**

The **Metrics** tab shows a graphical representation of fleet metrics over time.

**To display metrics information in the graph**

1. Click one or more metric name to the left of the graph area to add it to the graph display. Metric names are gray when turned off. Use the color key to identify the graphed line for a selected metric. Descriptions of individual metrics can be found at Amazon GameLift Metrics for Fleets (p. 125). The following categories of metrics are available:

   - **Instance Counts** – These metrics track changes in capacity and utilization at the instance level over time (also shown on the Scaling tab).
   - **Game** – These metrics show fleet activity and utilization at the game session level over time.
   - **Server processes** – These metrics track the status and health of server processes across the fleet. The Amazon GameLift service regularly polls each active server process for its health.
• **Instance performance** – These metrics reflect performance of the fleet's computing resources. See detailed descriptions of each metric at [EC2 Instance Metrics](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-performance-metrics.html).

2. Use the following filters, shown above the graph area, to change how metric data is displayed:

   • **Data & Period** – Offers two options for selecting a date range:
     - Use **Relative** to select a period of time relative to the current time, such as Last hour, Last day, Last week.
     - Use **Absolute** to specify a period with an arbitrary start and end date/time.
   
   • **Granularity** – Select a length of time to aggregate data points.
   
   • **Refresh rate** – Select how often you want the graph display to be updated.
   
   • **Format** – Select which time format to use in the graph display: UTC (universal coordinated time) or Browser time (local time).
   
   • **Show Points** – Toggle on or off to display discrete data points (as circles) or display lines only.

---

**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 API Reference.

**Scaling**

The **Scaling** tab contains information related to fleet capacity, including the current status and a graphical representation of capacity changes over time. It also provides tools to update capacity limits and manage auto-scaling.

**To view current and historical scaling information**

1. Go to the top of the **Scaling** tab. The current capacity status for this fleet is shown in the left column. These values are defined as follows:

   • **Scaling Limits** – These metrics track the history of changes to capacity limits.
     - **Minimum** – Hard lower limit on the number of instances to maintain in the fleet. Fleet capacity will not drop below the current minimum during auto-scaling or even if desired capacity is set below the minimum.
     - **Desired** – The number of active instances currently **wanted** in the fleet. Amazon GameLift's goal is to make the number of **Active** instances (explained later) match the number of desired instances; it achieves this by creating or terminating instances as needed.
     - **Maximum** – Hard upper limit on the number of instances to maintain in the fleet. Fleet capacity will not exceed the current maximum during auto-scaling or if desired capacity is set above the maximum.
   
   • **Instance Counts** – These metrics track changes in capacity and utilization over time. See descriptions of individual metrics at Amazon GameLift Metrics for Fleets (p. 125).

2. To view how fleet capacity has changed over time, display in the graph any or all of the scaling metrics listed on the left. Click the metric name to add it to the graph. (Metric names are gray when not in use.) Use the color key to identify which graphed line matches a selected metric.

3. (Optional) Use the following filters, shown above the graph area, to specify how metric data is displayed in the graph:

   • **Data & Period** – Offers two options for selecting a date range:
     - Use **Relative** to select a period of time relative to the current time, such as Last hour, Last day, Last week.
     - Use **Absolute** to specify a period with an arbitrary start and end date/time.
• **Granularity** – Select a length of time to aggregate data points.
• **Refresh rate** – Select how often you want the graph display to be updated.
• **Format** – Select which time format to use in the graph display: **UTC** (universal coordinated time) or **Browser time** (local time).
• **Show Points** – Toggle on or off to display discrete data points (as circles) or display lines only.

**To change fleet capacity**

• Below the history graph are tools for setting capacity limits and scaling capacity. See **Manually Set Fleet Capacity (p. 62)** for details on how to use these tools. See **Auto-Scale Fleet Capacity (p. 64)** for information about managing auto-scaling.

**Game sessions**

The **Game sessions** tab lists past and present game sessions hosted on the fleet, including some detail information. Click a game session ID to access additional game session information, including player sessions.

• **Status** – Game session status. Valid statuses are:
  • **Activating** – A game session has been initiated and is preparing to run.
  • **Active** – A game session is running and available to receive players (depending on the session’s player creation policy).
  • **Terminated** – Game session has ended.
• **Name** – Game generated for the game session.
• **ID** – Unique identifier assigned by Amazon GameLift to the game session.
• **IP address** – IP address specified for the game session.
• **Port** – Port number used to connect to the game session.
• **Player sessions** – Number of players connected to the game sessions along with total possible players the game session can support. For example: 2 (connected players) of 10 (possible players) means the fleet can support 8 additional players.
• **Uptime** – Total length of time the game session has been running.
• **Date created** – Date and time stamp indicating when the fleet was created.

**Build**

The **Build** tab displays the fleet's build-related configuration, which was set when the fleet was created. Select the build ID to see the full **build** detail page.

If your build has been deleted or an error has occurred while attempting to retrieve your build, you may see one of the following status messages:

• **Deleted** – The build for this fleet was deleted. Your fleet will still run properly despite the build having been deleted.
• **Error** – An error occurred while attempting to retrieve build information for the fleet.

**Capacity allocation**

The **Capacity allocation** tab displays the runtime configuration for the fleet, which specifies what server processes to launch on each instance and how. It includes the path for the game server executable and
optional launch parameters. You can change the fleet's capacity allocation either by editing the fleet in the console or by using the AWS CLI to update the runtime configuration.

Ports

The Ports tab displays the fleet's connection permissions, including IP address and port setting ranges. You can change connection permissions by either editing the fleet in the console or using the AWS CLI to update the fleet's port settings.

ARN

The ARN tab lists the Amazon Resource Name (ARN) assigned to this fleet.

View Data on Game and Player Sessions

You can view information about the games running on a fleet and as well as individual players. For more information about game sessions and player sessions, see How Players Connect to Games (p. 8).

To view game session data

1. In the Amazon GameLift console, open the detail page for the fleet you want to study. (Choose Fleets in the menu bar and click on a fleet name.)
2. Open the Game sessions tab. This tab lists all game sessions hosted on the fleet along with summary information.
3. Click a game session to view additional information about the game session as well as a list of players that were connected to the game.

Game sessions

A summary of your game session information is displayed at the top of the page and includes:

- **Status** – The status of the game session. Valid statuses include:
  - **Activating** – Amazon GameLift has created a game session and passed your game properties to the game server process. The game server interprets the game properties and calls back to Amazon GameLift when it is ready for potential player sessions to connect.
  - **Active** – The game session can support game play with zero or more player sessions connected to it.
  - **Terminated** – The game session has ended, and player sessions are no longer permitted to connect to the terminated game session.
- **Name** – The name automatically generated for the game session.
- **IP address** – For game sessions with a status of Activating or Active, the IP address used to connect to the game.
- **Port** – The port number used to connect to the game session.
- **Player sessions** – The number of players currently connected to the game session along with the total number of player slots in the game session. For example, the value **10 of 15** indicates that of the 15 available slots in the game, 10 are filled and 5 are open.
- **Player session creation policy** – The policy that determines whether new players can connect to the game. Values are **Accept all** or **Deny all**. For more information, see the GameSession object.
- **Uptime** – The total length of time the game session has been running.
- **Date created** – The date and time the game session was created.
Player sessions

The following player session data is collected for each game session:

- **Status** – The status of the player session. Options include:
  - **Reserved** – Player session has been reserved, but the player has not yet connected.
  - **Active** – Player session is currently connected to the game server.
  - **Completed** – Player session has ended; player is no longer connected.
  - **Timed Out** – Player session was reserved, but the player failed to connect.
- **ID** – The identifier assigned to the player session.
- **Player ID** – A unique identifier for the player. Click this ID to get additional player information.
- **Start time** – The time the player connected to the game session.
- **End time** – The time the player disconnected from the game session.
- **Total time** – The total length of time the player has been active in the player session.

Player information

View additional information for a selected player, including a list of all games the player connected to across all fleets in the current region. This information includes the status, start and end times, and total connected time for each player session. You can click to view data for the relevant game sessions and fleets.

View Your Aliases

You can view information on all of the fleet aliases you have created and take actions on them on the Aliases page. Aliases shown include only those created for the selected region.

Alias Catalog

All created aliases are shown on the Aliases catalog page. To view the Aliases page, choose **Aliases** from the Amazon GameLift console's menu bar.

The Aliases page provides summary information on all builds, including type. From this page you can:

- Create a new alias. Click **Create alias**.
- Filter and sort the aliases list. Use the controls at the top of the table.
- View alias details. Click an alias name to open the alias detail page.

Alias Detail

Access an alias's detail page from either the console dashboard or the Aliases catalog page by clicking the alias name. The Alias detail page displays a summary of information on the alias.

From this page you can:

- Edit an alias, including changing the name, description, and the fleet ID the alias is associated with. Click **Actions: Edit alias**.
- View information on the fleet the alias is currently associated with. This includes the fleet's status and current utilization (active game sessions and players).
• Delete an alias. Click Actions: Delete alias.

Alias detail information includes:

• Type – The routing option for the alias, which can be one of these:
  • Simple – A simple alias routes a player to games on an associated fleet. You can update the alias to point to a different fleet at any time.
  • Terminal – A terminal alias does not point to a fleet. Instead it passes a message back to the client. This alias type is useful for gracefully notifying players when a set of game servers is no longer available. For example, a terminal alias might inform players that their game clients are out of date and provide upgrade options.

• Alias ID – The unique number used to identify the alias.
• Description – The description of the alias.
• Date created – The date and time the alias was created.
Monitoring Amazon GameLift

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon GameLift and your other AWS solutions. There are three primary uses for metrics with Amazon GameLift: 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, report when something is wrong, and take automatic actions when appropriate:

- Amazon GameLift Console
- Amazon CloudWatch — You can monitor Amazon GameLift metrics in real time, as well as metrics for other AWS resources and applications that you’re running on AWS services. 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 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.

Topics

- Monitor Amazon GameLift with Amazon CloudWatch (p. 125)
- Logging Amazon GameLift API Calls with AWS CloudTrail (p. 134)

Monitor Amazon GameLift with Amazon CloudWatch

You can monitor Amazon GameLift 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, so that you can access historical information and gain a better perspective on how your game server hosting with Amazon GameLift is performing. You can also 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.

To access CloudWatch metrics for your games, you can use the AWS Management Console, the AWS CLI, or the CloudWatch API. The following tables list the metrics and dimensions for Amazon GameLift. All metrics that are available in CloudWatch are also available in the Amazon GameLift console, which provides the data as a set of customizable graphs.

Amazon GameLift Metrics for Fleets

The AWS/GameLift namespace includes the following metrics related to activity across a fleet or a group of fleets. The Amazon GameLift service sends metrics to CloudWatch every minute.

Instances

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveInstances</td>
<td>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</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>sessions. This metric measures current total instance capacity. This metric can be used with automatic scaling.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td>DesiredInstances</td>
<td>Target number of active instances that Amazon GameLift 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.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td>IdleInstances</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td>MaxInstances</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td>MinInstances</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PercentIdleInstances</td>
<td>Percentage of all active instances that are idle (calculated as ( \frac{\text{IdleInstances}}{\text{ActiveInstances}} )). This metric can be used for automatic scaling.</td>
</tr>
<tr>
<td></td>
<td>Units: Percent</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td>InstanceInterruptions</td>
<td>Number of spot instances that have been interrupted.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum</td>
</tr>
</tbody>
</table>

**Server Processes**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveServerProcesses</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td>HealthyServerProcesses</td>
<td>Active server processes that are reporting healthy. This metric is useful for tracking the overall health of the fleet's game servers.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td>PercentHealthyServerProcesses</td>
<td>Percentage of all active server processes that are reporting healthy (calculated as ( \frac{\text{HealthyServerProcesses}}{\text{ActiveServerProcesses}} )).</td>
</tr>
<tr>
<td></td>
<td>Units: Percent</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td>ServerProcessAbnormalTerminations</td>
<td>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 service. This occurs when a server process</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>stops responding, consistently reports failed health checks, or does not terminate cleanly (by calling ProcessEnding()).</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum</td>
</tr>
<tr>
<td>ServerProcessActivations</td>
<td>Server processes that successfully transitioned from ACTIVATING to ACTIVE status since the last report. Server processes cannot host game sessions until they are active.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum</td>
</tr>
<tr>
<td>ServerProcessTerminations</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum</td>
</tr>
</tbody>
</table>

**Game Sessions**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActivatingGameSessions</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td>ActiveGameSessions</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
</tbody>
</table>
### Amazon GameLift Metrics for Fleets

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AvailableGameSessions</strong></td>
<td>Game session slots on active, healthy server processes that are not currently being used. This metric measures the number of new game sessions that could be started immediately. This metric can be used with automatic scaling. Units: Count</td>
</tr>
<tr>
<td><strong>PercentAvailableGameSessions</strong></td>
<td>Percentage of game session slots on all active server processes (healthy or unhealthy) that are not currently being used (calculated as AvailableGameSessions / [ActiveGameSessions + AvailableGameSessions + unhealthy server processes]). This metric can be used with automatic scaling. Units: Percent</td>
</tr>
<tr>
<td><strong>GameSessionInterruptions</strong></td>
<td>Number of game sessions on spot instances that have been interrupted. Units: Count</td>
</tr>
</tbody>
</table>

### Player Sessions

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CurrentPlayerSessions</strong></td>
<td>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</td>
</tr>
<tr>
<td><strong>PlayerSessionActivations</strong></td>
<td>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</td>
</tr>
</tbody>
</table>
### Amazon GameLift Metrics for Queues

The `GameLift` namespace includes the following metrics related to activity across a game session placement queue. The Amazon GameLift service sends metrics to CloudWatch every minute.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AverageWaitTime</strong></td>
<td>Average amount of time that game session placement requests in the queue with status PENDING have been waiting to be fulfilled.</td>
</tr>
<tr>
<td></td>
<td>Units: Seconds</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td><strong>PlacementsCanceled</strong></td>
<td>Game session placement requests that were canceled before timing out since the last report.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td><strong>PlacementsStarted</strong></td>
<td>New game session placement requests that were added to the queue since the last report.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td><strong>PlacementsSucceeded</strong></td>
<td>Game session placement requests that resulted in a new game session since the last report.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td><strong>PlacementsTimedOut</strong></td>
<td>Game session placement requests that reached the queue’s timeout limit without being fulfilled since the last report.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td><strong>QueueDepth</strong></td>
<td>Number of game session placement requests in the queue with status PENDING.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
</tbody>
</table>

Version 130
Amazon GameLift Metrics for Matchmaking

The GameLift namespace includes the following metrics related to matchmaking activity. The Amazon GameLift service sends metrics to CloudWatch every minute.

For more information on the sequence of matchmaking activity, see How Amazon GameLift FlexMatch Works.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurrentTickets</td>
<td>Matchmaking requests currently being processed or waiting to be processed.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td>MatchAcceptancesTimedOut</td>
<td>For matchmaking configurations that require acceptance, the potential matches that timed out during acceptance since the last report.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Sum</td>
</tr>
<tr>
<td>MatchesAccepted</td>
<td>For matchmaking configurations that require acceptance, the potential matches that were accepted since the last report.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Sum</td>
</tr>
<tr>
<td>MatchesCreated</td>
<td>Potential matches that were created since the last report.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Sum</td>
</tr>
<tr>
<td>MatchesPlaced</td>
<td>Matches that were successfully placed into a game session since the last report.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Sum</td>
</tr>
<tr>
<td>MatchesRejected</td>
<td>For matchmaking configurations that require acceptance, the potential matches that were rejected by at least one player since the last report.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| PlayersStarted         | Players in matchmaking tickets that were added since the last report.  
Units: Count  
Relevant CloudWatch statistics: Sum                                                                                                                                                                                                                                                                                                                                                                         |
| RuleEvaluationsPassed  | Rule evaluations during the matchmaking process that passed since the last report. This metric is limited to the top 50 rules.  
Units: Count  
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                                                                                                                                                                                                                                                                                                                                                                         |
| 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                                                                                                                                                                                                                                                                                                                                                                         |
| 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, p99                                                                                                                                                                                                                                                                                                                                                       |
<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimeToTicketCancel</td>
<td>For matchmaking requests that were canceled before the last report, the amount of time between ticket creation and cancellation.</td>
</tr>
<tr>
<td></td>
<td>Units: Seconds</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Data Samples, Average, Minimum, Maximum, p99</td>
</tr>
<tr>
<td>TimeToTicketSuccess</td>
<td>For matchmaking requests that succeeded before the last report, the amount of time between ticket creation and successful match placement.</td>
</tr>
<tr>
<td></td>
<td>Units: Seconds</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Data Samples, Average, Minimum, Maximum, p99</td>
</tr>
</tbody>
</table>

### Dimensions for Amazon GameLift Metrics

Amazon GameLift supports filtering metrics by the following dimensions.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FleetId</td>
<td>Unique identifier for a single fleet. This dimension is used with all metrics for instances, server processes, game sessions, and player sessions. It is not used with metrics for queues and matchmaking.</td>
</tr>
<tr>
<td>FleetMetricsGroup</td>
<td>Unique identifier for a collection of fleets. A fleet is included in a fleet metric group by adding the metric group name to the fleet's attributes (see UpdateFleetAttributes()). This dimension is used with all metrics for instances, server processes, game sessions, and player sessions. It is not used with metrics for queues and matchmaking.</td>
</tr>
<tr>
<td>QueueName</td>
<td>Unique identifier for a single queue. This dimension is used with metrics for game session placement queues only.</td>
</tr>
<tr>
<td>MatchmakingConfigurationName</td>
<td>Unique identifier for a single matchmaking configuration. This dimension is used with metrics for matchmaking only.</td>
</tr>
<tr>
<td>MatchmakingConfigurationName-RuleName</td>
<td>Unique identifier for the intersect of a matchmaking configuration and a matchmaking rule. This dimension is used with metrics for matchmaking only.</td>
</tr>
<tr>
<td>InstanceType</td>
<td>Unique identifier for an EC2 instance type designation, such as &quot;C4.large&quot;. This dimension is used with metrics for spot instances only.</td>
</tr>
</tbody>
</table>
### Logging Amazon GameLift API Calls

Amazon GameLift is integrated with AWS CloudTrail, a service that captures all of the API calls made by or on behalf of Amazon GameLift in your AWS account. CloudTrail delivers the log files to an Amazon S3 bucket that you specify. CloudTrail captures API calls from the Amazon GameLift console or from the Amazon GameLift API. Using the information collected by CloudTrail, you can determine what request was made to Amazon GameLift, the source IP address from which the request was made, who made the request, when it was made, and so on. To learn more about CloudTrail, including how to configure and enable it, see the [AWS CloudTrail User Guide](https://docs.aws.amazon.com/AmazonCloudTrail/latest/userguide/).

#### Amazon GameLift Information in CloudTrail

When CloudTrail logging is enabled in your AWS account, API calls made to Amazon GameLift actions are tracked in log files. Amazon GameLift records are written together with other AWS service records in a log file. CloudTrail determines when to create and write to a new file based on a time period and file size.

All Amazon GameLift actions are logged by CloudTrail. For example, calls to `CreateGameSession`, `CreatePlayerSession` and `UpdateGameSession` generate entries in the CloudTrail log files. For the complete list of actions, see the [Amazon GameLift API Reference](https://docs.aws.amazon.com/gamelift/latest/developerguide/).

Every log entry contains information about who generated the request. The user identity information in the log helps you determine whether the request was made with AWS account root or IAM user credentials, with temporary security credentials for a role or federated user, or by another AWS service. For more information, see the `userIdentity` field in the CloudTrail Event Reference.

You can store your log files in your S3 bucket for as long as you want, but you can also define Amazon S3 lifecycle rules to archive or delete log files automatically. By default, your log files are encrypted with Amazon S3 server-side encryption (SSE).

If you want to take quick action upon log file delivery, you can choose to have CloudTrail publish Amazon Simple Notification Service (Amazon SNS) notifications when new log files are delivered. For more information, see Configuring Amazon SNS Notifications.

You can also aggregate Amazon GameLift log files from multiple AWS regions and multiple AWS accounts into a single S3 bucket. For more information, see Aggregating CloudTrail Log Files to a Single Amazon S3 Bucket.

### Understanding Amazon GameLift Log File Entries

CloudTrail log files can contain one or more log entries where each entry is made up of multiple JSON-formatted events. A log entry represents a single request from any source and includes information about the requested action, any parameters, the date and time of the action, and so on. The log entries are not guaranteed to be in any particular order. That is, they are not an ordered stack trace of the public API calls.

The following example shows a CloudTrail log entry that demonstrates the `CreateFleet` and `DescribeFleetAttributes` actions.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OperatingSystem</td>
<td>Unique identifier for the operating system of an instance. This dimension is used with metrics for spot instances only.</td>
</tr>
</tbody>
</table>
{ "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": "c4.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": "c8f6e01-fbf9-4c4e-a0fe-ad7dc205cel1",
  "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"
  },
  "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": "c4.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": "c8f6e01-fbf9-4c4e-a0fe-ad7dc205cel1",
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}]}
"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-3a63c0bad86f",
"eventType": "AwsApiCall",
"recipientAccountId": "111122223333"
],
}
Amazon GameLift Reference Guides

This section contains the set of reference guides available for using Amazon GameLift.

Topics
- Amazon GameLift Server API (C++) Reference (p. 137)
- Amazon GameLift Server API (C#) Reference (p. 152)
- Amazon GameLift Server API Reference for Unreal Engine (p. 166)
- FlexMatch Rules Reference (p. 172)
- FlexMatch Events Reference (p. 177)

Amazon GameLift Server API (C++) Reference

This Amazon GameLift C++ Server API reference can help you prepare your multiplayer game for use with Amazon GameLift. For details on the integration process, see Add Amazon GameLift to Your Game Server (p. 30).

This API is defined in GameLiftServerAPI.h, LogParameters.h, and ProcessParameters.h.

- Actions (p. 137)
- Data Types (p. 148)

Amazon GameLift Server API (C++) Reference: Actions

This Amazon GameLift C++ Server API reference can help you prepare your multiplayer game for use with Amazon GameLift. For details on the integration process, see Add Amazon GameLift to Your Game Server (p. 30).

This API is defined in GameLiftServerAPI.h, LogParameters.h, ProcessParameters.h.

- Actions
- Data Types (p. 148)

AcceptPlayerSession()

Notifies the Amazon GameLift service that a player with the specified player session ID has connected to the server process and needs validation. Amazon GameLift 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 changes the status of the player slot from RESERVED to ACTIVE.

Syntax

```cpp
GenericOutcome AcceptPlayerSession(const std::string& playerSessionId);
```
Parameters

playerSessionId

Unique ID issued by the Amazon GameLift service in response to a call to the AWS SDK Amazon GameLift 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.

```cpp
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 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 (p. 148) 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 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 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::GetNameForPlayerSessionStatus(Aws::GameLift::Server::Model::PlayerSessionStatus::Active));
request.SetLimit(10);
request.SetGameSessionId("the game session ID");    // can use GetGameSessionId()

// Call DescribePlayerSessions
Aws::GameLift::DescribePlayerSessionsOutcome playerSessionsOutcome =
    Aws::GameLift::Server::Model::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>.

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**

```cpp
Aws::GameLift::AwsStringOutcome sessionIdOutcome = 
Awss::GameLift::Server::GetGameSessionId();
```

**GetSdkVersion()**

Returns the current version number of the SDK in use.

**Syntax**

```cpp
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**

```cpp
Aws::GameLift::AwsStringOutcome SdkVersionOutcome = 
Awss::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 service. A server process may be shut down for several reasons: (1) process poor health, (2) when an instance is being terminated during a scale-down event, or (3) when an instance is being terminated due to a spot-instance interruption (p. 69).

If the process has received an `onProcessTerminate()` callback, the value returned is the estimated termination time in epoch seconds. If no termination time is available, the value returned is -1, which indicates that the server process may be terminated at any time. If the process has not received an `onProcessTerminate()` callback, the returned value will always be -1. Learn more about shutting down a server process (p. 32).

**Syntax**

```cpp
AwsLongOutcome GetTerminationTime();
```
Parameters

This action has no parameters.

Return Value

If successful, returns the current SDK version as an AwsLongOutcome object. The value is either the termination time in epoch seconds, or the value -1. If not successful, returns an error message.

Example

```cpp
Aws::GameLift::AwsLongOutcome TermTimeOutcome = 
    Aws::GameLift::Server::GetTerminationTime();
```

InitSDK()

Initializes the Amazon GameLift SDK. This method should be called on launch, before any other Amazon GameLift-related initialization occurs.

Syntax

```cpp
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() (p. 142).

Example

```cpp
Aws::GameLift::Server::InitSDKOutcome initOutcome = 
    Aws::GameLift::Server::InitSDK();
```

ProcessEnding()

Notifies the Amazon GameLift service that the server process is shutting down. The call triggers Amazon GameLift to change the instance's status from ACTIVE to TERMINATING. This method should be called after all other cleanup tasks, including shutting down all active game sessions. Amazon GameLift waits at least 30 seconds after receiving this call before terminating the instance. 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

```cpp
GenericOutcome ProcessEnding();
```

Parameters

This action has no parameters.
**Return Value**

Returns a generic outcome consisting of success or failure with an error message.

**Example**

```cpp
Aws::GameLift::GenericOutcome outcome = Aws::GameLift::Server::ProcessEnding();
```

**ProcessReady()**

Notifies the Amazon GameLift service that the server process is ready to host game sessions. This method should be called after successfully invoking `InitSDK()` (p. 141) and completing any setup tasks required before the server process can host a game session.

This call is synchronous. To make an asynchronous call, use `ProcessReadyAsync()` (p. 143). See Prepare a Server Process (p. 30) for more details.

**Syntax**

```cpp
GenericOutcome ProcessReady(  
    const Aws::GameLift::Server::ProcessParameters &processParameters);
```

**Parameters**

*processParameters*

A `ProcessParameters` (p. 150) object communicating the following information about the server process:

- Names of callback methods, implemented in the game server code, that the Amazon GameLift 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 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()` (p. 142) call and callback function implementations.

```cpp
// 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),  
);  
```

**Version**

142
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 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 to call in certain circumstances. Game server code must implement these functions.

This call is asynchronous. To make a synchronous call, use ProcessReady() (p. 142). See Prepare a Server Process (p. 30) for more details.

**Syntax**

```
GenericOutcomeCallable ProcessReadyAsync(
   const Aws::GameLift::Server::ProcessParameters &processParameters);
```

**Parameters**

processParameters

A ProcessParameters (p. 150) object communicating the following information about the server process:

- Names of callback methods, implemented in the game server code, that the Amazon GameLift 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 to capture and store.

Required: Yes

**Return Value**

Returns a generic outcome consisting of success or failure with an error message.
Example

```cpp
// 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 service that a player with the specified player session ID has disconnected from the server process. In response, Amazon GameLift changes the player slot to available, which allows it to be assigned to a new player.

Syntax

```
GenericOutcome RemovePlayerSession(
    const std::string& playerSessionId);
```

Parameters

**playerSessionId**

Unique ID issued by the Amazon GameLift service in response to a call to the AWS SDK Amazon GameLift 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

```cpp
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 in Backfill Existing Games with FlexMatch (p. 112).

This action is asynchronous. If new players are successfully matched, the Amazon GameLift 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() (p. 146) to cancel the original request.

Syntax

```cpp
StartMatchBackfillOutcome StartMatchBackfill (
    const Aws::GameLift::Server::Model::StartMatchBackfillRequest &startBackfillRequest);
```

Parameters

StartMatchBackfillRequest

A StartMatchBackfillRequest (p. 151) 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 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

```cpp
// 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::Server::Model::StartMatchBackfillOutcome backfillOutcome =
    Aws::GameLift::Server::Model::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() (p. 145). See also the AWS SDK action StopMatchmaking(). Learn more about the FlexMatch backfill feature in Backfill Existing Games with FlexMatch (p. 112).

Syntax

| GenericOutcome StopMatchBackfill (const Aws::GameLift::Server::Model::StopMatchBackfillRequest &stopBackfillRequest); |

Parameters

StopMatchBackfillRequest

A StopMatchBackfillRequest (p. 152) object identifying the matchmaking ticket to cancel:

- ticket ID assigned to the backfill request being cancelled
- 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::Model::StopMatchBackfillRequest(stopBackfillRequest);

**TerminateGameSession()**

Notifies the Amazon GameLift service that the server process has shut down the game session. Since each server process hosts only one game session at a time, there's no need to specify which session. This action should be called at the end of the game session shutdown process. After calling this action, the server process can call ProcessReady() (p. 142) to signal its availability to host a new game session. Alternatively it can call ProcessEnding() (p. 141) to shut down the server process and terminate the instance.

**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

Aws::GameLift::GenericOutcome outcome =
    Aws::GameLift::Server::TerminateGameSession();
outcome =
    Aws::GameLift::Server::ProcessReady(onStartGameSession, onProcessTerminate);
```

**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.

```cpp
Aws::GameLift::GenericOutcome outcome =
    Aws::GameLift::Server::UpdatePlayerSessionCreationPolicy(Aws::GameLift::Model::PlayerSessionCreationPolicy::ACCEPT_ALL);
```

**Amazon GameLift Server API (C++) Reference: Data Types**

This Amazon GameLift C++ Server API reference can help you prepare your multiplayer game for use with Amazon GameLift. For details on the integration process, see Add Amazon GameLift to Your Game Server (p. 30).

This API is defined in `GameLiftServerAPI.h`, `LogParameters.h`, and `ProcessParameters.h`.

- **Actions (p. 137)**
- **Data Types**

**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:
  ```text
  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.
Data Types

**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 (p. 37).

**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 to upload and store once the game session ends. This information is communicated to the Amazon GameLift service in a ProcessReady() (p. 142) call.
Contents

logPaths

Directory paths to game server log files that you want Amazon GameLift 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. 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 service in a `ProcessReady()` (p. 142) 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 (p. 149)
Required: No

onStartGameSession

Name of callback function that the Amazon GameLift service calls to activate a new game session. Amazon GameLift calls this function in response to the client request `CreateGameSession`. The callback function passes a `GameSession` object (defined in the Amazon GameLift Service API Reference).

Type: const std::function<void(Aws::GameLift::Model::GameSession)> onStartGameSession
Required: Yes

onProcessTerminate

Name of callback function that the Amazon GameLift service calls to force the server process to shut down. After calling this function, Amazon GameLift waits five minutes for the server process to shut down and respond with a `ProcessEnding()` (p. 141) call. If no response is receive, it shuts down the server process.
Type: `std::function<void()>` onProcessTerminate

Required: No

**onHealthCheck**

Name of callback function that the Amazon GameLift service calls to request a health status report from the server process. Amazon GameLift calls this function every 60 seconds. After calling this function Amazon GameLift 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 service calls to provide an updated game session object. Amazon GameLift calls this function once a match backfill (p. 112) request has been processed. 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 service in a `StartMatchBackfill()` (p. 145) call.

**Contents**

**GameSessionArn**

Unique game session identifier. The API action `GetGameSessionId()` (p. 139) 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 (p. 111).

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 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 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 service in a `StopMatchBackfill()` (p. 146) call.

**Contents**

**GameSessionArn**

Unique game session identifier associated with the request being cancelled.

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 cancelled.

Type: String

Required: Yes

### Amazon GameLift Server API (C#) Reference

This Amazon GameLift C# Server API reference can help you prepare your multiplayer game for use with Amazon GameLift. For details on the integration process, see Add Amazon GameLift to Your Game Server (p. 30).

This API is defined in `GameLiftServerAPI.cs, LogParameters.cs, and ProcessParameters.cs`.

- Actions (p. 153)
Amazon GameLift Server API (C#) Reference: Actions

This Amazon GameLift C# Server API reference can help you prepare your multiplayer game for use with Amazon GameLift. For details on the integration process, see Add Amazon GameLift to Your Game Server (p. 30).

This API is defined in GameLiftServerAPI.cs, LogParameters.cs, and ProcessParameters.cs.

- Actions
- Data Types (p. 162)

AcceptPlayerSession()

Notifies the Amazon GameLift service that a player with the specified player session ID has connected to the server process and needs validation. Amazon GameLift 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 changes the status of the player slot from RESERVED to ACTIVE.

Syntax

```
GenericOutcome AcceptPlayerSession(String playerSessionId)
```

Parameters

`playerSessionId`

Unique ID issued by the Amazon GameLift service in response to a call to the AWS SDK Amazon GameLift API action `CreatePlayerSession`. The game client references this ID when connecting to the server process.

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 playerSessionId){
    var acceptPlayerSessionOutcome = GameLiftServerAPI.AcceptPlayerSession(playerSessionId);
    if(acceptPlayerSessionOutcome.Success)
    {
        connectionToSessionMap.emplace(connection, playerSessionId);
        connection.Accept();
    }
}```
else
{
    connection.Reject(acceptPlayerSessionOutcome.Error.ErrorMessage);
}

### ActivateGameSession()

Notifies the Amazon GameLift 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**

```csharp
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.

```csharp
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**

```csharp
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 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 will return the first 10 player sessions records matching the request.

```javascript
// 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.

Syntax

```javascript
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

```javascript
var getGameSessionIdOutcome = GameLiftServerAPI.GetGameSessionId();
```

GetSdkVersion()

Returns the current version number of the SDK built into the server process.

Syntax

```javascript
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

```javascript
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 service. A server process may be shut down for several reasons: (1) process poor health, (2) when an instance is being terminated during a scale-down event, or (3) when an instance is being terminated due to a spot-instance interruption (p. 69).

If the process has received an onProcessTerminate() callback, the value returned is the estimated termination time in epoch seconds. If no termination time is available, the value returned is -1, which indicates that the server process may be terminated at any time. If the process has not received an onProcessTerminate() callback, the returned value will always be -1. Learn more about shutting down a server process (p. 32).

Syntax

```javascript
AwsLongOutcome GetTerminationTime()
```

Parameters

This action has no parameters.

Return Value

If successful, returns the current SDK version as an AwsLongOutcome object. The value is either the termination time in epoch seconds, or the value -1. If not successful, returns an error message.

Example

```javascript
var getTerminationTimeOutcome = GameLiftServerAPI.GetTerminationTime();
```

InitSDK()

Initializes the Amazon GameLift SDK. This method should be called on launch, before any other Amazon GameLift-related initialization occurs.

Syntax

```javascript
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() (p. 157).

Example

```javascript
var initSDKOutcome = GameLiftServerAPI.InitSDK();
```

ProcessEnding()

Notifies the Amazon GameLift service that the server process is shutting down. The call triggers Amazon GameLift to change the instance's status from ACTIVE to TERMINATING. This method should be called after all other cleanup tasks, including shutting down all active game sessions. Amazon GameLift waits at least 30 seconds after receiving this call before terminating the instance. 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

```javascript
GenericOutcome ProcessEnding()
```

Parameters

This action has no parameters.

Return Value

Returns a generic outcome consisting of success or failure with an error message.

Example

```javascript
var processEndingOutcome = GameLiftServerAPI.ProcessEnding();
```

ProcessReady()

Notifies the Amazon GameLift service that the server process is ready to host game sessions. This method should be called after successfully invoking InitSDK() (p. 156) and completing any setup tasks required before the server process can host a game session.

Syntax

```javascript
GenericOutcome ProcessReady(ProcessParameters processParameters)
```

Parameters

processParameters

A ProcessParameters (p. 164) object communicating the following information about the server process:
• Names of callback methods, implemented in the game server code, that the Amazon GameLift 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 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()` (p. 157) call and delegate function implementations.

```csharp
// 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
    {
        "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();
    GameLiftServerAPI.Destroy();
}

bool OnHealthCheck()
{
    bool isHealthy;
    // complete health evaluation within 60 seconds and set health
    return isHealthy;
}

**RemovePlayerSession()**

Notifies the Amazon GameLift service that a player with the specified player session ID has disconnected from the server process. In response, Amazon GameLift changes the player slot to available, which allows it to be assigned to a new player.
Syntax

| GenericOutcome RemovePlayerSession(String playerSessionId) |

Parameters

playerSessionId

Unique ID issued by the Amazon GameLift service in response to a call to the AWS SDK Amazon GameLift API action CreatePlayerSession. The game client references this ID when connecting to the server process.

Type: String

Required: Yes

Return Value

Returns a generic outcome consisting of success or failure with an error message.

Example

```java
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 in Backfill Existing Games with FlexMatch (p. 112).

This action is asynchronous. If new players are successfully matched, the Amazon GameLift 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() (p. 160) to cancel the original request.

Syntax

```java
StartMatchBackfillOutcome StartMatchBackfill (StartMatchBackfillRequest startBackfillRequest);
```

Parameters

StartMatchBackfillRequest

A StartMatchBackfillRequest (p. 165) 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 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

```java
// 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() (p. 159). See also the AWS SDK action StopMatchmaking(). Learn more about the FlexMatch backfill feature in Backfill Existing Games with FlexMatch (p. 112).

Syntax

```java
GenericOutcome StopMatchBackfill (StopMatchBackfillRequest stopBackfillRequest);
```

Parameters

StopMatchBackfillRequest

A StopMatchBackfillRequest (p. 166) object identifying the matchmaking ticket to cancel:
- ticket ID assigned to the backfill request being cancelled
- 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

```javascript
// 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
    GameSessionId = GameLiftServerAPI.GetGameSessionId().Result //gets the ID for the
};
var stopBackfillOutcome = GameLiftServerAPI.StopMatchBackfillRequest(stopBackfillRequest);
```

TerminateGameSession()

Notifies the Amazon GameLift service that the server process has shut down the game session.
(Currently, each server process hosts only one game session at a time, so there's no need to specify which
session.) This action should be called at the end of the game session shutdown process. After calling this
action, the server process can call `ProcessReady()` (p. 157) to signal its availability to host a new game
session. Alternatively, it can call `ProcessEnding()` (p. 157) to shut down the server process and terminate
the instance.

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.

```javascript
// 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
either to accept or deny all new player sessions. (See also the `UpdateGameSession()` action in the Amazon
GameLift Service API Reference).
Syntax

```c
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.

```c
var updatePlayerSessionCreationPolicyOutcome = 
    GameLiftServerAPI.UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy.ACCEPT_ALL);
```

Amazon GameLift Server API (C#) Reference: Data Types

This Amazon GameLift C# Server API reference can help you prepare your multiplayer game for use with Amazon GameLift. For details on the integration process, see Add Amazon GameLift to Your Game Server (p. 30).

This API is defined in `GameLiftServerAPI.h`, `LogParameters.h`, and `ProcessParameters.h`.

- Actions (p. 153)
- Data Types

LogParameters

This data type is used to identify which files generated during a game session that you want Amazon GameLift to upload and store once the game session ends. This information is communicated to the Amazon GameLift service in a `ProcessReady()` (p. 157) call.

Contents

logPaths

List of directory paths to game server log files you want Amazon GameLift to store for future access. These files are generated by a server process during each game session; file paths and names
Data Types

are defined in your game server and stored in the root game build directory. 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 NextToken to get results as a set of
sequential pages. 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 (p. 37).

Type: String

Required: No

PlayerSessionId

Unique identifier for a player session.
Data Types

**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).

**ProcessParameters**

This data type contains the set of parameters sent to the Amazon GameLift service in a `ProcessReady()` (p. 157) 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` (p. 162)

Required: Yes

**onStartGameSession**

Name of callback function that the Amazon GameLift service calls to activate a new game session. Amazon GameLift calls this function in response to the client request `CreateGameSession`. The callback function takes a `GameSession` object (defined in the `Amazon GameLift Service API Reference`).

Type: `void OnStartGameSessionDelegate(GameSession gameSession)`

Required: Yes

**onProcessTerminate**

Name of callback function that the Amazon GameLift service calls to force the server process to shut down. After calling this function, Amazon GameLift waits five minutes for the server process to shut down and respond with a `ProcessEnding()` (p. 157) call before it shuts down the server process.
Type: `void OnProcessTerminateDelegate()`

Required: Yes

`onHealthCheck` 

Name of callback function that the Amazon GameLift service calls to request a health status report from the server process. Amazon GameLift calls this function every 60 seconds. After calling this function Amazon GameLift waits 60 seconds for a response, and if none is received, records the server process as unhealthy.

Type: `bool OnHealthCheckDelegate()`

Required: Yes

`onHealthCheck` 

Name of callback function that the Amazon GameLift service calls to request a health status report from the server process. Amazon GameLift calls this function every 60 seconds. After calling this function Amazon GameLift waits 60 seconds for a response, and if none is received, records the server process as unhealthy.

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 service in a `StartMatchBackfill()` (p. 159) call.

**Contents**

**GameSessionArn**

Unique game session identifier. The API action `GetGameSessionId()` (p. 155) 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 (p. 111).

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 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 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 service in a StopMatchBackfill() (p. 160) call.

Contents

GameSessionArn
Unique game session identifier associated with the request being cancelled.

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 cancelled.

Type: String
Required: Yes

Amazon GameLift Server API Reference for Unreal Engine

This Amazon GameLift Server API reference can help you prepare your Unreal Engine game projects for use with Amazon GameLift. For details on the integration process, see Add Amazon GameLift to Your Game Server (p. 30).

This API is defined in GameLiftServerSDK.h and GameLiftServerSDKModels.h.
To set up the Unreal Engine plugin and see code examples Add Amazon GameLift to an Unreal Engine Game Server Project (p. 24).

- Actions (p. 167)
- Data Types (p. 171)

Amazon GameLift Server API Reference for Unreal Engine: Actions

This Amazon GameLift Server API reference can help you prepare your Unreal Engine game projects for use with Amazon GameLift. For details on the integration process, see Add Amazon GameLift to Your Game Server (p. 30).

This API is defined in GameLiftServerSDK.h and GameLiftServerSDKModels.h.

To set up the Unreal Engine plugin and see code examples Add Amazon GameLift to an Unreal Engine Game Server Project (p. 24).

- Actions
- Data Types (p. 171)

**AcceptPlayerSession()**

Notifies the Amazon GameLift service that a player with the specified player session ID has connected to the server process and needs validation. Amazon GameLift 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 changes the status of the player slot from RESERVED to ACTIVE.

**Syntax**

```
FGameLiftGenericOutcome AcceptPlayerSession(const FString& playerSessionId)
```

**Parameters**

**playerSessionId**

Unique ID issued by the Amazon GameLift service in response to a call to the AWS SDK Amazon GameLift 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 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.

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.

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 SDK. This method should be called on launch, before any other Amazon GameLift-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 service that the server process is shutting down. The call triggers Amazon GameLift to change the instance's status from ACTIVE to TERMINATING. This method should be called after all other cleanup tasks, including shutting down all active game sessions. Amazon GameLift waits at least 30 seconds after receiving this call before terminating the instance. 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 service that the server process is ready to host game sessions. This method should be called after successfully invoking InitSDK() (p. 169) and completing any setup tasks required before the server process can host a game session.

Syntax

FGameLiftGenericOutcome ProcessReady(FProcessParameters &processParameters)

Parameters

FProcessParameters

A FProcessParameters (p. 171) object communicating the following information about the server process:
• Names of callback methods, implemented in the game server code, that the Amazon GameLift 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 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 (p. 25).

RemovePlayerSession()
Notifies the Amazon GameLift service that a player with the specified player session ID has disconnected from the server process. In response, Amazon GameLift changes the player slot to available, which allows it to be assigned to a new player.

Syntax

FGameLiftGenericOutcome RemovePlayerSession(const FString& playerSessionId)

Parameters
playerSessionId
Unique ID issued by the Amazon GameLift service in response to a call to the AWS SDK Amazon GameLift 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.

TerminateGameSession()
Notifies the Amazon GameLift service that the server process has shut down the game session. (Currently, each server process hosts only one game session at a time, so there's no need to specify which session.) This action should be called at the end of the game session shutdown process. After calling this action, the server process can call ProcessReady() (p. 169) to signal its availability to host a new game session. Alternatively, it can call ProcessEnding() (p. 169) to shut down the server process and terminate the instance.

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 Service API Reference).

**Syntax**

```cpp
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.

---

**Amazon GameLift Server API Reference for Unreal Engine: Data Types**

This Amazon GameLift Server API reference can help you prepare your Unreal Engine game projects for use with Amazon GameLift. For details on the integration process, see Add Amazon GameLift to Your Game Server (p. 30).

This API is defined in `GameLiftServerSDK.h` and `GameLiftServerSDKModels.h`.

To set up the Unreal Engine plugin and see code examples Add Amazon GameLift to an Unreal Engine Game Server Project (p. 24).

- Actions (p. 167)
- Data Types

**FProcessParameters**

This data type contains the set of parameters sent to the Amazon GameLift service in a `ProcessReady()` (p. 169) 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 service calls to activate a new game session. Amazon GameLift calls this function in response to the client request CreateGameSession. The callback function takes a GameSession object (defined in the Amazon GameLift Service API Reference).

Type: FOnStartGameSession

Required: Yes

onProcessTerminate

Name of callback function that the Amazon GameLift service calls to force the server process to shut down. After calling this function, Amazon GameLift waits five minutes for the server process to shut down and respond with a ProcessEnding() (p. 169) call before it shuts down the server process.

Type: FSimpleDelegate

Required: No

onHealthCheck

Name of callback function that the Amazon GameLift service calls to request a health status report from the server process. Amazon GameLift calls this function every 60 seconds. After calling this function Amazon GameLift waits 60 seconds for a response, and if none is received, records the server process as unhealthy.

Type: FOnHealthCheck

Required: No

FlexMatch Rules Reference

Use the following property expressions when writing rules for FlexMatch rule sets.

Learn more about creating FlexMatch rules:

• Build a FlexMatch Rule Set (p. 93)
Rule Types

The following rule types are allowed. Each rule type requires a set of properties as defined here.

Distance rule (distance)

Distance rules measure the difference between two number values, such as the distance between skill levels. For example, a distance rule might require that all players be within two levels of each other.

- **measurement** – Player attribute value to measure distance for; must be a number value.
- **referenceValue** – Number value to measure distance against for a prospective match.
- **minDistance/maxDistance** – Maximum or minimum distance value allowed for a successful match.
- **partyAggregation** – How to handle multiple-player (party) requests. Valid options are to use the minimum (min), maximum (max), or average (avg) values for a request's players. Default is avg.

Comparison rule (comparison)

Comparison rules compare a player attribute value against another value. There are two types of comparison rules. The first type compares an attribute value to a reference value. Specify the reference value and any valid comparison operation. For example, the rule might require that matched players have skill level 24 or greater. The second type compares an attribute value across all players in a team or match. This type omits the reference value and specifies either equals or not-equals (all or none of the players have the same attribute value). For example, the rule might require that all players chose the same game map.

- **measurement** – Player attribute value to compare.
- **referenceValue** – Value to evaluate the measurement against for a prospective match.
- **operation** – How to evaluate the measurement. Valid operations include: <, <=, =, !=, >, >=.
- **partyAggregation** – How to sort multiple-player (party) requests. Valid options are to use the minimum (min), maximum (max), or average (avg) values for a request's players. Default is avg.

Collection rule (collection)

Collection rules evaluate collections of player attribute values. A collection might contain attribute values for multiple players, a player attribute that is a collection (a string list), or both. For example, a collection rule might look at the collection of characters chosen by a team's players and require that it contain at least one of a certain character.

- **measurement** – Collection of player attribute values to evaluate. Attribute values must be string lists.
- **referenceValue** – Value or collection of values to evaluate the measurement against for a prospective match.
- **operation** – How to evaluate a measurement collection. Valid operations include:
  - **intersection** measures the number of values that are common in all players' collections. See the MapOverlap rule in Example 4: Use Explicit Sorting to Find Best Matches (p. 100).
  - **contains** measures the number of player attribute collections that contain a certain reference value. See the OverallMedicLimit rule in Example 3: Set Team Requirements and Latency Limits (p. 98).
  - **reference_intersection_count** measures the intersection between a player attribute collection and a reference value collection. Each player attribute collection (string list) in the
measurement is evaluated against the reference collection separately. This operation can be used to evaluate across different player attributes. See the OpponentMatch rule in Example 5: Find intersections across multiple player attributes (p. 102).

- **minCount/maxCount** – Maximum or minimum count value allowed for a successful match.
- **partyAggregation** – How to sort multiple-player (party) requests. Valid options are to use the union or intersection of values for a request's players. Default is union.

**Latency rule (latency)**

Latency rules evaluate player latency settings for acceptable matches. For example, a rule might require all matched players must have a regional latency under a maximum limit.

- **maxLatency** – Highest acceptable latency value for a region. For each player, ignore all regions that exceed this latency.
- **maxDistance** – Maximum difference between the latency of each player and the distance reference value.
- **distanceReference** – Number value to measure distance against for a successful match. For latency, this value is an aggregate of latency values for multiple players. Valid options are minimum (min) or average (avg) player latency value. (See the Property Expressions section.)
- **partyAggregation** – How to sort multiple-player (party) requests. Valid options are to use the minimum (min), maximum (max), or average (avg) values for a request's players. Default is avg.

**Distance sort rule (distanceSort)**

Distance sort is an explicit sort option that directs the matchmaker to pre-sort matchmaking requests based on a player attribute. A distance sort rule evaluates matchmaking requests based on the distance from the anchor request.

- **sortDirection** – Direction to sort matchmaking requests. Valid options are ascending or descending.
- **sortAttribute** – Player attribute to sort players by.
- **mapKey** – How to evaluate the player attribute if it's a map. Valid options include:
  - **minValue** – For the anchor player, find the key with the lowest value.
  - **maxValue** – For the anchor player, find the key with the highest value.
- **partyAggregation** – How to sort multiple-player (party) requests. Valid options are to use the minimum (min), maximum (max), or average (avg) values for a request's players. Default is avg.

**Absolute sort rule (absoluteSort)**

Absolute sort is an explicit sort option that directs the matchmaker to pre-sort matchmaking requests based on a player attribute. An absolute sort evaluates matchmaking requests based on whether its player attribute matches that of the anchor request.

- **sortDirection** – Direction to sort matchmaking requests. Valid options are ascending or descending.
- **sortAttribute** – Player attribute to sort players by.
- **mapKey** – How to evaluate the player attribute if it's a map. Valid options include:
  - **minValue** – For the anchor player, find the key with the lowest value.
  - **maxValue** – For the anchor player, find the key with the highest value.
- **partyAggregation** – How to sort multiple-player (party) requests. Valid options are to use the minimum (min), maximum (max), or average (avg) values for a request's players. Default is avg.
Property Expressions

Property expressions are used in a rule set to refer to certain matchmaking-related properties. Properties can include the player attribute values from matchmaking requests. For example, a rule might use a property expression to identify which player attribute to evaluate.

They generally take one of two forms:

- Individual player data
- Calculated team data, which takes the form of collections of individual player data.

A valid property expression identifies a specific value for a single player, team, or match. The following partial expressions illustrate how to identify teams and players:

| To identify a specific team in a match: | teams[red] | The Red team | Team |
| To identify all teams in a match: | teams[*] | All teams | List<Team> |
| To identify players in a specific team: | team[red].players | Players in the Red team | List<Player> |
| To identify players in a match: | team[*].players | Players in the match, grouped by team | List<List<Player>> |

The following table illustrates some valid property expressions that build on the previous examples:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
<th>Resulting Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>teams[red].players[playerid]</td>
<td>The player IDs of all players on the red team</td>
<td>List&lt;string&gt;</td>
</tr>
<tr>
<td>teams[red].players.attributes[skill]</td>
<td>The &quot;skill&quot; attributes of all players on the red team</td>
<td>List&lt;number&gt;</td>
</tr>
<tr>
<td>teams[*].players.attributes[skill]</td>
<td>The &quot;skill&quot; attributes of all players in the match, grouped by team</td>
<td>List&lt;List&lt;number&gt;&gt;</td>
</tr>
</tbody>
</table>

Property expressions can be used to aggregate team data by using the following functions or combinations of functions:

<table>
<thead>
<tr>
<th>Aggregation</th>
<th>Input</th>
<th>Meaning</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>List&lt;number&gt;</td>
<td>Get the minimum of all numbers in the list.</td>
<td>number</td>
</tr>
<tr>
<td>max</td>
<td>List&lt;number&gt;</td>
<td>Get the maximum of all numbers in the list.</td>
<td>number</td>
</tr>
<tr>
<td>avg</td>
<td>List&lt;number&gt;</td>
<td>Get the average of all numbers in the list.</td>
<td>number</td>
</tr>
</tbody>
</table>
### Aggregation

<table>
<thead>
<tr>
<th>Aggregation</th>
<th>Input</th>
<th>Meaning</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>median</td>
<td>List&lt;number&gt;</td>
<td>Get the median of all numbers in the list.</td>
<td>number</td>
</tr>
<tr>
<td>sum</td>
<td>List&lt;number&gt;</td>
<td>Get the sum of all numbers in the list.</td>
<td>number</td>
</tr>
<tr>
<td>count</td>
<td>List&lt;?&gt;</td>
<td>Get the number of elements in the list.</td>
<td>number</td>
</tr>
<tr>
<td>stddev</td>
<td>List&lt;number&gt;</td>
<td>Get the standard deviation of all numbers in the list.</td>
<td>number</td>
</tr>
<tr>
<td>flatten</td>
<td>List&lt;List&lt;?&gt;&gt;</td>
<td>Turn a collection of nested lists into a single list containing all elements.</td>
<td>List&lt;?&gt;</td>
</tr>
<tr>
<td>set_intersection</td>
<td>List&lt;List&lt;string&gt;&gt;</td>
<td>Get a list of strings that are found in all string lists in a collection.</td>
<td>List&lt;string&gt;</td>
</tr>
</tbody>
</table>

All above List<List<?>> All operations on a nested list operate on each sublist individually to produce a list of results.

The following table illustrates some valid property expressions that use aggregation functions:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
<th>Resulting Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten(teams[*].players.attributes[skill])</td>
<td>The &quot;skill&quot; attributes of all players in the match (not grouped)</td>
<td>List&lt;number&gt;</td>
</tr>
<tr>
<td>avg(teams[red].players.attributes[skill])</td>
<td>The average skill of the red team players</td>
<td>number</td>
</tr>
<tr>
<td>avg(teams[*].players.attributes[skill])</td>
<td>The average skill of each team in the match</td>
<td>List&lt;number&gt;</td>
</tr>
<tr>
<td>avg(flatten(teams[*].players.attributes[skill]))</td>
<td>The skill level of all players in the match. This expression gets a flattened list of player skills and then averages them.</td>
<td>number</td>
</tr>
<tr>
<td>count(teams[red].players)</td>
<td>The number of players on the red team</td>
<td>number</td>
</tr>
<tr>
<td>count (teams[*].players)</td>
<td>The number of players on each team in the match</td>
<td>List&lt;number&gt;</td>
</tr>
<tr>
<td>max(avg(teams[*].players.attributes[skill]))</td>
<td></td>
<td>number</td>
</tr>
</tbody>
</table>
FlexMatch Events Reference

Amazon GameLift emits events that are related to the processing of matchmaking tickets. All the events that are listed here can be published to an Amazon SNS topic. These events are also emitted to Amazon CloudWatch Events. For more details working with matchmaking events, see Set up FlexMatch Event Notification (p. 105). For more information on matchmaking ticket statuses, see MatchmakingTicket in the Amazon GameLift Service API Reference.

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
<th>Resource</th>
<th>Detail</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatchmakingSearching</td>
<td>Ticket has been entered into matchmaking. This includes new requests and requests that were part of a proposed match that failed.</td>
<td>ConfigurationArn, tickets, estimatedWaitMillis, gameSessionInfo</td>
<td></td>
<td>{ &quot;version&quot;: &quot;0&quot;, &quot;id&quot;: &quot;cc3d3ebe-1d90-48f8-b268-c9665b8f013&quot;, &quot;detail-type&quot;: &quot;GameLift Matchmaking Event&quot;, &quot;source&quot;: &quot;aws.gamelift&quot;, &quot;account&quot;: &quot;123456789012&quot;, &quot;time&quot;: &quot;2017-08-08T21:15:36.421Z&quot;, &quot;region&quot;: &quot;us-west-2&quot;, &quot;resources&quot;: [ &quot;arn:aws:gameli...configuration/SampleConfiguration&quot; ], &quot;detail&quot;: { &quot;tickets&quot;: [ { &quot;ticketId&quot;: &quot;ticket-1&quot;, &quot;startTime&quot;: &quot;2017-08-08T21:15:35.676Z&quot;, &quot;players&quot;: [ { &quot;playerId&quot;: &quot;player-1&quot; } ] } ], &quot;estimatedWaitMillis&quot;: &quot;NOT_AVAILABLE&quot;, &quot;type&quot;: &quot;MatchmakingSearching&quot;, &quot;gameSessionInfo&quot;: { &quot;players&quot;: [ { &quot;playerId&quot;: &quot;player-1&quot; } ] } } }</td>
</tr>
<tr>
<td>PotentialMatchCreated</td>
<td>Potential match has been created. This is emitted for all new potential matches, regardless.</td>
<td>ConfigurationArn, tickets, acceptanceTimeout, acceptanceRequired, ruleEvaluationMetrics, gameSessionInfo, matchid</td>
<td></td>
<td>{ &quot;version&quot;: &quot;0&quot;, &quot;id&quot;: &quot;fcede8653f-aea3-45bc-9bde3c3e3cad2d4&quot;, &quot;detail-type&quot;: &quot;GameLift Matchmaking Event&quot;, &quot;source&quot;: &quot;aws.gamelift&quot;, &quot;account&quot;: &quot;123456789012&quot;, &quot;time&quot;: &quot;2017-08-08T21:17:41.178Z&quot;, &quot;gameSessionId&quot;: &quot;SampleGameSession&quot; }</td>
</tr>
</tbody>
</table>

Version
177
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description of whether acceptance is required.</th>
<th>Resource</th>
<th>Detail</th>
<th>Example</th>
</tr>
</thead>
</table>

```json
"region": "us-west-2",
"resources": [  
  
  "arn:aws:gamelift:us-west-2:123456789012:matchmakingconfiguration/SampleConfiguration"  
],  
"detail": {  
  "tickets": [  
    
    {"ticketId": "ticket-1",  
     "startTime": "2017-08-08T21:15:35.676Z",  
     "players": [  
      
      {"playerId": "player-1",  
       "team": "red"  
      }]  
    },  
    
    {"ticketId": "ticket-2",  
     "startTime": "2017-08-08T21:17:40.657Z",  
     "players": [  
      
      {"playerId": "player-2",  
       "team": "blue"  
      }]  
    }]  
  },  
  
  "acceptanceTimeout": 600,  
  "ruleEvaluationMetrics": [  
  
    {"ruleName": "EvenSkill",  
     "passedCount": 3,  
     "failedCount": 0  
    },  
    
    {"ruleName": "EvenTeams",  
     "passedCount": 3,  
     "failedCount": 0  
    },  
    
    {"ruleName": "FastConnection",  
     "passedCount": 3,  
     "failedCount": 0  
    },  
    
    {"ruleName": "NoobSegregation",  
     "passedCount": 3,  
     "failedCount": 0  
    }  
  ],  
  
  "acceptanceRequired": true,  
  "type": "PotentialMatchCreated",  
  "gameSessionInfo": {  
    "players": [  
      
      {"playerId": "player-1",  
       "team": "red"  
      }]  
  }  
```

Version 178
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
<th>Resource</th>
<th>Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
{
    "playerId": "player-2",
    "team": "blue"
  }
],
"matchId": "3faf26ac-f06e-43e5-8d86-08fef26f692"
}
```
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
<th>Resource Info</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcceptMatch</td>
<td>Players have accepted a potential match. This event contains the current acceptance status of each player in the match. Missing data means that AcceptMatch hasn't been called for that player.</td>
<td>ConfigurationArn, tickets, matchId, gameSessionInfo</td>
<td>{ &quot;version&quot;: &quot;0&quot;, &quot;id&quot;: &quot;b3f76d66-c8e5-416a-aa4c-aa1278153edc&quot;, &quot;detail-type&quot;: &quot;GameLift Matchmaking Event&quot;, &quot;source&quot;: &quot;aws.gamelift&quot;, &quot;account&quot;: &quot;133456789012&quot;, &quot;time&quot;: &quot;2017-08-09T20:04:42.660Z&quot;, &quot;region&quot;: &quot;us-west-2&quot;, &quot;resources&quot;: [ &quot;arn:aws:gamelift:us-west-2:123456789012:matchmakingconfiguration/SampleConfiguration&quot; ],  &quot;detail&quot;: { &quot;tickets&quot;: [ { &quot;ticketId&quot;: &quot;ticket-1&quot;, &quot;startTime&quot;: &quot;2017-08-09T20:01:35.305Z&quot;, &quot;players&quot;: [ { &quot;playerId&quot;: &quot;player-1&quot;, &quot;team&quot;: &quot;red&quot; } ] }, { &quot;ticketId&quot;: &quot;ticket-2&quot;, &quot;startTime&quot;: &quot;2017-08-09T20:04:16.637Z&quot;, &quot;players&quot;: [ { &quot;playerId&quot;: &quot;player-2&quot;, &quot;team&quot;: &quot;blue&quot;, &quot;accepted&quot;: false } ] } ],  &quot;type&quot;: &quot;AcceptMatch&quot;,  &quot;gameSessionInfo&quot;: {  &quot;players&quot;: [ { &quot;playerId&quot;: &quot;player-1&quot;, &quot;team&quot;: &quot;red&quot; }, { &quot;playerId&quot;: &quot;player-2&quot;, &quot;team&quot;: &quot;blue&quot;, &quot;accepted&quot;: false } ] },  &quot;matchId&quot;: &quot;848b5f1f-0460-488e-8631-2960934d13e5&quot; } }</td>
</tr>
<tr>
<td>Event Name</td>
<td>Description</td>
<td>Resource</td>
<td>Detail</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>AcceptMatchCompleted</td>
<td>Acceptance is complete due to player acceptance, player rejection, or acceptance timeout.</td>
<td>ConfigurationArn, tickets, acceptance, matchId, gameSessionInfo</td>
<td>{</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;version&quot;: &quot;0&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;id&quot;: &quot;b1990d3d-f737-4d6c-b150-5ace8c35d3&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;detail-type&quot;: &quot;GameLift Matchmaking Event&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;source&quot;: &quot;aws.gamelift&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;account&quot;: &quot;123456789012&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;region&quot;: &quot;us-west-2&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;resources&quot;: [</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;arn:aws:gamelift:us-west-2:123456789012:matchmakingconfiguration/SampleConfiguration&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>],</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;detail&quot;: {</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;tickets&quot;: [</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;ticketId&quot;: &quot;ticket-1&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;startTime&quot;: &quot;2017-08-08T20:30:40.972Z&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;players&quot;: [</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;playerId&quot;: &quot;player-1&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;team&quot;: &quot;red&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>},</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;ticketId&quot;: &quot;ticket-2&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;startTime&quot;: &quot;2017-08-08T20:33:14.111Z&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;players&quot;: [</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;playerId&quot;: &quot;player-2&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;team&quot;: &quot;blue&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>],</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;acceptance&quot;: &quot;TimedOut&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;type&quot;: &quot;AcceptMatchCompleted&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;gameSessionInfo&quot;: {</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;players&quot;: [</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;playerId&quot;: &quot;player-1&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;team&quot;: &quot;red&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>},</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;playerId&quot;: &quot;player-2&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;team&quot;: &quot;blue&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;matchId&quot;: &quot;a0d9bd23+4695-4f12-876f-ea6386d6dce&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td>Event Name</td>
<td>Description</td>
<td>Resource</td>
<td>Detail</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>----------</td>
<td>--------</td>
</tr>
</tbody>
</table>
| MatchmakingSucceeded | Matchmaking has successfully completed and a game session has been created. | ConfigurationArn, type, tickets, matchId, gameSessionInfo | | {
"version": "0",
"id": "5cb6523-0566-412d-b63c-1569e00d023d",
"detail-type": "GameLift Matchmaking Event",
"source": "aws.gamelift",
"account": "123456789012",
"time": "2017-08-09T19:59:09.159Z",
"region": "us-west-2",
"resources": [
"arn:aws:gamelift:us-west-2:123456789012:matchmakingconfiguration/SampleConfiguration"
],
"detail": {
"tickets": [
{
"ticketId": "ticket-1",
"startTime": "2017-08-09T19:58:59.277Z",
"players": [
{
"playerId": "player-1",
"playerSessionId": "psess-6e7c13cf-10d6-4756-a53f-db7de782ed67",
"team": "red"
}
],
"ticketId": "ticket-2",
"startTime": "2017-08-09T19:59:08.663Z",
"players": [
{
"playerId": "player-2",
"playerSessionId": "psess-786b342f-9c94-44eb-bb9e-c1de46c472ce",
"team": "blue"
}
]
},
"type": "MatchmakingSucceeded",
"gameSessionInfo": {
"gameSessionArn": "arn:aws:gamelift:us-west-2:123456789012:gamesession/836cf48d-bcb0-4a2c-be1c-9c456541352a",
"ipAddress": "192.168.1.1",
"port": 10777,
"players": [
{
"playerId": "player-1",
"playerSessionId": "psess-6e7c13cf-10d6-4756-a53f-db7de782ed67",
"team": "red"
}


Version
182
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
<th>Resource</th>
<th>Detail</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>`{</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;playerId&quot;: &quot;player-2&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;playerSessionId&quot;:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;psess-786b342f-9c94-44eb-bb9e-c1de46c472ce&quot;,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;team&quot;: &quot;blue&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>},</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;matchId&quot;:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;c0ec1a54-7fec-4b55-8583-76d67adb7754&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td>Event Name</td>
<td>Description</td>
<td>Resource</td>
<td>Detail</td>
<td>Example</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------</td>
<td>----------</td>
<td>--------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| MatchmakingTimedOut | Matchmaking ticket has failed by timing out. | Configuration Arn | type, tickets, ruleEvaluationMetrics, message, matchId, gameSessionInfo | ```json
    { 
        "version": "0",
        "id": "fe528a7d-46ad-4bdc-96cb-b094b5f6bf56",
        "detail-type": "GameLift Matchmaking Event",
        "source": "aws.gamelift",
        "account": "123456789012",
        "time": "2017-08-09T20:11:35.598Z",
        "region": "us-west-2",
        "resources": [ 
            "arn:aws:gamelift:us-west-2:123456789012:matchmakingconfiguration/SampleConfiguration" 
        ],
        "detail": { 
            "reason": "TimedOut",
            "tickets": [ 
                { 
                    "ticketId": "ticket-1",
                    "startTime": "2017-08-09T20:01:35.305Z",
                    "players": [ 
                        { 
                            "playerId": "player-1",
                            "team": "red" 
                        } 
                    ] 
                } 
            ],
            "ruleEvaluationMetrics": [ 
                { 
                    "ruleName": "EvenSkill",
                    "passedCount": 3,
                    "failedCount": 0 
                },
                { 
                    "ruleName": "EvenTeams",
                    "passedCount": 3,
                    "failedCount": 0 
                },
                { 
                    "ruleName": "FastConnection",
                    "passedCount": 3,
                    "failedCount": 0 
                },
                { 
                    "ruleName": "NoobSegregation",
                    "passedCount": 3,
                    "failedCount": 0 
                } 
            ],
            "type": "MatchmakingTimedOut",
            "message": "Removed from matchmaking due to timing out."
        },
        "gameSessionInfo": { 
            "players": [ 
                { 
                    "playerId": "player-1",
                    "team": "red" 
                } 
            ] 
        }
    }``` |
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
<th>Resource</th>
<th>Detail</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td><strong>Event Name</strong></td>
<td><strong>Description</strong></td>
<td><strong>Resource</strong></td>
<td><strong>Detail</strong></td>
<td><strong>Example</strong></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
| MatchmakingCancelled | Matchmaking ticket has been cancelled. | ConfigurationArn, tickets, ruleEvaluationMetrics, message, matchId, gameSessionInfo | | ```json
"version": "0",
"id": "8d6f84da-5e15-4741-8d5c-5ac99091c27f",
"detail-type": "GameLift Matchmaking Event",
"source": "aws.gamelift",
"account": "123456789012",
"time": "2017-08-09T20:00:07.843Z",
"region": "us-west-2",
"resources": [
  "ARN: aws:gamelift:us-west-2:123456789012:matchmakingconfiguration/SampleConfiguration"
],
"detail": {
  "reason": "Cancelled",
  "tickets": [
    {
      "ticketId": "ticket-1",
      "startTime": "2017-08-09T19:59:26.118Z",
      "players": [
        {
          "playerId": "player-1"
        }
      ]
    }
  ],
  "ruleEvaluationMetrics": [
    {
      "ruleName": "EvenSkill",
      "passedCount": 0,
      "failedCount": 0
    },
    {
      "ruleName": "EvenTeams",
      "passedCount": 0,
      "failedCount": 0
    },
    {
      "ruleName": "FastConnection",
      "passedCount": 0,
      "failedCount": 0
    },
    {
      "ruleName": "NoobSegregation",
      "passedCount": 0,
      "failedCount": 0
    }
  ],
  "type": "MatchmakingCancelled",
  "message": "Cancelled by request.",
  "gameSessionInfo": {
    "players": [
      {
        "playerId": "player-1"
      }
    ]
  }
}
<table>
<thead>
<tr>
<th>Event Name</th>
<th>Description</th>
<th>Resource</th>
<th>Detail</th>
<th>Example</th>
</tr>
</thead>
</table>
# Document History for Amazon GameLift

The following table describes important changes to the Amazon GameLift documentation. For details on releases of new and updated features, see the Amazon GameLift Release Notes.

<table>
<thead>
<tr>
<th>Date</th>
<th>Change</th>
<th>Documentation updates</th>
<th>API Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>May 10, 2018</td>
<td><strong>New Features:</strong></td>
<td>• Auto-scaling with Target Tracking – Use Amazon GameLift's new scaling method.</td>
<td>AWS SDK: 2018-02-15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Enable/disable auto-scaling – Switch between auto-scaling and manual scaling without having to delete scaling policies.</td>
<td>Server SDK 3.2.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Developer Guide:</strong></td>
<td>GameLiftLocal 1.0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• New section Manage Fleet Capacity (p. 60) encompasses all scaling-related topics, including auto-scaling.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated section on Managing Capacity and Scaling (p. 3) in the &quot;How Amazon GameLift Works&quot; topic.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Service API Reference for the AWS SDK:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• PutScalingPolicy topic updated</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• New code examples for all scaling policy API actions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• New API actions StopFleetActions and StartFleetActions</td>
<td></td>
</tr>
<tr>
<td>February 15, 2018</td>
<td><strong>New Features:</strong></td>
<td>• Spot Instances and FleetIQ – Use Amazon GameLift's new spot fleets to significantly lower hosting costs.</td>
<td>AWS SDK: 2018-02-15</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Developer Guide:</strong></td>
<td>Server SDK 3.2.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Spot Fleet Integration Guide (p. 69) – New step-by-step guide for spot fleet usage</td>
<td>GameLiftLocal 1.0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Design a Game Session Queue (p. 79) – New design tips for creating and using queues, including with spot fleets</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updated topics:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Choose Computing Resources (p. 52)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create a Fleet (p. 54)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shut Down a Server Process (p. 32)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Service API Reference for the AWS SDK:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated topics:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CreateFleet</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FleetAttributes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Event</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Change</td>
<td>Documentation updates</td>
<td>API Versions</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td></td>
<td><strong>New Features:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>February 8,</td>
<td>• <strong>FlexMatch Backfill</strong> – Use FlexMatch</td>
<td>• How Amazon GameLift FlexMatch Works (p. 11) – Updated FlexMatch feature overview</td>
<td></td>
</tr>
<tr>
<td>2018</td>
<td>to inject new players into matched</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>game sessions that are in progress.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• <strong>Game session search</strong> – use custom game</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>properties</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Developer Guide:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• StartMatchBackfill</td>
<td>• Get Game Sessions (p. 34) – Updated for custom game properties</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SearchGameSessions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• GameSession</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Service API Reference for the AWS SDK:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• StartMatchBackfill</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SearchGameSessions</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• GameSession</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Server SDK:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• StartMatchBackfill (C++) (p. 145)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• StartMatchBackfill (C#) (p. 159)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• StopMatchBackfill (C++) (p. 146)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• StopMatchBackfill (C#) (p. 160)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Change</td>
<td>Documentation updates</td>
<td>API Versions</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------</td>
<td>------------------------------------------------------------</td>
<td>----------------------------------------</td>
</tr>
</tbody>
</table>
| September 1, 2017 | **New Features:** VPC Peering for Amazon GameLift fleets – You can now use VPC peering to set up direct, private connections between your Amazon GameLift game servers and other resources on AWS. | **Developer Guide:**  
- Networking With AWS Resources (p. 5) – Get more information on VPC peering and how it works with Amazon GameLift.  
- Set Up VPC Peering (p. 74) – Learn more about how to set up VPC peering. | AWS SDK: 2017-09-01  
Server SDK 3.1.7  
GameLiftLocal 1.0.0 |
| August 16, 2017   | **New Features:** Amazon GameLift FlexMatch – Add matchmaking to your games using Amazon GameLift's new customizable matchmaking service. With FlexMatch, you can design a rule set based on the team formats and player characteristics that best fit your game. | **Developer Guide:**  
- Amazon GameLift FlexMatch (p. 10) – Get more information on FlexMatch key features and how it works.  
- Matchmaking with FlexMatch (p. 89) – Learn more about how to set up FlexMatch and customize it to your game. | AWS SDK: 2017-08-16  
Server SDK 3.1.7  
GameLiftLocal 1.0.0 |
<table>
<thead>
<tr>
<th>Date</th>
<th>Change</th>
<th>Documentation updates</th>
<th>API Versions</th>
</tr>
</thead>
</table>
| May 16, 2017 | **New Features:**                           | • Amazon GameLiftMetrics are now supported in Amazon CloudWatch. This includes the ability to work with aggregated metrics for a group of fleets.  
• Limit game session activations on instances in a fleet. | AWS SDK: 2017-05-16   
Server SDK 3.1.5  
GameLiftLocal 1.0.0 |
|            | **Updates:**                                | • Take advantage of additional metrics for automatic scaling.                         |                       |
|            |                                             | • Use new console UI to set fleet scaling.                                           |                       |
|            | **Developer Guide:**                        | • Monitoring Amazon GameLift (p. 125) – New monitoring section including list of metrics available in the Amazon GameLift console and in CloudWatch.  
• Create a Fleet (Console) (p. 54) and Update a Fleet (p. 59) – Updated instructions on creating and updating fleet configurations.  
• Auto-Scale Fleet Capacity (p. 64) – Updated instructions on setting manual and auto-scaling for a fleet. |                       |
|            | **Service API Reference:**                  | • New MetricGroups parameter added to enable aggregated metrics:                     |                       |
|            |                                             | • CreateFleet  
• UpdateFleetAttributes  
• FleetAttributes  
• New parameters for game session activation limits added to RuntimeConfiguration. |                       |
<table>
<thead>
<tr>
<th>Date</th>
<th>Change</th>
<th>Documentation updates</th>
<th>API Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 11, 2017</td>
<td><strong>New Features:</strong></td>
<td><strong>Developer Guide:</strong></td>
<td>AWS SDK: 2017-04-11 Server SDK 3.1.5 GameLiftLocal 1.0.0</td>
</tr>
<tr>
<td></td>
<td>• Amazon GameLift Local – Test your game integration locally.</td>
<td>• Testing Your Integration (p. 41) – New topic on setting up and using Amazon GameLift Local.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Set a Queue policy that limits the latency allowed for individual players.</td>
<td>• Create a Queue (p. 83) – Updated topic on creating queues, with new information on player latency policies.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Updates:</strong></td>
<td><strong>Service API Reference:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Changes to the Amazon GameLift Service API (part of the AWS SDK) to improve usability.</td>
<td>• New data type PlayerLatencyPolicy used with CreateGameSessionQueue and UpdateGameSessionQueue.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• New data type PlacedPlayerSession describes a player session that was created while fulfilling a game session placement request.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Revised data type GameSessionPlacement now includes these properties after a game session placement request has been fulfilled: GameSessionId, IpAddress, Port, and PlacedPlayerSessions.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CreateGameSession has a new parameter, IdempotencyToken, to replace GameSessionId.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GameSessionQueue has a new unique identifier, GameSessionQueueArn, with region and game session ID.</td>
<td></td>
</tr>
<tr>
<td>February 21, 2017</td>
<td><strong>New features:</strong></td>
<td><strong>Developer Guide:</strong></td>
<td>AWS SDK: 2017-02-21 Server SDK 3.1.5</td>
</tr>
<tr>
<td></td>
<td>• Multiple game engine support, including Unreal Engine, Unity, and custom C++ and C# game engines</td>
<td>• Game Architecture with Amazon GameLift (p. 6) – Architecture diagram.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Language support for the Server SDK is expanded to include C#</td>
<td>• Game Engines and Amazon GameLift (p. 22) – Help with using Amazon GameLift with various game engines, and plugin setup instructions for Unreal Engine and Unity.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• New game session creation using game session placements and cross-region queues</td>
<td>• Working with Queues (p. 79) – Help with creating, managing, and tracking metrics for queues.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Custom player data support, with delivery directly to game server</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Change</td>
<td>Documentation updates</td>
<td>API Versions</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td></td>
<td>• Remote access to</td>
<td>• New topics:</td>
<td>Server SDK for C++:</td>
</tr>
<tr>
<td></td>
<td>Amazon GameLift fleet</td>
<td>• Remotely Access Fleet Instances (p. 72) – How to get access and remotely connect to</td>
<td>3.1.0</td>
</tr>
<tr>
<td></td>
<td>instances</td>
<td>Amazon GameLift instances.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Debug Fleet Creation Issues (p. 58) – Troubleshooting tips for new fleets that fail</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>to activate.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Service API Reference:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For remote access:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GetInstanceAccess (new)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• InstanceAccess (new)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• InstanceCredentials (new)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Resource creation</td>
<td>• Revised topics:</td>
<td>Server SDK for C++:</td>
</tr>
<tr>
<td></td>
<td>protection</td>
<td>• How Amazon GameLift Works (p. 2) – Added description of resource protection, and</td>
<td>3.1.0</td>
</tr>
<tr>
<td></td>
<td>• Access to instance data</td>
<td>improved description of capacity handling.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Updates and corrections:</td>
<td>• Added Linux-specific help:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Additional help for Linux.</td>
<td>Package Build Files (p. 46) – Install scripts for Linux.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Upload Build Files to Amazon GameLift (p. 47) – New Linux examples.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create a Fleet (Console) (p. 54) – New launch path example for Linux.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Service API Reference:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CreateFleet and UpdateFleetAttributes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ResourceCreationLimitPolicy (new)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CreateGameSession – New CreatorId parameter.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DescribeInstances (new)</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Change</td>
<td>Documentation updates</td>
<td>API Versions</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------</td>
<td>---------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
</tbody>
</table>
| September 1, 2016 | New features:  
  - Game servers can now run on Linux | **Developer Guide:**  
  - New topics:  
    - *Amazon GameLift SDKs (p. 14)* – Reference topic describing all Amazon GameLift SDKs, including supported languages and operating systems.  
  **Service API Reference:**  
  - New OS parameters were added to the following:  
    - *upload-build (p. 47)* (CLI only)  
    - *CreateBuild()*  
    - *Build*  
    - *FleetAttributes* | AWS SDK:  
  2016-09-01  
  Server SDK for C++:  
  3.1.0 |
| August 4, 2016 | New features:  
  - Game session search  
  - Customized health checks  
  Updates:  
  - Expanded support for capacity allocation (multiple processes per fleet instance)  
  - Amazon GameLift Server SDK for C++ now available for download  
  - All APIs for game client integration are now included in the AWS SDK. | **Developer Guide:**  
  - New topics:  
    - *Amazon GameLift Server API (C++) Reference (p. 137)* – Complete reference documentation.  
    - *Run Multiple Processes on a Fleet (p. 70)* – Technical overview of capacity allocation and how to configure a fleet to run multiple processes.  
    - *Tools and Resources (p. 13)* – Comprehensive list of tools & resources, including SDK version compatibility.  
  - Revised topics:  
    - *How Players Connect to Games (p. 8)* – Expanded topic describes features related to game sessions, including the new search feature.  
    - *Add Amazon GameLift to Your Game Server (p. 30)* – Integration steps have been revised for use with version 3.0.7 Server SDK for C++.  
    - *Add Amazon GameLift to Your Game Client (p. 33)* – Integration steps have been revised for use with the AWS SDK for C++.  
  **Service API Reference:**  
  - *SearchGameSessions() (new)* | AWS SDK:  
  2016-08-04  
  Server SDK for C++:  
  3.0.7 |
<table>
<thead>
<tr>
<th>Date</th>
<th>Change</th>
<th>Documentation updates</th>
<th>API Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• New server process health metrics</td>
<td>• Revised topics:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Revised processes for fleet capacity allocation and game server launch settings</td>
<td>• Package Build Files (p. 46) – Description now reflects how Amazon GameLift handles an install.bat file in a game build.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Revised build packaging instructions</td>
<td>• Create a Fleet (Console) (p. 54) and Create a Fleet (AWS CLI) (p. 56) – Instructions for creating a fleet now cover capacity allocation using a runtime configuration.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• View Fleet Details (p. 118) and View Data on Game and Player Sessions (p. 122) – Console page descriptions now reflect current metrics and scaling tabs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Amazon GameLift and Game Client/Server Interactions (p. 37) – Descriptions and diagram (p. 40) have been corrected to use callback function names from the samples, and to clarify that the onProcessTerminate() callback refers to shutting down a game server, not a game session.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Service API Reference:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For new capacity allocation:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CreateFleet() – Runtime configuration added.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DescribeRuntimeConfiguration (new)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• UpdateRuntimeConfiguration (new)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For game server launch process:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GameSession – Port number added. PlayerSession – Port number added.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For health metrics:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• FleetUtilization – New count added for active server processes.</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Change</td>
<td>Documentation updates</td>
<td>API Versions</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>March 10, 2016</td>
<td>New features:</td>
<td>Developer Guide:</td>
<td>AWS SDK</td>
</tr>
<tr>
<td></td>
<td>Auto-scaling</td>
<td>New topics:</td>
<td>2016-03-10</td>
</tr>
<tr>
<td></td>
<td>Game session protection</td>
<td>• Auto-Scale Fleet Capacity (p. 64) – How to set up and manage auto-scaling policies.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fleet capacity limits</td>
<td>• Manually Set Fleet Capacity (p. 62) – How to change the number of instances in a fleet and set limits.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• How Amazon GameLift Works (p. 2) – A technical overview of how Amazon GameLift manages game deployment across virtual resources.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Revised topics:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Create a Fleet (Console) (p. 54) – Revised to include settings for game session protection and safe scaling.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Other changes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Lumberyard-Amazon GameLift tutorial was moved to the GameDev Tutorials repository.</td>
<td></td>
</tr>
<tr>
<td>February 9, 2016</td>
<td>Service launch</td>
<td>Developer Guide and API Reference for the Amazon GameLift service released on AWS.</td>
<td>AWS SDK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Service API Reference:</td>
<td>2016-02-09</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For auto-scaling:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• PutScalingPolicy</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DescribeScalingPolicies</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DeleteScalingPolicy</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For game session protection:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DescribeGameSessionDetails</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CreateFleet (revised)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• UpdateFleetAttributes (revised)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DescribeFleetAttributes (revised)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• UpdateGameSession (revised)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For fleet capacity limits:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• UpdateFleetCapacity (revised)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DescribeFleetCapacity (revised)</td>
<td></td>
</tr>
</tbody>
</table>