# Table of Contents

What Is Amazon GameLift? ................................................................. 1
Why GameLift? .......................................................................................... 1
GameLift solutions .................................................................................. 1
  GameLift hosting ................................................................................. 1
  GameLift hosting with Realtime Servers ............................................. 2
  GameLift FleetIQ for hosting on Amazon EC2 .................................... 2
How GameLift works ............................................................................. 3
  Key components .................................................................................. 3
  Hosting game servers .......................................................................... 4
  Running game sessions ....................................................................... 5
  Scaling fleet capacity .......................................................................... 6
  Monitoring fleet activity and troubleshooting ..................................... 7
  Interacting with other AWS resources .............................................. 8
How Realtime Servers Work ............................................................... 8
  What are Realtime Servers? ............................................................... 8
  Choosing Realtime Servers for Your Game ...................................... 9
  Key Components ............................................................................... 9
  How Realtime Servers Manages Game Sessions .............................. 9
  How Realtime Clients and Servers Interact ...................................... 10
  Customizing a Realtime Server ....................................................... 11
  Deploying and Updating Realtime Servers ..................................... 11
How Players Connect to Games .......................................................... 12
  Game and Player Session Features ................................................ 12
  Game architecture with managed GameLift .................................... 13
  Game Architecture with Realtime Servers ..................................... 15
Setting Up ............................................................................................ 18
  Set up an AWS account ................................................................. 18
  IAM policy examples ........................................................................ 18
  Set up a role for GameLift access ................................................... 21
  GameLift SDKs ................................................................................ 22
    For custom game servers .............................................................. 22
    For Realtime Servers .................................................................... 23
    For client services ........................................................................ 23
  Tools and Resources ....................................................................... 24
    Core Tools .................................................................................... 24
    Additional Resources .................................................................... 24
  Setting up AWS billing alerts .......................................................... 25
  Using GameLift in AWS Regions .................................................... 25
    GameLift hosting .......................................................................... 25
    GameLift FlexMatch .................................................................... 26
    GameLift in China ....................................................................... 26
Getting Started .................................................................................... 27
  Explore GameLift ............................................................................ 27
    Custom Game Server Sample (Console Experience) ...................... 27
    Realtime Servers Sample Game (Full Source) .............................. 27
  Get Started with Custom Servers ................................................... 28
    Hosting Components Overview .................................................. 30
    Prepare Your Custom Game Server ............................................. 31
    Test the Integration Locally .......................................................... 31
    Plan your Global Resources Deployment ................................... 32
    Deploy your Resources .................................................................. 32
    Design your Backend Service ...................................................... 33
    Define and Implement Metrics and Logs Solutions ...................... 37
    Prepare for Launch ....................................................................... 37
What Is Amazon GameLift?

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

Why GameLift?

Here are some of the benefits of using Amazon GameLift:

- Bring your own fully custom multiplayer game servers or use ready-to-go Realtime Servers that require minimal configuration and little or no backend experience.
- Provide low-latency player experience to support fast-action game play.
- Enhance your matchmaking services with intelligent queuing, game session placement, and match backfill.
- Reduce engineering and operational effort to deploy and operate game servers globally.
- 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.

Tip

Learn more about ways to explore Amazon GameLift features, including Realtime Servers, using sample games (p. 27).

GameLift solutions

GameLift offers a range of solutions for game developers:

- GameLift hosting for custom-built game servers.
- GameLift hosting with Realtime Servers
- GameLift FleetIQ game hosting optimizations for use with Amazon EC2

GameLift hosting

Amazon GameLift offers a fully managed service for deploying, operating, and scaling session-based, multiplayer game servers. GameLift replaces the work required to host your own custom game servers, including buying and setting up hardware, and managing ongoing activity, security, storage, and performance tracking. Auto-scaling capabilities provide additional protection from having to pay for more resources than you need, while making sure you always have games available for new players to join with minimal waiting.

To learn more about how the GameLift hosting solution works, see How GameLift works (p. 3).

Key features
• Provide high-quality game hosting to players around the world by deploying computing resources in multiple AWS Regions.
• Deploy game servers to run on Amazon Linux or Windows Server operating systems.
• Let FleetIQ optimize the use of low-cost Spot Instances. On their own, Spot Instances are not always viable for game hosting due to potential interruptions. FleetIQ prediction algorithms find the Spot Instances that are best suited to host new game sessions.
• Use automatic scaling tools to adjust your game hosting capacity to meet actual player demand. These tools allow you to keep hosting costs in line while maintaining enough capacity to get new players into games fast.
• Build a custom matchmaking service for your game using FlexMatch. Create single-team or multi-team matches for up to 200 players.
• Manage game sessions and player sessions. Configure game session characteristics, such as maximum number of players allowed, join rules, and game-specific properties.
• Choose from a range of options to help players find suitable game sessions. Use GameLift Queues to intelligently place new game sessions across multiple regions, provide players with filtered and sorted lists of available game sessions ("list and pick"), or implement a full matchmaking system with FlexMatch.
• 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.
• Manage your game resources using AWS CloudFormation templates for GameLift.

GameLift hosting with Realtime Servers

Use Realtime Servers to stand up games that don't need custom-built game servers. This lightweight server solution provides ready-to-go game servers that can be configured to fit your game. You can deploy game servers with anything from minimal configuration settings to custom logic that is specific to your game and players.

To learn more about how GameLift hosting with Realtime Servers works, see How Realtime Servers Work (p. 8).

Key features

• Use GameLift management features, including auto-scaling, multi-region queues, game session placement with FleetIQ, game session logging, and metrics.
• Use GameLift hosting resources, choose the type of AWS computing hardware for your fleets. Use either Spot or On-Demand Instances.
• Take advantage of a full network stack for game client/server interaction.
• Get core game server functionality with customizable server logic.
• Make live updates to Realtime configurations and server logic. Update your Realtime server configuration at any time.
• Implement FlexMatch matchmaking.

GameLift FleetIQ for hosting on Amazon EC2

GameLift FleetIQ optimizes the use of low-cost Spot Instances for cloud-based game hosting. With this feature, you can work directly with your hosting resources in Amazon EC2 and Auto Scaling and take advantage of GameLift optimizations to deliver inexpensive, resilient game hosting for your players. This solution is designed for game developers who need more flexibility than is offered in the fully managed GameLift solutions.
To learn more about how GameLift FleetIQ works with Amazon EC2 and Auto Scaling for game hosting, see the GameLift FleetIQ Guide.

Key features

- Get optimized Spot balancing using GameLift FleetIQ prediction algorithms.
- Use player routing features to efficiently manage your game server resources and provide optimal player experience when joining games.
- Automatically scale hosting capacity based on player usage.
- Directly manage Amazon EC2 instances in your own AWS account.
- Use any of multiple supported game server executable formats, including Windows, Linux, containers, and Kubernetes.
- Choose from multiple types of Amazon EC2 computing resources.
- Reach players worldwide by deploying across 15 Regions, including China.

How GameLift works

This topic provides a general overview of the GameLift managed hosting solution. It covers the core components for game hosting and describes how GameLift makes your multiplayer game servers available to players. If you want to learn more about the mechanisms for GameLift managed hosting, start with this topic and the following related topics. To learn about other GameLift solutions, see What Is Amazon GameLift? (p. 1).

- How PlayersConnect to Games (p. 12)
- Game Engines and Amazon GameLift (p. 43)
- Setting up GameLift queues for game session placement (p. 137)
- How GameLift FlexMatch works

Ready to start prepping your game for hosting on GameLift? See these Getting Started with Amazon GameLift (p. 27) topics, including integration pathways.

Key components

Setting up GameLift to host your game involves working with the following components. The relationships between these components is illustrated in Game architecture with managed GameLift (p. 13).

- A game server is your game's server software running in the cloud. You upload your game server build (or script if you're using Realtime Servers (p. 8)) to the GameLift service and tell GameLift how many game servers make available for game sessions.
- A game session is your game in progress with players. You define the basic characteristics of a game session, such as its life span and number of players. Players connect to the game server to join a game session.
- The GameLift service manages the computing resources to host your game server and makes it possible for players to connect to games. It regulates the resources needed to host games, finds available game servers to host new game sessions, and puts players into games. The service also collects metrics on hosting activity and server health.
- A game client is your game's software running on a player's device. Players might use the game client to get information about available game sessions and/or request to join games. Game clients connect directly to a game server to join a game session, based on connection information it receives from the GameLift service.
• **Game services** are additional custom services that you might create to handle special tasks related to GameLift. As a best practice, all client communication with the GameLift service is handled by a client service; this is recommended for security reasons and efficiency.

For a detailed description of how these components interact, see [GameLift and game client/server interactions (p. 62)].

## Hosting game servers

Your uploaded game servers are hosted on GameLift virtual computing resources, called **instances**. You set up your hosting resources by creating a **fleet** of instances and deploying them to run your game servers (either a custom game server or Realtime Servers). You can design a fleet to fit your game's needs.

### Fleet architecture

Build a fleet that suits your core requirements.

- **What type of resources does your game need?** – GameLift supports a range of operating systems and instance types. An instance type is a specific configuration of computing hardware, including processing power, memory, and networking capacity. Note that hosting costs are based on the instance type, location, and volume of instances you use. Depending on your game's requirements, you might opt to use many small instances or fewer more powerful instances. Learn more about [Choosing computing resources (p. 107)].

- **Where do you want to run your game servers?** – You set up fleets to deploy instances wherever you have players waiting to join games. You can build multi-location fleets that deploy instances to any or all AWS Regions that are supported in GameLift. See more information on [Using Amazon GameLift in AWS Regions (p. 25)].

- **How critical is game server reliability?** – Your fleet uses either Spot instances or On-Demand instances. Spot instances (based on EC2's Spot instances) usually cost less, but may be interrupted during a game session. However, GameLift has additional safeguards that make game session interruptions extremely rare, and fleets with Spot instances are a good choice for most games. On-demand instances, in contrast, provide consistent availability but can be more expensive. They are a good option for games where players are strongly impacted if a game session is interrupted. Learn more about [On-Demand versus Spot Instances (p. 108)].

- **How many players do you need to support?** – A fleet can have many instances, each one capable of hosting multiple simultaneous game sessions. You can add or remove instances from your fleet as needed, and you can use auto-scaling to automatically adjust as player demand shifts. Learn more about [Scaling fleet capacity (p. 6)].

### Server runtime configuration

A fleet instance can run multiple processes simultaneously, and it can run any executable in your game server build. To determine what processes each instance should run, you create a runtime configuration. The configuration specifies: (1) which executables to run, (2) how many processes of each executable to run concurrently, and (3) any launch parameters to use when starting each executable. The number of processes that an instance can run simultaneously depends on the instance's computing power (instance type) as well as the requirements of your game server build. Learn more about [running multiple processes on a fleet (p. 109)]. Runtime configurations can be updated throughout the life of the fleet.

A runtime configuration can also affect how new game sessions are started on an instance. Some games require a lot of resources during the start-up phase, and it can be a good idea to limit the instance resources that are used to activate new game sessions at any one time. You can specify a maximum number of simultaneous game session activations per instance. You can also place a time limit on each game session activation to quickly detect and shut down any game sessions that are failing to activate.
Server security

Enable PKI resource generation for a fleet. When this feature is turned on, GameLift generates a TLS certificate for the fleet and creates a DNS entry for each instance in the fleet. With these resources, your game can authenticate the client and server connection and encrypt all game client/server communication. This feature is particularly valuable when deploying mobile multi-player games. These services are provided through the AWS Certificate Manager (ACM) and are currently available at no additional cost.

Fleet aliases

An alias is a designation that can be transferred from one actual fleet to another, making it a convenient way to genericize a fleet location. For example, your game client needs to specify where (which fleet) to place a new game session. Using an alias lets you switch game clients from using one fleet to another without having to change your game client. There are several GameLift features where you have the option to specify a fleet or an alias. You can also create a “terminal” alias, which lets you point to content (such as a URL) instead of connecting to a server. This capability can be useful, for example, to prompt players to upgrade their clients.

Running game sessions

After your game server code is successfully deployed to a fleet and game server processes are launched on each instance, the fleet is ready to host game sessions. New game sessions are started when your game client service sends a placement request to the GameLift service.

Game session placement and FleetIQ

Game session placement handles the task of selecting an available game server to host a new game session. The key component for game session placement is the GameLift game session queue. Each placement request for a new game session is sent to a specified queue for processing.

A game session queue is assigned a list of fleets, which determines where it can place game sessions. As a best practice, a queue’s fleets vary in fleet type (Spot or On-Demand), locations, and/or instance type (computing hardware). This diversity gives the queue greater flexibility to make placements wherever they most make sense for players. Queues with Spot instances can be configured to minimize the impact of Spot instance interruptions on your game sessions. See more about game session queues and how to design them for your game in Design a game session queue (p. 139).

The game session placement process does more than simply place new game sessions with any available game server it finds. Queues use a special algorithm, called FleetIQ, to look for the “best possible” placement for each game session request. For more information on FleetIQ, see How GameLift FleetIQ works (p. 155). The FleetIQ algorithm prioritizes the search for available game servers based on low player latency, hosting costs, geographical locations, or other fleet characteristics.

You create a game session queue to serve each pool of players for your game. Player pools are groups of people who can play together in the same game sessions. For example, you might have different player pools for people who speak different languages or use different game versions.

Note

GameLift gives you the option to locate and place game sessions manually, a useful option if you want to build a custom game session placement mechanism. However, this approach bypasses the placement optimizations offered by FleetIQ. If you’re using FlexMatch matchmaking or Spot fleets, game session queues are required.

Player connections to games

As part of the game session placement process, the queue prompts the selected game server to start a new game session. The game server, which has been integrated with the GameLift Server SDK, responds
to the prompt and reports back to the GameLift service when it's ready to accept player connections. The 
GameLift service then delivers connection information to your game client service, in the form of an IP 
address or DNS name and a port. Your game clients use this information to connect directly to the game 
session and begin gameplay.

If the fleet is created with a TLS certificate, your game client and server can use it to establish a secure 
connection.

You can optionally set up your game server to verify incoming players with the GameLift service, and to 
regularly report player connection status. These options are recommended to help GameLift keep track 
of game session usage.

Scaling fleet capacity

Once a fleet is active and ready to host game sessions, you can adjust your fleet capacity to meet player 
demand. The cost of hosting is based on the amount of capacity you use, so you'll want to find a balance 
between making sure all incoming players can find a game and overspending on resources that sit idle.

You scale a fleet by adjusting the number of instances in it. For fleets with instances in multiple locations, 
you adjust capacity by location instead of for the entire fleet.

Your runtime configuration determines how many game sessions and players each instance can host, 
so by scaling instances you're increasing or decreasing the availability for game sessions and players. 
GameLift provides a highly effective auto-scaling tool, or you can opt to manually set fleet capacity. 
Learn more about how to Scaling GameLift hosting capacity (p. 129).

Auto-scaling

With auto-scaling enabled, GameLift tracks a fleet's hosting metrics and determines when to add or 
remove instances based on a set of guidelines that you define. With the right auto-scaling policies in 
place, GameLift can adjust capacity directly in response to changes in player demand. Learn more about 
improving cost efficiency with automatic scaling. For multi-location fleets, auto-scaling policies apply to 
an entire fleet, but you have the option to turn them on or off for each location.

There are two methods of auto-scaling available:

- **Target-based scaling** – With this method, you specify a desired outcome and GameLift scales the 
  fleet up or down to achieve that outcome. Target tracking uses the metric "percent available game 
sessions", that is, the percentage of healthy server processes that are not currently hosting a game 
session. Available game sessions is your buffer--they represent the number of new game sessions and 
new players that can join a game with a minimal wait time. With target tracking, you choose the buffer 
size that fits your game. For example, for a game with highly volatile demand, you may need a larger 
buffer size. This method is the preferred option, as it is simpler and more effective for more games. 
Learn more about how Target Tracking works.

- **Rule-based scaling** – This method gives your more fine-grained control of scaling actions. It is also 
  more complex to set up and manage and is more likely to have unexpected results. Each policy 
specifies when to trigger a scaling event and what action to take in response. For example, a policy 
might state: “If idle instances falls below 20 for 10 consecutive minutes, then increase capacity by 
10%.” Most fleets require multiple policies to manage fleet capacity effectively, but multiple policies 
can have unexpected compound effects, which adds to the complexity. Learn how to Auto-scale with 
rule-based policies (p. 134).

Capacity scaling in action

A 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 or location's "desired" instance count does not match its "active" instance count. This circumstance causes GameLift to add or remove instances, as needed, to make the active instance count match the desired instance count.

- When desired instance count exceeds active instance count, 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. GameLift continues to add instances until the two count values are even.

- When active instance count exceeds desired instance count, GameLift begins searching for instances it can remove. Any available instance (that is, one not hosting any game sessions) can be terminated, as well as any non-protected instance even when hosting game sessions. If no instances can be removed, 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 be removed. 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. GameLift continues to remove instances until the active and desired instance 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. 108)).

- **Scaling limits** – Control overall instance usage by setting minimum and maximum limits on the number of instances in a fleet. These limits apply when auto-scaling or when manually setting capacity.

- **Enabling/disabling auto-scaling** – Switch auto-scaling on or off at the fleet level without changing or deleting your auto-scaling policies. This feature allows you to temporarily scale your fleets manually when needed.

- **Scaling metrics** – Track a fleet's history of capacity and scaling events in graph form. View capacity in conjunction with fleet utilization metrics to evaluate the effectiveness of your scaling approach.

### Monitoring fleet activity and troubleshooting

Once you have fleets up and running, 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, location, 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** – 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** – 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 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 [GameLift console](https://console.aws.amazon.com/game_lift). 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.

### Interacting with other AWS resources

In many situations, you want your hosted game servers and applications to be able to communicate with other AWS resources. For example, you might use a set of web services for player authentication or social networking. This type of communication poses a challenge due to resource ownership issues. When you deploy game servers using Amazon GameLift, the fleets and instances are allocated to your account but they are owned and managed by the GameLift service. As a result, to access AWS resources that are managed by your AWS account, you need to explicitly allow access by the Amazon GameLift service.

GameLift provides a couple of options for managing this type of access. Learn more about how to [Communicate with other AWS resources from your fleets](https://console.aws.amazon.com/game_lift/). (p. 55).

### How Realtime Servers Work

This topic provides an overview of the managed Amazon GameLift with Realtime Servers solution. It discusses when it is a good fit for your game, and explains how Realtime Servers supports multiplayer gaming. To learn more about other GameLift solutions, see [What Is Amazon GameLift?](https://console.aws.amazon.com/game_lift/) (p. 1).

### What are Realtime Servers?

Realtime Servers are lightweight, ready-to-go game servers that are provided by GameLift for you to use with your multiplayer games. While many games need a custom game server to handle complex physics and computations, this is overkill for many other games. Since Realtime Servers eliminate the need to develop, test, and deploy a custom game server, choosing this solution can help minimize the time and effort required to complete your game.

Key features include:

- **Full network stack for game client/server interaction.** Realtime Servers makes use of TCP and UDP channels for messaging. You can also opt to use built-in server authentication and data packet encryption by enabling GameLift-generated TLS certificates.

- **Core game server functionality.** A Realtime server starts (and stops) game sessions, manages game and match data, and accepts client connections. The game server maintains a synchronized game session state by receiving game state information from each client and relaying it to other clients in the game session.

- **Integrated with the GameLift service.** A Realtime server is set up to communicate with the GameLift service, which triggers the Realtime server to start game sessions, validate players when they connect, and collects player connection status and game health state from the game server. In contrast, this functionality must be implemented in a custom game server.

- **Customizable server logic.** You can configure your Realtime servers and customize them with server-side game logic as best fits your game. Alternatively, provide a minimal configuration and to use them as simple relay servers. Learn more about [Customizing a Realtime Server](https://console.aws.amazon.com/game_lift/) (p. 11).

- **Live updates to Realtime configurations and server logic.** Update your Realtime server configuration at any time. GameLift regularly checks for updated configuration scripts, so once you upload a new version, it is quickly deployed to your fleet and used with all new game sessions.

- **FlexMatch matchmaking.** Game clients that use Realtime Servers can make use of all FlexMatch matchmaking features, including for large matches.
Choosing Realtime Servers for Your Game

Choosing Realtime Servers instead of building a custom game server primarily comes down to your game’s need for server complexity. Unless your game needs complicated server-side game logic, split-second computations for gameplay physics, or other custom capabilities, Realtime Servers may be the better solution for your game. Games that use Realtime Servers to best effect include lighter weight games or games that manage a higher percentage of the computational work on the game client. Examples include messaging games, turn-based strategy games, and many types of mobile games. Realtime Servers, coupled with the use of FleetIQ, provides effective tools to minimize player latency suitable for nearly all game types.

Key Components

When working with Realtime Servers, you work with the following components. Learn more about these components and how they work together in Game Architecture with Realtime Servers (p. 15).

- A Realtime server provides client/server networking for your game. It starts game sessions when triggered by the GameLift service, requests validation for players when they connect, and reports back on the status player connections and game health. The server relays game state data between all connected players, and executes custom game logic if provided.

- A game client is your game’s software running on a player’s device. The game client (through a client service) makes requests to the GameLift service to find game sessions to join or to start new ones, and connects to a Realtime server to participate in a game. Once connected, a game client can send and receive data, through the Realtime server, with other players in the game.

- A Realtime script provides configuration settings and optional custom game logic for your game. The script may contain minimal configuration settings or have more complex game logic. The Realtime script is deployed along with the Realtime server when starting up new hosting resources. Scripts are written in Node.js-based JavaScript.

- The GameLift service manages the computing resources needed to host your Realtime servers and makes it possible for players to connect to games. It regulates the number of resources for player demand, handles player join requests by finding and reserving player slots in active game sessions, triggers Realtime servers to start game sessions, and validates players when they connect to a game server. The service also collects metrics on Realtime server health and player usage.

- A game session is an instance of your game, run on a Realtime server. Players connect to a game session to play the game and interact with the other players.

How Realtime Servers Manages Game Sessions

GameLift manages game sessions with Realtime Servers in the same way that it handles game sessions with fully custom game servers. Players, using a game client, send requests to create new game sessions
or to find and join existing game sessions. Most methods for creating game sessions, including game
session placement and FlexMatch matchmaking, are available with Realtime Servers (match backfill is
not yet available).

A Realtime server, once it is deployed on a fleet of hosting instances, maintains communication with
the GameLift service. The Realtime server starts a game session when it is prompted to by the GameLift
service and receives available game session and player data, including matchmaking data from the
service. If your game uses player sessions to reserve game slots or to authenticate player connections,
the Realtime server can send a validation request to the GameLift service when the player connects. A
Realtime server also reports its health status back to the GameLift service, and notifies the service when
players connect/disconnect and when a game session ends. It also responds to prompts from GameLift
to force a game session termination. This interaction with the GameLift service is fully built into all
Realtime Servers.

You have the option of adding custom logic for game session management by building it into the
Realtime script. You might write code to access server-specific objects, add event-driven logic using
callbacks, or add logic based on non-event scenarios, such as a timer or status check. For example, you
might want to or access game session objects, or trigger an action when a game session starts or ends.

How Realtime Clients and Servers Interact

During a game session, the interaction between game clients in the game is done by messaging. Game
clients use messages to exchange activity, game state, and relevant game data. Game clients send
messages to the Realtime server, which then relays the messages among the game clients. Game clients
communicate with the server using the Realtime Client SDK, which must be integrated into your game
client. The Client SDK defines a set of synchronous API calls that allow clients to connect to games, send
and receive messages, and disconnect from games. It also defines a set of asynchronous callbacks, which
can be implemented on the game client to enable the client to respond to certain events.

In addition, you can customize how clients and servers interact by adding game logic to the Realtime
script. With custom game logic, a Realtime might implement callbacks to trigger event-driven responses.
For example, when a game client notifies the server that a certain achievement is reached, the server
sends a message to other game clients to prompt an announcement.

Communication Protocol

Communication between a Realtime server and connected game clients uses two channels: a TCP
connection for reliable delivery and a UDP channel for fast delivery. When creating messages, game
clients choose which protocol to use depending on the nature of the message. Message delivery is set
to UDP by default. If a UDP channel is not set up or not available, all messages are sent using TCP as a
fallback.

Message Content

Message content consists of two elements: a required operation code (opCode) and an optional payload.
A message's opCode identifies a particular player activity or game event, while the payload provides
additional data, as needed, related to the operation code. Both of these elements are developer-defined;
that is, you define what actions map to which opCodes, and whether a message payload is needed. You
game client takes action based on the opCodes in the messages it receives.

Player Groups

Realtime Servers provides functionality to manage groups of players. By default, all players who are
connected to a game are placed in an "all players" group. In addition, developers can set up other groups
for their games, and players can be members of multiple groups simultaneously. Group members can
send messages to all players in the group or share game data with the group. One possible use for
groups is to set up player teams and manage team communication.
Realtime Servers with TLS Certificates

You can opt to create Realtime Servers fleets with TLS certificate generation turned on. GameLift generates a TLS certificate for the fleet and creates a DNS entry for each instance in the fleet. This allows your game to authenticate the client/server connection and encrypt all game client/server communication. This feature makes it possible to publish games on a range of platforms, including mobile, that require enhanced security and encrypted communication. It helps to protect your game clients (and players) from server spoofing attacks, and prevents malicious actors from hacking or monitoring data transmissions. These services are provided through the AWS Certificate Manager (ACM) and are currently available at no additional cost.

With Realtime Servers, server authentication and data packet encryption is already built into the service, and is enabled when you turn on TLS certificate generation. When the game client tries to connect with a Realtime server, the server automatically responds with the TLS certificate, which the client validates. Encryption is handled using TLS for TCP (Websockets) communication and DTLS for UDP traffic.

Customizing a Realtime Server

In its most basic form, a Realtime server performs as a stateless relay server. The Realtime server relays packets of messages and game data between the game clients that are connected to the game, but does not evaluate messages, process data, or perform any gameplay logic. Used in this way, each game client maintains its own view of the game state and provides updates to other players via the relay server. Each game client is responsible for incorporating these updates and reconciling its own game state.

Alternatively, you can customize your servers by building out the Realtime script functionality. There are many server-side processes you may choose to implement even while taking advantage of the simplicity of the Realtime Servers feature. With game logic, for example, you might opt to build a stateful game with a server-authoritative view of the game state.

A set of server-side callbacks are defined for Realtime scripts. Implement these callbacks to add event-driven functionality to your server. For example, you might:

- Authenticate a player when a game client tries to connect to the server.
- Validate whether a player can join a group when requested.
- Evaluate when to deliver messages from a certain player or to a target player, or perform additional processing in response.
- Take action, such as notifying all players, when a player leaves a group or disconnects from the server.
- Evaluate the content of game session objects or message objects and use the data.

Deploying and Updating Realtime Servers

Realtime Servers is powered by GameLift’s dedicated server resources. There is no difference in stability and security provided. As with all servers, latency can be minimized by using GameLift’s matchmaking and queues with Fleet IQ, which optimizes game session placement based on player locations.

When deploying Realtime Servers games with GameLift, the process is nearly identical to deploying traditional game servers on GameLift. You create fleets of computing resources and deploy them with your Realtime script, which contains configuration details and optional custom logic. Using GameLift, you choose the type of fleets to use, manage fleet capacity, and control how game server processes are started and run on your fleets. The detailed description of game hosting in How GameLift works (p. 3) represents game hosting with Realtime Servers as well as with custom game servers.

A key advantage Realtime Servers is the ability to update your scripts at any time. You do not need to create a new fleet to deploy an updated script. When you update a script, the new version is propagated to all hosting resources within a few minutes. Once the new script is deployed, all new game sessions
created after that point will use the new script version (existing game sessions will continue to use the original version).

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. 58).

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 the Amazon GameLift service'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.

**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. 167) and Metrics (p. 164).

**Game architecture with managed GameLift**

The diagram shown below illustrates the key components of a game architecture that is hosted using the managed GameLift solution.
Key components are described as follows.

**Game clients**

To join a game being hosted on GameLift, your game client must first find an available game session. The game client searches for existing game sessions, requests matchmaking, or starts a new game session by communicating with GameLift service. This communication is done through a backend client service in order to help game owners maintain secure control of their game servers and hosting resources. The client service makes requests to the GameLift service and in response receives game session information, including connection details, which it relays it back to the game client. The game client then uses this information to connect directly to the game server and join the game. The green arrow represents the direct connection between game client and game server during gameplay.

**Client services**

A backend client service handles communication between game clients and the GameLift service by calling the GameLift service APIs in the AWS SDK. Client services might also be used for other game-specific tasks such as player authentication and authorization, inventory, or currency control. For example, when a player joins a game, your game client might first call an authentication service to first verify the player's identity, and only then send a player slot request to the GameLift service. Relevant information, such as connection details, are relayed back to the game client.

**External services**
Your game may rely on an external service, such as for validating a subscription membership. As shown in the architecture diagram, information from an external service can be passed to your game servers (via a client service and the GameLift service) without going through the game client.

**Game servers**

Your game server software is uploaded to the GameLift service and is deployed onto hosting machines to host game sessions and accept player connections. Game servers communicate with the GameLift service by using the GameLift Server SDK, exchanging requests to start new game sessions, validate newly connected players, and report status of game sessions, player connections, and available resources. Game clients connect directly to a game server after receiving connection details from the GameLift service.

**GameLift service**

The GameLift service is the core service that deploys and manages fleets of resources to host your game servers, coordinates how game sessions are placed across your available resources, starts and stops game sessions, and tracks game server health and activity in order to maintain game availability as player traffic fluctuates. When setting up and managing hosting resources, game owners use the GameLift service APIs in the AWS SDK and CLI to upload game server builds, create and configure fleets, and manage fleet capacity. Client services start new game sessions, request matchmaking, and slot players into game sessions by calling the GameLift service APIs in the AWS SDK. Game servers that are deployed onto GameLift fleets use the GameLift Server SDK to maintain communication with the GameLift service to start and stop game sessions, report server health, exchange game and player data as needed, etc.

**Hosting management tools**

The GameLift tool set in the AWS SDK provides multiple ways for you to configure your game hosting resources, scale capacity to player demand, and monitor the current status of resources, as well as track metrics on game server performance and game and player activity. In addition, you can remotely access any individual game server for troubleshooting.

---

**Game Architecture with Realtime Servers**

The diagram shown below illustrates the key components of a game architecture that is hosted using the managed GameLift with Realtime Servers solution.
Key components are defined as follows:

Game clients

To join a game being hosted on GameLift, your game client must first find an available game session. The game client searches for existing game sessions, requests matchmaking, or starts a new game session by communicating with GameLift service. This communication is done through a backend client service in order to help game owners maintain secure control of their game servers and hosting resources. The client service makes requests to the GameLift service and in response receives game session information, including connection details, which the client service relays back to the game client. The game client then uses this information, with the Realtime Client SDK, to connect directly to the game server. Once connected, the game client can join the game and exchange game state updates with other players in the game. The green arrow represents the direct connection between game client and game server during gameplay.

Client services

A backend client service handles communication between game clients and the GameLift service by calling the GameLift service APIs in the AWS SDK. Client services might also be used for other game-specific tasks such as player authentication and authorization, inventory, or currency control. For example, when a player joins a game, your game client might first call an authentication client service to verify the player's identity, and only then send a game session request to the GameLift service. Relevant information that the client service receives from the GameLift service, such as connection details, are relayed back to the game client.
External services

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

Realtime Servers

To host game sessions, you create a Realtime Servers fleet that is configured for your game. The Realtime servers take the place of an integrated full-fledged game server; instead they run script, which you customize for your game and upload to GameLift. Realtime servers track player connections to a game session and relay game data between players to keep each player's game state in sync. They also communicate with the GameLift service to start new game sessions, validate newly connected players, and report on the status of game sessions, player connections, and available resources. When joining a game, a game client connects directly to a Realtime server after receiving connection details from the GameLift service.

GameLift service

The GameLift service is the core service that deploys and manages fleets of resources to host your Realtime servers, coordinates how game sessions are placed across your available resources, starts and stops game sessions, and tracks game server health and activity in order to maintain game availability as player traffic fluctuates. When setting up and managing hosting resources, game owners use the GameLift service APIs in the AWS SDK and CLI to upload game server builds and scripts, create and configure fleets, and manage fleet capacity. Client services start new game sessions, request matchmaking, and slot players into game sessions by calling the GameLift service APIs in the AWS SDK.

Hosting management tools

The GameLift tool set in the AWS SDK provides multiple ways for you to configure your game hosting resources, scale capacity to player demand, and monitor the current status of resources, as well as track metrics on game server performance and game and player activity. In addition, you can remotely access any individual game server for troubleshooting.
Setting Up

Tip
Learn more about ways to explore Amazon GameLift features, including Realtime Servers, using sample games (p. 27).

Topics
- Set up an AWS account (p. 18)
- Set up a role for GameLift access (p. 21)
- GameLift SDKs (p. 22)
- Tools and Resources (p. 24)
- Setting up AWS billing alerts (p. 25)
- Using Amazon GameLift in AWS Regions (p. 25)

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 GameLift

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

These policy examples illustrate how to provide full administrative access to a user.

Policy for GameLift resource permissions

The following policy example covers access to all GameLift-related resources (fleets, queues, game sessions, matchmakers, etc.). All users who manage or view these resources need this type of permissions policy.

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

Policy for GameLift resource and PassRole permissions

This policy example provides access to GameLift-related resources as above. It also allows the user to pass an IAM service role to GameLift. Not all users need to have PassRole permission; it is used to give GameLift limited ability to access resources in other services on your behalf. For example, you need this permission when calling CreateBuild with an IAM role that allows GameLift to access your build files in an S3 bucket. For more information on PassRole, see the IAM User Guide topic IAM: Pass an IAM role to a specific AWS service.

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

Simple policy examples for players

The following policy examples illustrate how to enable game clients and/or game client services with the functionality to get players into game sessions. These examples cover the key scenarios that games might use to start new game sessions and assign players to available player slots.

Policy for game session placements

This policy example is for a game client service that uses game session queues and placements to start new game sessions. Players might be added to a game session either in the initial placement request or by creating new player sessions for an existing game session.
IAM policy examples

Policy for matchmaking

This policy example is for a game client or client service that uses GameLift FlexMatch matchmaking. Players might be matched and placed into a new game session or they might be added to an existing game session through the backfill process.

Policy for manual game session placement

This policy example is for a game client or client service that creates new game sessions on specific fleets and might create new player sessions in specific game sessions. This scenario supports a game that uses the "list-and-pick" method to let players choose from list of available game sessions.
Set up a role for GameLift access

This topic refers to GameLift hosted solutions. If you use GameLift FleetIQ to optimize game hosting on your EC2 instances, see Set up your AWS account for GameLift FleetIQ.

Some GameLift features require you to extend limited access to your AWS resources. This is done by creating an AWS Identity and Access Management (IAM) role. A role specifies two things: (1) who can assume the role, and (2) which resources they can control while using the role. This topic provides guidance on how to set up a role to extend access to the GameLift service.

Users who make API calls that extend this role to GameLift must have IAM PassRole permissions, as illustrated in Simple policy examples for administrators (p. 19).

To set up an IAM role for the GameLift service

By creating a role specifically for GameLift, you define which of your AWS resources can be accessed either by the GameLift service or by your applications (such as game servers) that are running on GameLift.

Currently, a service-specific role for Amazon GameLift must be manually constructed with inline permissions and trust policies. You can update the role any time using the IAM service via the console or the AWS CLI.

1. Create an IAM role by using the AWS CLI. (The IAM console currently does not allow you to create a generic service role and add or edit policies.) See the IAM user guide topic Creating a Role for a Service (AWS CLI) for specific instructions. The role must be created under the AWS account that you use to manage GameLift, so make sure you’re using the proper AWS account credentials.

2. Create an inline permissions policy and attach it to the role. The permissions policy is where you specify what level of access that is covered by the role. You can specify access to a service or a resource, such as an Amazon S3 bucket, and you can limit permissions to specific actions. You can opt to create separate IAM roles for different sets of permissions to limit vulnerability. See these Examples of Policies for Delegating Access. Once you’ve defined your policy syntax, attach it to the service as described in the instructions linked to in Step 1.

3. Create a trust policy and attach it to the role. A trust policy specifies which AWS service(s) can assume the role. Use the following syntax:

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

   If you need to access resources in your own account and you have a multi-region fleet with locations in one of the opt-in regions, add `gamelift.opt-in-region.amazonaws.com` to the role trust policy. The following example includes the four supported opt-in regions:

   ```bash
   • gamelift.ap-east-1.amazonaws.com
   • gamelift.me-south-1.amazonaws.com
   • gamelift.af-south-1.amazonaws.com
   • gamelift.eu-south-1.amazonaws.com
   ```
4. Once you've created the role, you can view it in the IAM console. Make a note of the new role's ARN, as you may need to use it when setting up a GameLift feature.

**GameLift SDKs**

This topic describes the SDKs for use with managed GameLift solutions for custom game server builds and Realtime Servers. To learn more about other GameLift solutions, see What Is Amazon GameLift? (p. 1).

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

For detailed information on using the GameLift SDKs with your game engine, see Game Engines and Amazon GameLift (p. 43). For the latest version information on GameLift SDKs and SDK compatibility, see GameLift release notes (p. 269).

**For custom game servers**

Create and deploy 64-bit custom game servers with the GameLift Server SDK. This SDK enables the GameLift service to deploy and manage game server processes across your GameLift hosting resources. Download the Server SDK and learn about how to Add GameLift to your game server (p. 51) projects. See GameLift release notes (p. 269) for version-specific information.

**SDK support**

The GameLift Server SDK download contains source for the following versions. Build the version you need for your game; see the README files with each version for build instructions and minimum requirements.

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

**Development environments**

Build the SDK from source as needed for these supported development operating systems 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 GameLift Server SDK to create game servers that run on the following platforms:

- Windows Server 2012 R2
- Amazon Linux
- Amazon Linux 2

**For Realtime Servers**

Configure and deploy Realtime servers to host your multiplayer games, and enable your game clients to connect to them with the *GameLift Realtime Client SDK*. Game clients use this SDK to exchange messages with a Realtime server and with other game clients that are connected to the server. Download the Realtime Client SDK and learn about how to use it with your game clients (p. 71).

**SDK support**

The Realtime Client SDK contains source for the following languages:

- C# (.NET)

**Development environments**

Build the SDK from source as needed for these supported development operating systems and game engines.

- **Operating systems** – Windows, Linux, Android, iOS.
- **Game engines** – Unity, engines that support C# libraries

**Game server operating systems**

Realtime servers are deployed onto hosting resources that run the following platforms:

- Amazon Linux
- Amazon Linux 2

**For client services**

Create 64-bit client services using the AWS SDK with the GameLift API. This SDK enables client services to find or create game sessions and join players to games that are being hosted on GameLift. Download the AWS SDK or view the 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.

- C++ (SDK docs) (Amazon GameLift)
- Java (SDK docs) (Amazon GameLift)
- .NET (SDK docs) (Amazon GameLift)
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 GameLift SDKs (p. 22).

Realtime Client SDK for Amazon GameLift

For games using Realtime Servers, The Realtime Client SDK enables a game client to connect to a deployed Realtime server, join a game session, and keep its game state in sync with other players in the game. Download the SDK and learn more about making API calls with the Realtime Servers Client API (C#) Reference (p. 208).

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. Get the AWS Command Line Interface download and install it using these instructions. You can also view the complete AWS CLI Command Reference for all AWS services, including 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. 65).

Additional Resources

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

Five-click sample

Get a sample multiplayer game (client and custom game server) up and running on Amazon GameLift in under an hour. This sample gives you a jump start so you can quickly start
experimenting with Amazon GameLift tools, set up fleets and deploy the game server for hosting, adjust autoscaling, and track game activity and performance. You can find the sample in the Amazon GameLift console; from the GameLift intro page click "Test Amazon GameLift", or from any other console page select "Sample game" in the Amazon GameLift navigation menu.

Amazon GameLift forum

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

GameTech blog

Watch the 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.

AWS Samples on GitHub

Find a large collection of code samples for all AWS services posted in AWS Samples.

Getting Started tutorials

Use the 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.

Amazon GameLift product information

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

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.

Setting up AWS billing alerts

You may want to set up and configure a billing alert to notify you of billing events. For more information, see Creating a Billing Alarm.

In addition to receiving billing alerts, you can view your current estimated bill for GameLift on the Billing and Cost Management console at https://console.aws.amazon.com/billing/. This will help you review your resource consumption, make decisions about future usage, and determine your scaling needs.

To avoid incurring unnecessary charges, you may want to scale down your fleet (p. 131) when not in use.

Using Amazon GameLift in AWS Regions

GameLift is available in multiple AWS Regions. For a complete list, see Amazon GameLift endpoints and quotas. A global AWS account allows you to work with resources in most Regions.

GameLift hosting

GameLift fleets reside in the AWS Region where they are created, which is called the fleet's "home" Region. The work of managing fleet activity, including auto-scaling, is done in the home Region. All fleets deploy their instances in the home Region.
Multi-location fleets are capable of deploying instances to other Regions in addition to the fleet’s home Region. These fleets are configured with additional "remote" locations. With multi-location fleets, you can manage capacity for each location individually, and you can place game sessions by location. Multi-location fleets can have remote locations in any AWS Region that is supported by GameLift, with the exception of China. They can be created in the following GameLift-supported AWS Regions: US East (N. Virginia), US West (Oregon), EU Central (Frankfurt), EU West (Ireland), Asia Pacific Southeast (Sydney), and Asia Pacific Northeast (Seoul and Tokyo).

For game session placement, game session queues can be created in any GameLift-supported Region. A queue can place new game sessions with fleets in any other Region or location. The work of game session placement is done in the Region where the queue was created and resides. Requests for new game sessions that reference a queue are routed to the queue, wherever it resides.

GameLift FlexMatch

For FlexMatch, all GameLift-supported Regions can host match-generated game sessions. FlexMatch resources, including matchmaking configurations and rule sets, can be created in the following Regions: US East (N. Virginia), US West (Oregon), EU Central (Frankfurt), EU West (Ireland), Asia Pacific Southeast (Sydney), Asia Pacific Northeast (Seoul and Tokyo), and China (Beijing). The work of processing matchmaking requests is done in the Region where the matchmaking configuration and rule set were created and reside. Requests for matchmaking that reference a matchmaking configuration are routed to it, wherever it resides.

GameLift in China

When using Amazon GameLift for resources in the China (Beijing) Region, which is operated by Sinnet, keep in mind that you must have a separate AWS (China) account. In addition, there are some differences in how Amazon GameLift is implemented, including how it handles cross-region interactions and dependencies. In particular, multi-location fleets cannot be supported. For more information on using Amazon GameLift in the China (Beijing) Region, see these resources:

- AWS in China
- Amazon GameLift in China
Getting Started with Amazon GameLift

The resources in this section help you get started with the managed GameLift solutions for custom game servers and Realtime Servers.

Topics
- Explore Amazon GameLift (p. 27)
- Get Started with Custom Servers (p. 28)
- Get Started with Realtime Servers (p. 41)

Explore Amazon GameLift

Looking to experiment with Amazon GameLift features before diving in with your own game? Try out these sample experiences. The console samples give you hands-on experience with game hosting in the GameLift console. The source code example and walkthrough shows you how to prepare a game for hosting using Realtime Servers.

Custom Game Server Sample (Console Experience)

This sample experience quickly gets you working with a live game on GameLift. Upload a sample game build, create a fleet to run the game server, and connect to it from a sample game client. You can start up multiple game clients and play to generate hosting data. Once you have some data, explore the GameLift console to view your hosting resources, track metrics, and experiment with ways to scale hosting capacity.

To access the sample wizard, sign into the GameLift console, open the Amazon GameLift menu, and select Custom game server sample.

About the sample game

The sample game is developed using the Amazon Lumberyard game engine. To run the game client, you need a Windows 7 64-bit system and 300 MB of space. See additional requirements.

Realtime Servers Sample Game (Full Source)

Mega Frog Race is a complete multiplayer game sample with source code. Follow a hands-on tutorial that walks through how to prepare the sample to run online using GameLift Realtime Servers. This sample is a good way to better understand how to get your game client ready to work with GameLift Realtime Servers. You can also use it as a starting point to experiment with other GameLift features such as FlexMatch.

To read the hands-on tutorial, see the GameTech blog post Creating Servers for Multiplayer Mobile Games with Just a Few Lines of JavaScript.

To get the source code, go to the GitHub repository.

The source material for the MegaFrogRace sample includes all the elements to deploy the multiplayer game hosted with Realtime Servers.
• Game client – Source code for the Unity-created C++ game client. It illustrates how to get game session connection information from GameLift, connect to a Realtime server, and exchange game updates with other players via the Realtime server.

• Client service – Source (in Node-based JavaScript) for an AWS Lambda function that manages direct API calls to the GameLift service. When called by the game client, the service makes requests to find or start new game sessions and assign players, and then returns connection details back to the game client.

• Realtime script – Source script file (in Node-based JavaScript) that configures a fleet of Realtime servers for the game. This script includes the minimum configuration to enable Realtime servers to communicate with the GameLift service and to start and stop game sessions. It also includes some custom logic for the sample game.

Get Started with Custom Servers

This roadmap outlines the key steps to setting up your multiplayer games with custom game servers to run with the managed Amazon GameLift solution for custom game servers. If you're interested in using GameLift Realtime Servers, which let you deploy your game client with our ready-to-deploy game servers, see Get Started with Realtime Servers (p. 41). To learn about other GameLift solutions, see What Is Amazon GameLift? (p. 1).

Tip
Learn more about ways to explore Amazon GameLift features, including Realtime Servers, using sample games (p. 27).

New to GameLift? We recommend that you read What Is Amazon GameLift? (p. 1). If you're unsure whether GameLift supports your operating systems and development environments, see the topics GameLift SDKs (p. 22) and Game Engines and Amazon GameLift (p. 43).

Before you start integrating with GameLift, you need to have an AWS account and configure it for GameLift. Learn more at Set up an AWS account (p. 18). All essential tasks related to creating and managing your game servers can be done using the GameLift console, but you might also want to Get and install the AWS Command Line Interface tool.

To use custom game servers with GameLift

1. **Prepare your custom game server for hosting on GameLift**
   - Get the Amazon GameLift Server SDK and build it for your preferred programming language and game engine. If you're using the Amazon Lumberyard game engine, a version of the SDK is built in. See the GameLift SDKs For custom game servers (p. 22) and Game Engines and Amazon GameLift (p. 43).
   - Add code to your game server project to enable communication with the GameLift service. A game server must be able to notify GameLift about its status, to start and stop game sessions when prompted, and to perform other tasks. See Add GameLift to your game server (p. 51).

2. **Prepare your game client to connect to GameLift hosted game sessions.**
   - Set up a game client to communicate with GameLift to start game sessions and place players into games when prompted by a game client.
   - Add the AWS SDK to your client service project. See the GameLift SDKs For client services (p. 23).
   - Add functionality to retrieve information on game sessions, place new game sessions, and (optionally) reserve space for players on a game session. See Add Amazon GameLift to Your Game Client (p. 58). Recommended: Use game session placements to take advantage of FleetIQ and optimize resource usage and player experience. This option is required if you're using FlexMatch.
• (optional) Enable the client service to request player matchmaking using FlexMatch. Learn more in FlexMatch integration roadmap.

• Enable your game client to connect directly with a hosted game session. Add code to acquire connection information for a game session and (optionally) a reserved player session. Use this connection information and a unique player ID to communicate with the game server and join the game. See Join a Player to a Game Session (p. 60).

3. Test your GameLift integration.

• Use Amazon GameLift Local to test your game client and game server integration using a version of the GameLift service that's running locally. You can use this tool to test your integration without uploading game builds or setting up fleets. Use GameLift Local to verify that your game components are communicating with the GameLift service and test core functionality. See Testing Your Integration (p. 65).

4. Build a fleet of computing resources to host your game.

• Package and upload your custom game server build to the GameLift service. Be sure to upload your build to each Region where you plan to deploy your game. See Upload a custom server build to GameLift (p. 97).

• Design a fleet configuration for your game. Decide, for example, the type of computing resources to use, which Regions to deploy to, whether to use queues, and other options. See GameLift fleet design guide (p. 106).

• Create fleets and deploy them with your custom game server. When a fleet is active, it is ready to host game sessions and accept players. See Setting up GameLift fleets (p. 106).

• Experiment with GameLift fleet configuration settings and refine as needed to optimize usage of your fleet resources. Adjust the number of game sessions to run concurrently on each instance, or set game session activation limits. See GameLift fleet design guide (p. 106). See also how to Remotely access GameLift fleet instances (p. 126).

• Create a queue to manage how new game sessions are placed with available hosting resources. See Design a game session queue (p. 139).

• Enable automatic scaling to manage your fleet's hosting capacity for expected player demand. See Scaling GameLift hosting capacity (p. 129).

• (optional) Set up a FlexMatch matchmaker with a set of custom matchmaking rules for your game. Learn more in FlexMatch integration roadmap.

Note
After you create your queues, you must update your client service to use the correct queue ID when requesting game session placements and/or matchmaking.

After you've fully integrated 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 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. 161).

For more information about implementing a custom game server, see the following topics:

Topics
• Hosting Components Overview (p. 30)
• Prepare Your Custom Game Server (p. 31)
• Test the Integration Locally with Amazon GameLift Local (p. 31)
• Plan Your Global GameLift Resources Deployment (p. 32)
• Deploy your Amazon GameLift Resources (p. 32)
Hosting Components Overview

This section provides a high-level overview of the different components in a custom server. It also provides an example game session placement flow.

**GameLift FlexMatch matchmaking configuration and rule set** — If you want to use the matchmaking capabilities of GameLift, you must define a matchmaking configuration, provide a JSON-document based rule set, and use the matchmaking APIs of GameLift from your backend service.

**GameLift fleets** — A fleet contains EC2 instances hosted by GameLift. A fleet uses the configuration and scaling logic that you define to run your game server build. A fleet can be used directly without a queue. Multiple fleets can be associated with a GameLift queue. A common pattern is to use Spot Fleets configured with your preferred locations, and a backup on-demand fleet with the same locations. Using multiple Spot Fleets of different instance types reduces the chance of on-demand placement.

**GameLift queue** — To place game sessions across GameLift fleets globally, you must create a GameLift queue. The queue contains a prioritized order of your fleets as well as your latency requirements. The queue places game sessions on the lowest cost and lowest latency fleet. With the introduction of multi-Region fleets, you do not need to separate fleets per Region. The queue can manage placement across multiple locations within a single fleet.

One fleet can span across GameLift supported Regions. You don't need to run different fleets in all the different Regions when the fleets don't need custom configuration or builds. Since fleets are managed in your home Region, we recommend that you have fleets in two different home Regions for redundancy.

**Game backend service** — In addition to GameLift, you always need to host a backend service for your game instead of accessing GameLift directly from the game client. The backend uses the GameLift APIs to initiate matchmaking, request game session placement, and so on. Popular options for game backend hosting include serverless backends with Amazon API Gateway and AWS Lambda, or container-based backends with Elastic Load Balancing and Amazon Elastic Container Service. You can store player data in a managed NoSQL database like Amazon DynamoDB or a managed SQL database like Amazon Relational Database Service.

The following diagram shows an example game session placement flow. In the diagram, fleets are deployed to multiple Regions. All of the fleets are registered to a single GameLift queue. You can also use a single fleet and define multiple Regional locations for the fleet for the same result.
Prepare Your Custom Game Server

The Amazon GameLift Server SDK is available for C++, C# (Unity), and as an Unreal Engine plugin. You will need to integrate the SDK to your game server build and implement a set of callbacks to communicate with the GameLift service for session management.

Download the GameLift Managed Game Servers SDK and build it for your preferred programming language and game engine. If you are using the Amazon Lumberyard game engine, a version of the SDK is built in. For more information about GameLift SDKs for custom game servers, see For custom game servers (p. 22). For detailed information on using the GameLift SDKs with your game engine, see Game Engines and Amazon GameLift (p. 43). For the latest version information on GameLift SDKs and SDK compatibility, see GameLift release notes (p. 269).

Add code to your game server project to enable communication with the GameLift service. A game server must be able to notify GameLift about its status, to start and stop game sessions when prompted, and to perform other tasks. For more information, see Add GameLift to your game server (p. 51).

Test the Integration Locally with Amazon GameLift Local

Amazon GameLift Local is a Java application included with the managed game servers SDK. You can run it locally to simulate the integration between your game server and the GameLift service. Uploading a build and deploying fleets can take time. We recommend you test SDK integration with GameLift Local to identify issues before uploading your build and deploying fleets.

With GameLift Local, you can verify that your game components are communicating with the GameLift service and test core functionality. To simulate the GameLift environment as closely as possible, you can run an Amazon Linux 2 Docker container locally or test on an Amazon EC2 instance. For more information about how to test locally, see Testing Your Integration (p. 65).
Plan Your Global GameLift Resources Deployment

Use the following tips to help plan your global GameLift resources deployment.

- A single Amazon GameLift fleet has a home Region (the Region you deploy it to), but it can deploy resources to any number of GameLift supported Regions. Select Regions based on where your players are located and your latency needs. For example, a turn-based game might not have low latency requirements and you could use a single Region per continent where a first-person shooter requires a very low latency experience and you likely want to use all the Regional locations, especially if your player base is global.

- If there are any Regional differences in the matchmaking logic, you will need to deploy your FlexMatch configurations and queues in a single Region for each geographical area of your players. More commonly, the configurations are the same for all Regions. GameLift queues have a hard limit of 100 transactions per second, so achieving a higher rate is another reason to host multiple queues.

- FlexMatch configurations and queues are Regional resources. To do cross-Region failover for fault tolerance, you need to deploy a backup FlexMatch configuration and queue in a separate Region and implement logic in your backend service to fail over in the rare case of Region-wide issues.

- For China, you commonly deploy a completely separate set of resources including fleets, queues, and FlexMatch matchmaking configurations. This is because connectivity between China and the rest of the world will not in most cases meet your latency and reliability requirements. For more information about getting access to the China Regions, see Getting Started with AWS Services in China.

Deploy your Amazon GameLift Resources

To deploy your Amazon GameLift resources, complete the following tasks:

- **Package and upload your custom game server build to the GameLift service.** You only need to upload to the home Region of your fleet, and it will automatically be distributed to any additional locations that you choose. For more information, see Upload a custom server build to GameLift (p. 97).

- **Design a GameLift fleet configuration for your game.** Decide, for example, the type of computing resources to use, which locations to deploy to, whether to use queues, and other options. For more information, see GameLift fleet design guide (p. 106).

- **Create fleets and deploy them with your custom game server.** When a fleet is active, it is ready to host game sessions and accept players. For more information, see Create a new GameLift fleet (p. 112).

- **Experiment with GameLift fleet configuration settings and refine as needed to optimize usage of your fleet resources.** Adjust the number of game sessions to run concurrently on each instance, or set game session activation limits if your activation uses CPU and memory resources extensively. For more information, see GameLift fleet design guide (p. 106). If you need to debug issues on the instance itself, see Remotely access GameLift fleet instances (p. 126). See amazon-gamelift-remote-plus for a tool that simplifies the remote access.

- **Create a queue to manage how new game sessions are placed with available hosting resources and to implement your Spot Instance strategy.** For more information, see Design a game session queue (p. 139).

- **Enable automatic scaling to manage your fleet's hosting capacity for expected player demand.** For more information, see Scaling GameLift hosting capacity (p. 129).

- **[optional] Set up a FlexMatch matchmaker with a set of custom matchmaking rules for your game.** For more information, see FlexMatch integration roadmap.

To reduce any manual errors and to simplify the global deployment of your GameLift resources, we recommend you use Infrastructure as Code to define all of the resources. GameLift hosting supports
AWS CloudFormation so you can define your resources with YAML or JSON templates and deploy one or more stacks of those resources across multiple Regions. You can parameterize the template for any deployment-specific configurations.

In addition to Infrastructure as Code, we recommend you use a Continuous Integration / Continuous Delivery (CI/CD) tool like AWS CodePipeline or open-source tools such as Jenkins to manage the deployment of your AWS CloudFormation stacks. This will help you deploy either automatically or with an approval step whenever your game server binary is built.

Common automated steps of GameLift resources deployment for a new game server version with a CI/CD tool include:

- Building and testing your game server binary
- Uploading the binary to GameLift using the AWS SDK
- Deploying new GameLift fleet(s) using the new build with AWS CloudFormation
- After the new fleet is successfully deployed, removing the previous version fleet(s) from your GameLift queue and replacing with the new fleet(s)
- After the old fleet(s) have successfully terminated all game sessions, deleting the AWS CloudFormation stack(s) of these fleet(s)

Alternatively, you can use the AWS Cloud Development Kit (CDK) to define your GameLift resources with TypeScript, Python, or other supported programming language. You can use the AWS CLI to deploy and update the resources with generated AWS CloudFormation templates.

**Design your Backend Service**

You should always implement a backend service that authenticates your players and communicates with the Amazon GameLift APIs. Implementing a custom backend service has the following benefits:

- You can customize authentication for your players.
- You control how matching and game session placement are initiated.
- The backend service is authoritative. You can use your player database for player attributes such as skill rating for matchmaking instead of trusting the client.

You can build a serverless backend using services such as Amazon API Gateway, AWS Lambda, and Amazon DynamoDB to minimize your operational efforts, scale seamlessly to your needs, and pay for...
exactly what you use. In case you need more control, you can build container-based backends using an
Application Load Balancer and Amazon Elastic Container Service or Amazon Elastic Kubernetes Service.
In addition to containers, you can always host your backend application directly on virtual machines
with EC2 instances behind an Application Load Balancer and leverage Auto Scaling groups to scale the
capacity. You can use AWS AppSync to build a GraphQL based backend for your game using WebSockets.

For an example implementation with a serverless backend, see GameLift Example with Serverless
Backend. To learn more about building an API with Lambda integrations, see Build an API Gateway REST
API with Lambda integration. For step-by-step instructions on using DynamoDB to model player data,
see Introduction: Modeling Game Player Data with Amazon DynamoDB.

Authenticating your Players

You can use Amazon Cognito to authenticate your game clients. Amazon Cognito can be used to manage
the whole lifecycle and properties of your player identities with Amazon Cognito user pools, or you can
integrate to your custom identities and external identities like Google, Apple ID, or Facebook with the
more lightweight Amazon Cognito identity pools option that is used in the examples.

You can also build a custom identity solution or use a third-party identity solution that you host on AWS
or is managed for you. You can use Lambda authorizers for custom authorization logic with Amazon API
Gateway.

GameLift hosting provides player session IDs. You can use a player session ID to make sure that the
correct player has joined the correct game server. For GameLift FleetIQ, you must implement this type of
validation yourself.

Additional resources:
- Using Cognito Identity Pools
- Using Getting Started with Cognito User Pools
- How to Set Up Player Authentication for Game Servers with Amazon Cognito

Standalone Game Session Servers with a Serverless Backend

You can use a serverless backend with your standalone game session servers in GameLift.

The following architecture shows how to use a serverless implementation for a backend service where
the game client authenticates against Amazon API Gateway with an Amazon Cognito identity to access
backend services running in Lambda functions. The backend Lambda functions communicate with
GameLift APIs to request game sessions through FlexMatch. Matchmaking events are received through
an Amazon SNS event and stored in an Amazon DynamoDB NoSQL database by a Lambda function.
Using this architecture, the backend can check the status of the matchmaking tickets from a highly
scalable database instead of directly accessing the GameLift APIs.
The diagram shows the process of getting players into games running on the GameLift game servers. It includes the following steps:

1. The game client requests an Amazon Cognito identity from an Amazon Cognito identity pool. This can be connected optionally to external identity providers such as Google, Apple, Xbox Live, and Steam identities.
2. The game client receives temporary access credentials and requests a game session through an API hosted with API Gateway by signing the request with these credentials.
3. API Gateway invokes a Lambda function.
4. The Lambda function requests player data from a DynamoDB NoSQL table. The Amazon Cognito identity can be used to securely request the correct player data because the authenticated identity is provided in the request context data.
5. With the correct player data for any additional information like player skill level, the Lambda function requests a match through FlexMatch matchmaking. You can define a FlexMatch matchmaking configuration with JSON-based configuration documents. The game client can send latency data against the different Regions and can allow latency-based matchmaking.
6. After FlexMatch matches a suitable group of players with suitable latency to a Region, it requests a game session placement through a GameLift queue. The queue has fleets with one or more Region locations registered to it.
7. When the session is placed on one of the fleet's locations, an event notification is sent to an Amazon SNS topic.
8. A Lambda function will receive the Amazon SNS event and process it.
9. If the ticket is a MatchmakingSucceeded event, the Lambda function writes the result to DynamoDB with the server port and IP address. A time-to-live (TTL) value is used to make sure that the tickets are deleted from DynamoDB when they are not needed anymore.
10. The game client makes a signed request to API Gateway to check the status of the matchmaking ticket on a specific interval.
11. API Gateway invokes a Lambda function that checks the matchmaking ticket status.
12. The Lambda function checks DynamoDB to see if the ticket has already succeeded. If it has succeeded, the Lambda function sends the IP address, port, and the player session ID back to the client. If not, it sends a response that the match is not ready yet.
The game client connects to the game server using TCP or UDP by using the port and IP address provided by the backend. It sends the player session ID to the game server and the game server can validate it using the GameLift Server SDK.

### Standalone Game Session Servers with WebSockets-based Backend

The serveless architecture can be modified to use API Gateway WebSockets with GameLift. Using this architecture, you can make matchmaking requests with WebSockets, and push notifications of matchmaking completion (or failure) using server-initiated messages over WebSockets. This architecture improves performance by having a two-way communication between the client and the server. The matchmaking result is received immediately after success. WebSockets increases complexity because the WebSocket connections from clients need to be managed with a set of AWS Lambda functions (OnConnect and OnDisconnect). A database like Amazon DynamoDB is needed to store the connections.

The diagram shows the process of getting players into games running on the GameLift game servers using WebSockets. It includes the following steps:

1. Game client requests an Amazon Cognito identity from an Amazon Cognito identity pool. This can be connected optionally to external identity providers such as Google, Apple, Xbox Live, and Steam identities.
2. Game client signs a WebSocket connection to the API Gateway with the Amazon Cognito credentials.
3. API Gateway calls the OnConnect Lambda function on the connection. The connection information is stored in an Amazon DynamoDB table.
4. Game client sends a message over the WebSocket connection to request a session.
5. A Lambda function receives the message and requests a match through FlexMatch matchmaking. FlexMatch allows you to define the matchmaking configuration with JSON-based configuration documents. The game client can send latency data against the different Regions in addition to allowing latency-based matchmaking.
6. After FlexMatch has matched a suitable group of players with suitable latency to a Region, it will request a game session placement through a GameLift queue that has fleets with one or more Region locations registered to it.

7. When the session is placed on one of the fleets, an event notification is sent to an Amazon SNS topic.

8. A Lambda function will receive the Amazon SNS event and process it.

9. If the event is a `MatchmakingSucceeded` event, the Lambda function requests the correct player connection from DynamoDB and uses the API Gateway APIs to send a message directly to that player over the WebSocket. The game client does not need to actively poll the status of matchmaking in this model.

10. Game client receives the port and IP address of the game server as well as the player session ID through the WebSocket connection.

11. The game client connects to the game server using TCP or UDP using the port and IP address provided by the backend. It also sends the player session ID to the game server and the game server can validate it using the GameLift Server SDK.

Define and Implement Metrics and Logs Solutions

You can use data collected from your GameLift game servers and resources to help you spot anomalies and to make sure that everything is working as expected. You can also use metrics to help you improve performance.

Key areas of observability include:

- **GameLift service metrics** — GameLift provides you Amazon CloudWatch metrics on resources including game servers, fleets, queues, and FlexMatch. These metrics are available in the GameLift console as well as the Amazon CloudWatch console. You can create CloudWatch alarms for any activity. Alarms can be received via email, SMS, and other SMS enabled devices. You can also create CloudWatch dashboards to visualize important GameLift metrics.

- **Game server metrics** — GameLift does not have visibility into your game server internals and cannot provide you metrics outside of the session-related ones. You can send custom metrics to CloudWatch directly from your game server by using the CloudWatch agent, enabling the StatsD agent within it, and using a StatsD client in your game to send the metrics. You can also use the fleet IAM role and AWS SDK to send metrics directly to CloudWatch. An example of how to configure this can be found in the GameLift Example for Unity with Serverless Backend reference solution. It includes a sample CloudWatch agent configuration and code for a simple C# StatsD client.

- **Game server logs** — Use the GameLift server SDK configuration to configure your game server log files on the game server. These logs are available from the GameLift console after a game session has finished. CloudWatch Logs can be used for a real-time log management solution with the possibility to create metrics from your log data, and alarms from those metrics. Logs can be configured with the CloudWatch agent. You will need a unique identifier for all log streams to avoid collisions. GameLift does not manage log rotation so you will need to implement that yourself. The CloudWatch agent supports different log rotation configurations. For more information, see the GameLift Example for Unity with Serverless Backend reference solution.

Prepare for Launch

When you start preparing for your game launch, you should review the Testing Phase Checklist (p. 39) and the Launch Phase Checklist (p. 40). At minimum, you should consider the following:

- Raise Amazon GameLift and other AWS service quotas to ensure that the live environment can scale up to the maximum size of your expected peak. For more information about GameLift service quotas, see https://docs.aws.amazon.com/general/latest/gr/gamelift.html.
Checklists

You can use these checklists to validate different phases of development of your game server. In the checklists, Items marked [Critical] are critical for your production launch.

Topics

• Onboarding Phase Checklist (p. 38)
• Testing Phase Checklist (p. 39)
• Launch Phase Checklist (p. 40)
• Post-launch Phase Checklist (p. 41)

Onboarding Phase Checklist

Use the following checklists to keep track of onboarding phase items. Items marked [Critical] are critical for your production launch.

• [Critical] Make sure that the game clients (game executable running on the player machines) are not making AWS SDK calls directly to Amazon GameLift APIs. Requests should be authenticated through a backend service.
• [Critical] Make sure that you are carefully managing any keys or secrets that are used by the servers and that a rotation schedule is in place. Compromised server processes are unlikely, but processes should not have access to the keys directly. Best practice is to use IAM roles that you provide to GameLift server instances for access to other AWS resources.
• [Critical] For a fault tolerant architecture, plan for a failover to another Region for FlexMatch and queues. If you preconfigure this multi-Region setup, the fleets can be destinations on both queues. In case of a Region-wide issue, your backend should fail over from one matchmaker to the other automatically.
• Plan the roll-out of fleets to your target Regions, with queue and fleet structure. A single fleet can have multiple locations spanning across any number of GameLift supported Regions. Automate the deployment process. You can use Infrastructure as Code with AWS CloudFormation and AWS CDK to implement the automation of your deployment.
• Build a load test client. Ideally, automate some level of load testing as well. Load testing can be done, for example, with a headless version of your game client that mimics player actions.
• Perform load testing to see where your API throttles are. You may be calling the API at a much higher rate than you expect.
Test your system under gradually increasing loads with alpha, beta, and soft launch tests. Most sensitive to extremely high loads are the matchmaking, game session placement, and game session creation operations. Running longer game sessions or reusing game sessions for automated rematching, where possible, improves these dynamics considerably. For example, 100,000 concurrent users (CCU) 1v1 with 1 minute matches creates 833 new games per second (high) and 100,000 CCU 4v4 with 4 minute matches and 2X reuse of sessions creates 13 new games per second (good). Spread game creation between Regions, if possible, if you are expecting very high amounts of traffic. If you have unavoidably high requirements, contact AWS Support for help.

Collect plenty of data. Logs and analytics are your best actionable data, so think carefully about what you need to collect and how to effectively manage and analyze that data. It will drive operational management, design changes, future technical development, business intelligence, customer satisfaction, and product longevity throughout the whole game lifecycle.

If the game servers write logs to Amazon EBS volumes, check that these are eventually deleted (e.g. an hour or so later), even if they are collected and uploaded by GameLift. Use a different directory or log file name for each server process and each game session. GameLift does not manage your log file lifecycle, and logs can exhaust Amazon EBS storage on the instance. Instances can also be removed at any time, causing you to lose some logs. To mitigate this, use CloudWatch Logs service, an Elastic Load Balancing stack, or other log aggregation service. Standardize the log data so that it can be read and understood in context, in real time, and it is easy to set alarms for different failure conditions.

Testing Phase Checklist

The following checklists to keep track of testing phase items. Items marked [Critical] are critical for your production launch.

- **[Critical]** Raise GameLift and other AWS service quotas to ensure that the live environment can scale up to the maximum size, which is at least the sum of the maximum instances for all fleets and fleet locations in the Region. You should also fill out the launch questionnaire. It is located in the AWS Management Console under GameLift | Service Limits. If you send it to the address given, it will help the GameLift team better understand your API limit requirements.

- **[Critical]** Check that the ports that are open on live fleets match the range of ports that could possibly be used by any server on the fleet instances and are open to 0.0.0.0/0. For example, if servers listen to ports in the range 38200 to 38300, these must be open to the world. GameLift service blocks ports 1-1025.

- **[Critical]** Check that RDP port 3389 and SSH port 22 are closed or limited to a small address range reflecting the developer’s public IP address range or ranges. These ports (and any unused ports) should not be open to 0.0.0.0/0 under any circumstances.

- Make sure that your DevOps team has enough visibility into your system, and effective command and control tools and protocols. Document what you think could fail, and what protocols and procedures will be used to mitigate the failure. These protocols and procedures are sometimes referred to as runbooks. Automate the procedures if possible.

- Check that the number of servers running on an instance at full usage are within the capabilities of the server instance type. Also, if there are busy activity periods during non-core gameplay, like loading a map, check that one or more games loading (typically no players attached) does not impact running gameplay sessions. Decrease process or thread priority from within the game server as necessary during these operations. You can also limit the number of concurrent game session activations on an instance for a GameLift fleet.

- Do not let the fleet scaling policy under-provision instances. Tune the scaling policy to be more conservative at first, providing a bit more idle capacity than you think you will need. You can optimize for cost later. Consider the use of target-based scaling policy with 20% idle capacity.

- Check that the FlexMatch rules contain a latency rule (if you are passing latency data) so that players who are logically close to the same Region typically play together. Test how this behaves under load with synthetic latency data from your load test client. If you use latency data in one part of the system,
then use it everywhere that it can be used including FlexMatch, game session queues, and backfill requests. If you do not, the results are undefined.

- Make sure that your player authentication and infrastructure that’s related to getting players in games (such as your own custom matchmaking) can scale effectively to meet demand. Make sure you load test these backend services as well.
- Check that the server, if left running for several days, is still able to accept connections. For example, leaving a C# `TcpListener` open for days will eventually stop being able to receive and respond to connections.
- If you are using CloudWatch Logs or CloudWatch custom metrics, set CloudWatch alarms for messages indicating severe or fatal problems on the server fleet. Simulate some failures if practical and test the runbooks.
- Check that Spot Instances are typically available for the Regions and instance types you are using. If, in the spot history graph in the GameLift console, the Spot prices approach the on-demand price, this is a prime indicator that the Spot Instances will not be available for FleetIQ.
- Raise your AWS Support level to business or enterprise level to ensure that AWS can respond to you in the case of problems or outages outside of office hours. Your account team has no access to your fleets and cannot help you if your servers stop working.
- Create your live environment separate from development on another AWS account. Check that each queue in use has an on-demand fleet with locations in all your required Regions that it must serve. Understand the rules for how queues place games (which is different depending on whether there is latency data present) and create your live on-demand and Spot Fleets accordingly using your final executables.
- Practice updates to the live environment servers. Be clear on your upgrade strategy, and how you will revert if you discover a problem with a new version. Consider using GameLift aliases instead of fleet IDs in your system. This is mandatory if you are not using GameLift game session queues.
- Validate with a load test that GameLift fleets in your live environment are available and performant.
- Validate that the correct group of people have IAM access to the live fleets and other GameLift and AWS resources. Restrict access of other personnel with IAM policies.

## Launch Phase Checklist

Use the following checklists to keep track of launch phase items. Items marked [Critical] are critical for your production launch.

- [Critical] Set the fleet protection policy to full protection on all live fleets so that scaling down cannot evict an active game session.
- [Critical] As you are ready for players to start joining, pre-warm the fleets by scaling them up to a level that can accommodate the launch spike. You can set the minimum amount relatively high for your launch and reduce it when you have more stable traffic. Optionally, to scale up in preparation for the release, you can disable automatic scaling and manually set a desired fleet capacity. When the traffic has stabilized, you can re-enable automatic scaling. This helps keep your minimum instance amount at a lower level through the launch.
- [Critical] Set fleet maximum sizes sufficiently high to accommodate at least peak anticipated demand. We recommend that you include a good margin, typically double, for unanticipated demand.
- Monitor everything very carefully as the first players arrive. Encourage the whole team to participate in the event in a war room.
- Matchmakers, including FlexMatch, tend to behave less optimally with small player pools and their behavior changes subtly with scale and global player distribution, so monitor particularly whether matching times are acceptable.
- Monitor that player latency is within tolerable limits, and players are getting a good experience.
Post-launch Phase Checklist

Use the following checklists to keep track of launch phase items. Items marked [Critical] are critical for your production launch.

- Tune scaling rules so that you are able to minimize idle capacity and, at the same time, supply all players with games.
- Measure the latency for your players and modify FlexMatch rules or roll-out in additional Region locations based on your latency requirements.
- Optimize the server executable, as its performance efficiency has a direct effect on the fleet costs. Increasing the number of server processes per instance allows you to run more game sessions with the same infrastructure.
- Use your analytics data to drive continued development, improving player experience and game longevity, and to optimize monetization.

Get Started with Realtime Servers

This roadmap outlines the key steps to getting your multiplayer game clients up and running with the managed GameLift solution with Realtime Servers. If you have a game with a custom game server, see Get Started with Custom Servers (p. 28). To learn about other GameLift solutions, see What Is Amazon GameLift? (p. 1).

New to Realtime Servers or unsure about whether this feature is appropriate for your game? We recommend that you read How Realtime Servers Work (p. 8).

Note
If you're familiar with how to integrate and deploy games with GameLift, here's a quick summary of what's different with Realtime Servers:

- Create and upload a Realtime script with optional game logic to run game sessions on Realtime Servers instances. You no longer need to develop a custom game server and integrate it with the GameLift Server SDK.
- When creating a fleet to host your game sessions, deploy it with the Realtime script instead of a game server build.
- Integrate your game client with the Realtime Client SDK to manage connections to game sessions.

Before you start integration, you need to have an AWS account and configure it for GameLift. Learn more at Set up an AWS account (p. 18). All essential tasks related to creating and managing your game servers can be done using the GameLift console, but you may also want to Get and install the AWS Command Line Interface tool.

1. Create a Realtime script for hosting on GameLift.
   - Create a Realtime script with your server configuration and optional custom game logic. Realtime Servers are already built to start and stop game sessions, accept player connections, and manage communication with the GameLift service and between players in a game. There are also hooks that allows you to add custom server logic for your game. Realtime Servers is based on Node.js, and server script is written in JavaScript. See Creating a Realtime Script (p. 74).

2. Build a fleet of computing resources to host your game.
   - Upload the Realtime script to the GameLift service. Be sure to upload your script to each region where you plan to deploy your game. See Upload a Realtime Servers script to GameLift (p. 103).
Design a fleet configuration for your game. Decide, for example, the type of computing resources to use, which regions to deploy to, whether to use queues, and other options. See GameLift fleet design guide (p. 106).

- Create Realtime Servers fleets and deploy them with the Realtime script. Once a fleet is active, it is ready to host game sessions and accept players. See Setting up GameLift fleets (p. 106).
- Experiment with GameLift fleet configuration settings and refine as needed to optimize usage of your fleet resources. Adjust the number of game sessions to run concurrently on each instance, or set game session activation limits. See GameLift fleet design guide (p. 106). See also how to Remotely access GameLift fleet instances (p. 126).
- Create a queue to manage how new game sessions are placed with available hosting resources. See Design a game session queue (p. 139).
- Enable auto-scaling to manage your fleet’s hosting capacity for expected player demand. See Scaling GameLift hosting capacity (p. 129).
- (optional) Set up a FlexMatch matchmaker with a set of custom matchmaking rules for your game. Learn more in the FlexMatch integration roadmap.

3. Prepare your game client to join GameLift-hosted game sessions.

- Create a mechanism to assign unique player IDs for use with GameLift.
- Set up a client service to send requests to the GameLift for new game sessions and to reserve space for players in existing game sessions. See Add Amazon GameLift to Your Game Client (p. 58).
- (optional) Enable the client service to request player matchmaking using FlexMatch. Learn more in the FlexMatch integration roadmap.
- Enable your game client to connect directly with a hosted game session that is running on a Realtime server and exchange information through messaging. See Integrating a Game Client for Realtime Servers (p. 71).

Once you’ve fully integrated GameLift and Realtime Servers into your game components, it’s a matter of managing your game server fleets for optimal availability and performance over the long term. Use 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. 161).
Preparing Games for Amazon GameLift

To get your multiplayer game up and running on the managed GameLift service, you need to do some work to set up communication between your game and the GameLift service. The guides in this section provide detailed help for integrating your game with GameLift, whether you plan to deploy a custom game server or use GameLift ready-built Realtime Servers, and for adding a matchmaking service using FlexMatch.

Tip
Learn more about ways to explore Amazon GameLift features, including Realtime Servers, using sample games (p. 27).

Topics
- Integrating Games with Custom Game Servers (p. 43)
- Integrating Games with Amazon GameLift Realtime Servers (p. 70)
- Adding FlexMatch matchmaking (p. 78)
- Integrating Games with the Amazon GameLift Plug-in for Unity (p. 79)

Integrating Games with Custom Game Servers

Amazon GameLift provides a full tool set for preparing your multiplayer games and custom game servers to run on the managed GameLift service. The GameLift SDKs contain libraries needed to enable game clients and servers to communicate with the GameLift service. The SDKs can be downloaded directly; they are also included in the Amazon Lumberyard game engine. For more details on the SDKs and where to get them, see GameLift SDKs (p. 22).

The topics in this section contain detailed instruction on how to add the necessary GameLift functionality to your game client and game server before deploying on GameLift. For a complete roadmap to getting your game up and running on GameLift, see Get Started with Custom Servers (p. 28).

Topics
- Game Engines and Amazon GameLift (p. 43)
- Integrating your game server for Amazon GameLift (p. 51)
- Integrating your Game Client for Amazon GameLift (p. 57)
- GameLift and game client/server interactions (p. 62)
- Testing Your Integration (p. 65)

Game Engines and Amazon GameLift

You can use the managed GameLift service with most major game engines that support C++ or C# libraries, including Amazon Lumberyard, Unreal Engine, and Unity. Build the version you need for your game; see the README files with each version for build instructions and minimum requirements.
For more information on available GameLift SDKs, supported development platforms and operating systems, see GameLift SDKs (p. 22) for game servers.

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

- Get Started with Custom Servers (p. 28) – A six-step workflow for successfully integrating GameLift into your game and setting up hosting resources.
- Add GameLift to your game server (p. 51) – Detailed instructions on integrating GameLift into a game server.
- Add Amazon GameLift to Your Game Client (p. 58) – Detailed instructions on integrating into a game client or service, including creating game sessions and joining players to games.

**Amazon Lumberyard**

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

**Game servers**

Prepare your game servers for hosting on GameLift using the GameLift Server SDK for C++ (p. 224). See Add GameLift to your game server (p. 51) 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 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 GameLift into your Lumberyard game project, see Add GameLift to an Amazon Lumberyard game client (p. 45) and Add Amazon GameLift to Your Game Client (p. 58).

**Unreal Engine**

**Game servers**

Prepare your game servers for hosting on GameLift by adding the GameLift Server SDK for Unreal Engine (p. 253) to your project and implementing the required server functionality. For help setting up the Unreal Engine plugin and adding GameLift code, see Add Amazon GameLift to an Unreal Engine Game Server Project (p. 45).

**Game clients and services**

Enable your game clients and/or game services to interact with 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 GameLift into your Unreal Engine game project, see Add Amazon GameLift to Your Game Client (p. 58).

**Unity**

**Game servers**

Prepare your game servers for hosting on GameLift by adding the GameLift Server SDK for C# (p. 239) to your project and implementing the required server functionality. For help setting up with Unity and adding GameLift code, see Add Amazon GameLift to a Unity Game Server Project (p. 49).

**Game clients and services**
Enable your game clients and/or game services to interact with 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 GameLift into your Unity game project, see Add Amazon GameLift to Your Game Client (p. 58).

Other Engines

For a full list of the GameLift SDKs available for game servers and clients, see the section called “GameLift SDKs” (p. 22).

Add GameLift to an Amazon Lumberyard game client

All game clients must be configured to enable communication with the managed 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
Learn more about ways to explore Amazon GameLift features, including Realtime Servers, using sample games (p. 27).

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. 18) for a user with "player" access in your AWS account
   - `gamelift_aws_secret_key` = part of the IAM security credentials (p. 18) 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` = 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. 61)

2. Add the following command to launch the server browser:

   Follow this pattern when using a GameLift fleet ID (`gamelift_fleet_id`):
   ```
   Bin64\[your game executable] +gamelift_fleet_id [your fleet ID] +gamelift_aws_region us-west-2 +gamelift_aws_access_key [your AWS access key] +gamelift_aws_secret_key [your AWS secret key] +sv_port 64091 +map [map name]
   ```

   Follow this pattern when using a GameLift alias ID (`gamelift_alias_id`):
   ```
   Bin64\[your game executable] +gamelift_alias_id [your alias ID] +gamelift_aws_region us-west-2 +gamelift_aws_access_key [your AWS access key] +gamelift_aws_secret_key [your AWS secret key] +sv_port 64091 +map [map name]
   ```

Add Amazon GameLift to an Unreal Engine Game Server Project

This topic helps you set up and use the GameLift Server SDK plugin for Unreal Engine in your game server projects. If you're unsure whether the GameLift service supports the operating systems you're using, see For custom game servers (p. 22).
Set Up the Unreal Engine Server SDK Plugin

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

To set up the GameLift SDK plugin for Unreal Engine

1. **Download the GameLift Server SDK.** To verify that your game system requirements are supported, see GameLift SDKs (p. 22).

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>}). Check the README file in this directory for minimum requirements and additional information before building the SDK.

   To build the SDK libraries, go to the directory GameLift-cpp-ServerSDK-{<version>} and compile with the flag -DBUILD_FOR_UNREAL set to true. The following instructions show how to compile using cmake.

   For Linux users:
   ```
   mkdir out
   cd out
   cmake -DBUILD_FOR_UNREAL=1 ..
   make
   ```

   The following binary files are generated:
   - out/prefix/lib/libaws-cpp-sdk-gamelift-server.so

   For Windows users:
   ```
   mkdir out
   cd out
   cmake -G "Visual Studio 15 2017 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 GameLift plugin files.** Open the directory for the plugin version of UE4 that you are working with (for example, GameLift-SDK-Release-3.3.3\GameLift-Unreal-plugin-3.3.3\UB4.25\GameLiftServerSDK). Copy the binary files that you created in Step 2 into the ThirdParty directory of the Unreal plugin:

   For Linux use these paths:
   ```
   ...
   ```
   for Windows use these paths:
   ```
   ```

Version
46
4. **Import the GameLift plugin into a project.** There are 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 name as a dependency to your game's list of ModuleRules. The following example shows a sample list of module names with the GameLift plugin added to it.

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

### Add GameLift Code

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

- Add GameLift to your game server (p. 51)
- GameLift Server API reference for Unreal Engine (p. 253)

When adding 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 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 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 GameLift.

```csharp
//This is an example of a simple integration with GameLift server SDK that makes game server processes go active on Amazon GameLift
```
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 communication.
  gameLiftSdkModule->InitSDK();

  //Respond to new game session activation request. GameLift sends activation request
  //to the game server along with a game session object containing game properties
  //and other settings. Once the game server is ready to receive player connections,
  //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 before shutting down the instance
  //that is hosting this game server to give it time to gracefully shut down on its own.
  //In this example, we simply tell GameLift we are indeed going to shut down.
  params->OnTerminate.BindLambda([=](){gameLiftSdkModule->ProcessEnding();});

  //HealthCheck callback. GameLift invokes this callback about every 60 seconds. By
  //default,
  //GameLift API automatically responds 'true'. A game can optionally perform checks on
  //dependencies and such and report status based on this info. If no response is
  //received
  //within 60 seconds, health status is recorded as 'false'.
  //In this example, we're always healthy!
  params->OnHealthCheck.BindLambda([=]() {return true; });

  //Here, the game server tells GameLift what port it is listening on for incoming
  //connections. In this example, the port is hardcoded for simplicity. Since active game
  //sessions that are on the same instance must have unique ports, you may want to assign port
  //values
  //from a range, such as:
  //const int32 port = FURL::UrlConfig.DefaultPort;
  //params->port;
  //params->port = 7777;

  //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.
  TArray<FString> logfiles;
  logfiles.Add(TEXT("aLogFile.txt"));
  params->logParameters = logfiles;

  //Let's run this code only if GAMELIFT is enabled. Only with Server targets!
  #if WITH_GAMELIFT
Add Amazon GameLift to a Unity Game Server Project

This topic helps you set up the GameLift C# Server SDK in your Unity game server projects. If you're unsure whether the managed GameLift service supports the operating systems you're using, see For custom game servers (p. 22).

Set up the C# Server SDK for Unity

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

To set up the GameLift Server SDK for Unity

1. **Download the GameLift Server SDK.** To verify that your game system requirements are supported, see GameLift SDKs (p. 22). The Server SDK zip file includes the C# Server SDK, with source files so that you can build the SDK as needed for your project.

2. **Build the C# SDK libraries.** In an IDE, load the C# Server SDK solution file that you want to use. Use the IDE's functionality to restore NuGet files for the project. See the README.md file for the C# Server SDK for minimum requirements and additional build options. Build the solution to generate the C# SDK libraries.

3. **Check the Configuration settings.** In the Unity Editor, open your game project. Go to File, Build Settings, Player Settings. Under Other Settings, Configuration, check the following settings:
   - Scripting Runtime Version: Set to the .NET solution you're using.

4. **Add the GameLift libraries to your Unity project.** In the Unity Editor, import the libraries that were produced by the solution build into the Assets/Plugins directory of your project. See the README.md file for a complete list of the libraries for the SDK version that you're using.

Add GameLift Server Code

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

- Add GameLift to your game server (p. 51)
- GameLift Server API reference for C# (p. 239)

The following code example uses a MonoBehaviour to illustrate a simple game server initialization with 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()
    {
        // Call ProcessReady to tell GameLift this game server is ready to receive game sessions!
        gameLiftSdkModule->ProcessReady(*params);
    }
}
```
//Set the port that your game service is listening on for incoming player connections (hard-coded here for simplicity)
var listeningPort = 7777;

//InitSDK establishes a local connection with the Amazon GameLift agent to enable further communication.
var initSDKOutcome = GameLiftServerAPI.InitSDK();
if (initSDKOutcome.Success)
{
    ProcessParameters processParameters = new ProcessParameters(
        (gameSession) => {
            //Respond to new game session activation request. GameLift sends activation request
            //to the game server along with a game session object containing game properties
            //and other settings. Once the game server is ready to receive player connections,
            //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.
            //GameLift will default to 'false' if the game server doesn't respond in time.
            //In this case, we're always healthy!
            return true;
        });

    //Here, the game server tells GameLift what port it is listening on for incoming player connections. In this example, the port is hardcoded for simplicity.
    Active game that are on the same instance must have unique ports.
    listeningPort, 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.ProcessEnding() when the application quits.
    // This resets the local connection with GameLift's agent.
    GameLiftServerAPI.ProcessEnding();
}

Integrating your game server for Amazon GameLift

Your custom game server needs to interact with the GameLift service, and potentially other resources, once it is deployed and running on GameLift instances. This section provides guidance on how to integrate your game server software with GameLift.

Integrating your game server is Step 2 on the Get Started with Custom Servers (p. 28) roadmap. These topics assume that you've created an AWS account and have an existing game server project.

The topics in this section describe how to handle the following integration tasks:

- Establish communication between the GameLift service and your deployed and running game servers.
- Get a TLS certificate to establish a secure connection between game client and game server.
- Enable your game server software, when deployed on a GameLift instance, to interact with other AWS resources.
- Allow game server processes to get information about the fleet they are running on.

Topics

- Add GameLift to your game server (p. 51)
- Communicate with other AWS resources from your fleets (p. 55)
- Get fleet data for a GameLift instance (p. 56)
- GameLift Server SDK references:
  - GameLift Server API reference for C++ (p. 224)
  - GameLift Server API reference for C# (p. 239)
  - GameLift Server API reference for Unreal Engine (p. 253)

Add GameLift to your game server

Your custom game server needs to communicate with the GameLift service once it is deployed and running on GameLift instances. Each game server process must be able to respond to events when they are triggered by the GameLift service. It must also keep GameLift informed about the server process status and (optionally) player connections. For more detail on how your game server, your game client, and the GameLift service work together to manage game hosting, see GameLift and game client/server interactions (p. 62).

To prepare your game server to interact with GameLift, add the GameLift Server SDK to your game server project and build in basic functionality described in this topic. The Server SDK is available in...
several languages. See GameLift SDKs (p. 22) to learn more about the GameLift Server SDK. Download the latest version of Server SDK.

Server SDK API references:

- C++ Server API Reference (p. 224)
- C# Server API Reference (p. 239)
- Unreal Engine Plugin API Reference (p. 253)

Initialize the server process

This task is required.

Add code to establish communication with the GameLift service and report that the server process is ready to host a game session. This code must run before any GameLift-dependent code, such as on launch.

Note

The names of Server API operations are the same in all available languages.

1. Initialize GameLift API client by calling `InitSdk()`.
2. Notify GameLift that a server process is ready to host a game session. Call `ProcessReady()` (C# (p. 245) C++ (p. 229) Unreal (p. 257)) with the following information. `ProcessReady()` should be called only once per server process.
   
   - Port number being used by the server process. The port number and an IP address are provided to game clients so that they can connect to the server process to join a game session.
   - Location of files, such as game session logs, that you want GameLift to retain. These are files that are generated by the server process during a game session. They are stored temporarily on the instance where the server process is running, and are lost when the instance shuts down. Any files that are listed here are uploaded to the GameLift service. You can access them through the GameLift console or by calling the GameLift Service API GetGameSessionLogUrl(). Tip: If your fleet is configured to run multiple concurrent server processes per instance, consider using a file naming scheme that uniquely identifies game sessions.
   - Names of callback functions that can be invoked by the GameLift service on the server process. Your game server needs to implement these functions. Instructions for implementing these functions is covered in the remainder of this topic. See also ProcessParameters (p. 237).
     - `onHealthCheck` (optional) is called regularly to request a health status report from the server process.
     - `onStartGameSession` (required) is called when the GameLift service receives request to start a new game session (service API CreateGameSession()).
     - `onProcessTerminate` (required) is called when the GameLift service needs to force the server process to terminate, allowing the server process to shut down gracefully.
     - `onUpdateGameSession` (optional) is called when the 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 if you're using the FlexMatch backfill feature.

You can set up a game server to securely access AWS resources that you own or control. Learn more about how to Communicate with other AWS resources from your fleets (p. 55).

Report server process health

This task is optional.
Add code to implement the callback function `onHealthCheck()`. This function is invoked by the GameLift service to periodically collect health metrics from the server process. The server process's response to a health check is a binary: either healthy or unhealthy. To implement 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 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 choose to not implement a health check callback, the GameLift service considers the server process to be healthy unless the process does not respond, in which case it is considered unhealthy.

Server process health is used by 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 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 GameLift console.

**Retrieve a TLS certificate**

*This task is optional.*

If the server process is running on a fleet that has TLS certificate generation enabled, you can retrieve the TLS certificate and use it to establish a secure connection with a game client and encrypt client/server communication. A copy of the certificate is stored on the instance. To get the file location, call `GetInstanceCertificate()` (C# (p. 241)) (C++ (p. 225)) (Unreal (p. 254)).

**Start a game session**

*This task is required.*

Add code to implement the callback function `onStartGameSession`. The GameLift service invokes this method in order to trigger a new game session on the server process, as part of game session placement or matchmaking placement activities.

The `onStartGameSession` function takes a `GameSession` object as an input parameter. This object includes key game session information, such as maximum players, and may include game data and player data. The function implementation should accomplish the following tasks:

- Initiate actions to create a new game session based on the `GameSession` properties. At a minimum, the game server needs to associate the game session ID, which game clients will reference when connecting to the server process.
- Process game data and player data as needed. This data is in the `GameSession` object.
- Notify GameLift when the new game session is ready to accept players. Call the server API action `ActivateGameSession()` (C# (p. 241)) (C++ (p. 225)) (Unreal (p. 254)). In response to a successful call, the GameLift service changes the game session status to ACTIVE.

**Validate a new player**

*This task is optional.*

If you're tracking the status of player sessions, add code to validate new players when they connect to a game server. The purpose of validation is to enable GameLift to keep track of current players and
available game session slots. In essence, your game server is verifying that the player who is trying to connect is the same person who reserved an open slot.

For validation, a game client that is requesting access to the game session must include a player session ID. This ID is automatically generated by the GameLift service when a player asks to join a game (such as with StartGameSessionPlacement() or StartMatchmaking()). The player session reserves an open slot in a game session.

When game server process receives a game client connection request, it calls AcceptPlayerSession() (C# (p. 240)) (C++ (p. 224)) (Unreal (p. 254)) with the player session ID. In response, the GameLift service verifies that player session ID corresponds to an open slot reserved in the game session. After the player session ID is validated, 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 GameLift service, the server process should deny the connection.

Report a player session ending

This task is optional.

If you're tracking the status of player sessions, add code to notify the GameLift service when a player leaves the game session. This code should run whenever the server process detects a dropped connection. This notification enables the GameLift service to accurately track current players and available slots in the game session.

In your code to handle dropped connections, add a call to the server API action RemovePlayerSession() (C# (p. 246)) (C++ (p. 232)) (Unreal (p. 257)) with the corresponding player session ID.

End a game session

This task is required.

Add code to notify the GameLift service when a game session is ending. As a best practice, server processes are shut down once a game session is completed in order to recycle and refresh hosting resources. This code should be added to the server process shutdown sequence.

At the start of the process server shutdown code, call the server API action ProcessEnding() (C# (p. 244)) (C++ (p. 229)) (Unreal (p. 256)). This call notifies the GameLift service that the server process is shutting down. On receipt of this notification, the GameLift service changes the game session status and server process status to TERMINATED. After calling ProcessEnding(), it is safe for the process to shut down.

Respond to a server process shutdown notification

This task is required.

Add code to shut down the server process in response to notification from the GameLift service. GameLift sends this notification when the server process has been reported consistently unhealthy, or if the instance where the server process is running is being terminated. An instance might be terminated as part of a capacity scale-down event, or in response to Spot Instance interruption.

Make the following changes to your game server code to handle a shutdown notification:

- Implement the callback function onProcessTerminate(). This function should call the code that shuts down the server process. When GameLift invokes this method, the server process usually has a few minutes to gracefully disconnect players, preserve game state data, and perform other cleanup tasks. Spot Instance interruptions provide a two-minute notice.
- Call the server API action GetTerminationTime() (C# (p. 243)) (C++ (p. 228)) from your game server shut-down code. If GameLift has issued a call to terminate the server process, GetTerminationTime() returns the estimated termination time.
• At the start of your game server shut-down code, call the server API action `ProcessEnding()` (C# (p. 244)) (C++ (p. 229)) (Unreal (p. 256)). This call notifies the GameLift service that the server process is shutting down. On receipt of this notification, the GameLift service changes the server process status to TERMINATED. After calling `ProcessEnding()`, it is safe for the process to shut down.

Communicate with other AWS resources from your fleets

This topic describes how to set up your game server software to communicate directly and securely with other AWS resources while running on GameLift instances. For example, you might want to:

• Send instance log data to Amazon CloudWatch logs.
• Capture CloudWatch metrics to get better visibility into instance performance.
• Obtain sensitive information, such as passwords, that are stored remotely in an Amazon S3 account.
• Dynamically read and write game data that is stored in Amazon DynamoDB or other data storage service, such as game modes or inventory.
• Use Amazon SQS to send signals directly to an instance.
• Access custom resources that are deployed and running on Amazon EC2.

When you deploy your game build with GameLift, fleets and instances are allocated to your account but they are owned by the GameLift service. In order to have your hosted software interact with AWS resources that you own, you need to give limited access permissions to the GameLift service. You can use either of the following methods to establish this access:

• Create an AWS Identity and Access Management (IAM) role. This option is useful to access resources directly associated with AWS services, such as an Amazon S3 bucket, Amazon CloudWatch metrics, or AWS Lambda scripts. This is the simplest and recommended method.
• Set up Amazon Virtual Private Cloud (VPC) peering connections. This is an advanced feature and not a commonly used solution.

Access AWS resources using an IAM role

The following tasks must be completed to enable your game server or other applications to access your AWS resources while running on GameLift instances.

1. **Set up an IAM role for the GameLift service.** Follow the instructions at Set up a role for GameLift access (p. 21) to create an IAM role. A role lays out a set of permissions for limited access to your AWS resources and specifies the entity (in this case the GameLift service) that can assume the role. Once you've created the role, take note of the new role's Amazon Resource Name (ARN), which you'll need when creating a fleet.

2. **Associate the service role with a GameLift fleet.** Once the service role is created, you need to enable your game servers and other applications to assume the service role while running on GameLift instances. To do this, you must provide the service role ARN to a fleet. Applications that are running on any instance in the fleet can then assume the role and acquire the necessary credentials for access.

   A service role can only be specified during fleet creation. Once the fleet is created, you cannot add or change the role ARN. Only one ARN can be provided to a fleet.

   For help with providing a service role ARN to a fleet, see Deploy a GameLift fleet with a custom game build (p. 112).

3. **Add code to your application to assume the service role.** Any application that is running on a GameLift instance can assume an IAM role, if one is associated with the fleet. This includes game
servers and other executables, such as install scripts and daemons. An application can access resources during fleet creation and build installation, when starting or stopping service process and game sessions, or in response to game events.

In the application code, before accessing an AWS resource, the application must first assume the service role by calling the AWS Security Token Service API AssumeRole and specifying the same service role ARN that is associated with the fleet. This action returns a set of temporary credentials, which enable the application to access the AWS resource. Learn more about Using Temporary Security Credentials to Request Access to AWS Resources.

Access AWS resources with VPC peering

You can use Amazon Virtual Private Cloud (VPC) peering to establish fast and secure communication between an application running on a GameLift instance and another AWS resource. An Amazon VPC is a virtual network, defined by you, that includes a set of resources managed through your AWS account. Each GameLift fleet has its own VPC. With VPC peering, you can establish a direct network connection between the VPC for your fleet and the 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 GameLift’s VPC peering 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.

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 VPC peering for GameLift (p. 156).

See more information on Amazon’s virtual private clouds and VPC peering. You can peer your GameLift fleets with VPCs in any AWS account that you have access to.

Get fleet data for a GameLift instance

There are some situations where your custom game build or Realtime Servers script, once deployed and running on a GameLift instance, may need to access information about the fleet that the instance belongs to. For example, your game build or script might include code to:

- Monitor instance activity filtered based on fleet data.
- Roll up instance metrics to track activity by fleet data. Many games use this data for live-ops activities.
- Provide relevant data to custom game services, such as for matchmaking, additional capacity scaling, or A/B testing.

Fleet information is available as a JSON file on each instance in the following locations:

- Windows: C:\GameMetadata\gamelift-metadata.json
- Linux: /local/gamemetadata/gamelift-metadata.json

The gamelift-metadata.json file includes the following fleet data, which corresponds to the attributes of a GameLift Fleet resource.

- fleetArn – Amazon resource name (ARN) that identifies the fleet of the current instance. This identifier is unique across all AWS Regions.
- fleetId – Unique identifier that is assigned to the fleet of the current instance. This identifier is specific to an AWS Region.
**Integrating your Game Client for Amazon GameLift**

The topics in this section describe the managed GameLift functionality that you can add to a game client or game service. A game client or service may need to handle the following tasks:

- Request information about active game sessions from the GameLift service.
- Join a player to an existing game session.
- Create a new game session and join players to it.
- Change metadata about an existing game session.
Integrating a Game Client

Adding GameLift to your multiplayer game client is Step 5 in the Get Started with Custom Servers (p. 28). The following instructions assume that you’ve created an AWS account, generated a GameLift-enabled game server and uploaded it to GameLift, and used GameLift tools (such as the GameLift console) to create and configure a virtual fleet to host your game sessions. When adding 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 GameLift service and game servers running on GameLift, see GameLift and game client/server interactions (p. 62).

Topics

• Add Amazon GameLift to Your Game Client (p. 58)
• Create Game Sessions with Queues (p. 61)
• Generate Player IDs (p. 61)

Add Amazon GameLift to Your Game Client

Integrate 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 is usually placed in client services that handle tasks such as player authentication, matchmaking, or game session placement.

To integrate GameLift functionality into your game, use the AWS SDK, which includes APIs for GameLift. The AWS SDK is available in C++, C#, and several other languages. For details on the AWS SDK, version information, and language support, see For client services (p. 23). You can find general information about the GameLift APIs in the GameLift Service API Reference. For language-specific versions, see the AWS SDK for .NET API Reference or the AWS SDK for C++ API Reference.

Note
Interested in adding matchmaking to your game? See the GameLift FlexMatch Developer Guide for detailed information on how to set up matchmaking for your GameLift hosted game.

Set Up GameLift on a Client or Service

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

Note
To set up your game client for testing with GameLift Local, see Testing Your Integration (p. 65).

1. Decide whether to use the default client configuration or create custom settings. For custom settings, you must create a custom client configuration object. See AWS::Client::ClientConfiguration (C++) or AmazonGameLiftConfig (C#).

   A client configuration specifies a target region and endpoint. The region determines which resources (fleets, queues, matchmakers, etc.) 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 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 a GameLift client. Call Aws::GameLift::GameLiftClient() (C++) or AmazonGameLiftClient() (C#) with either a default client configuration or a custom configuration.

3. Add a mechanism to generate a unique identifier for each player. GameLift requires a unique player ID to connect to a game session. For more details, see Generate Player IDs (p. 61).
4. Collect and store the following information to use when contacting GameLift:

- **Target fleet** – Most 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 GameLift service must provide credentials for the AWS account that hosts the game. This is the account you used to set up your 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. 12) 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 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. 139).

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, 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 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 the GameLift resource protection feature to limit the number of game sessions one player can create, you'll need to specify the game session 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. 12) 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, GameLift first verifies that the game session is accepting new players and has an available player slot. If successful, GameLift reserves a slot for the player, creates the new player session, and returns a PlayerSession object containing the DNS name, 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. 61) 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 GameLift service to the game server when the player connects directly to the game session. Player data is not used by 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 connection information. Use this information to establish a direct connection to the server process.

- You can connect using the specified port and either the DNS name or IP address assigned to the server process.
- If your fleets have TLS certificate generation enabled, you must connect using the DNS name and port. This is required even if you haven't implemented a server authentication process.
- If your game server is set up to validate incoming player connections, you must reference the player session ID.

Once connected, the game client and server process communicate directly without involving the GameLift service. The server process maintains communication with the GameLift service to report player connection status, health status, etc. If the game server validates incoming players, it verifies that the player session ID matches a reserved slot in the game session, and either accepts or denies the player connection. When the player disconnects, the server process reports the dropped connection.

If you want to enable server authentication and encrypt data packets travelling between game client and the game session, you need to build this functionality. When the TLS certificate generation feature is enabled for a new fleet, GameLift only gets the TLS certificate and creates DNS entries for each instance in the fleet.

### 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 (`StartGameSessionPlacement`), 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

The managed GameLift service 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.
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 GameLift. For more information on player sessions, see View Data on Game and Player Sessions (p. 167).

## GameLift and game client/server interactions

This topic describes the interactions between a client app service, a game server, and the GameLift service. See also the Amazon GameLift–Game Server/Client Interactions (p. 64) diagram.

### Setting up a new server process

1. The 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.

   The server process then waits for a callback from the GameLift service.
3. The GameLift service updates the status of the server process to ACTIVE to allow placement of game sessions with the server process. (If the server process is the first one to become active on an instance, the instance status is also updated to ACTIVE.)
4. The GameLift service begins calling the `onHealthCheck` callback and continues to call it periodically while the server process is active. The server process can report either healthy or not healthy within one minute.

### Creating a game session

1. The Client app calls the client API action `StartGameSessionPlacement()`.
2. The GameLift service creates a new `GameSessionPlacement` ticket with status PENDING and returns it to the requesting client app.
3. The Client app polls the placement ticket status by periodically calling `DescribeGameSessionPlacement()`.
4. The GameLift service initiates game session placement, selecting an appropriate fleet and searching for an active server process in the fleet with 0 game sessions. When a server process is located, GameLift does the following:
   - Creates a `GameSession` object with the game session settings and player data from the placement request and status ACTIVATING.
   - Invokes the `onStartGameSession` callback on the server process. It passes the `GameSession` object with information that the server process may need to set up the game session.
   - Changes the server process's number of game sessions to 1.
5. The **server process** runs the `onStartGameSession` callback function. When ready to accept player connections, the server process calls `ActivateGameSession()` and waits for player connections.

6. The **GameLift service** does the following:
   - Updates the `GameSession` object with connection information for the server process (including the port setting that was reported with `ProcessReady()`) and changes the status to ACTIVE.
   - Updates the `GameSessionPlacement` ticket with the connection information and sets the ticket status to FULFILLED.

7. The **Client app** detects the updated ticket status and is able to use the connection information to connect to the server process and join the game session.

### Adding a player to a game session

This sequence describes the process of adding a player to an existing game session. Player sessions can also be requested as part of a game session placement request.

1. The **Client app** calls the client API action `CreatePlayerSession()` with a game session ID.

2. The **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 GameLift service passes the `PlayerSession` object to the server process. The server process either accepts or rejects the connection.

5. The **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 **GameLift service** changes the `PlayerSession` status to COMPLETED and reopens the player slot in the game session.

### Shutting down a game session

This sequence is used when a server process is ending the current game session terminating itself.

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

2. The **GameLift service** does the following:
   - Uploads game session logs to Amazon Simple Storage Service (Amazon S3).
   - Changes the `GameSession` status to TERMINATED.
   - Changes the server process status to TERMINATED.
• Recycles instance resources based on the fleet's runtime configuration.

**Responding to a shutdown request**

This sequence is used by the GameLift service to force a server process to shut down. This action may be done to end an unhealthy process or to gracefully shut down a process when the instance the process is on is being terminated, such as during autoscaling. It might also be used when handling a Spot Instance interruption.

1. The **GameLift service** invokes the server process's `onProcessTerminate` callback.
2. The **server process** runs the `onProcessTerminate` callback function, which triggers the process's termination sequence, ending with a call to `ProcessEnding()`.
3. The **GameLift service** does the following, either in response to receiving the `ProcessEnding()` call or after five minutes:
   - If a game session was in progress, uploads game session logs (if any) to Amazon S3 and changes the `GameSession` status to TERMINATED.
   - Changes the server process status to TERMINATED.
   - Recycles instance resources based on the fleet's runtime configuration.

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

The diagram below outlines API interactions that occur when managing multiplayer game session tasks using the managed GameLift solution.
Testing Your Integration

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

With GameLift Local, you can verify the following:

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

GameLift local supports the following APIs:

- CreateGameSession
- CreatePlayerSession
- CreatePlayerSessions
- DescribeGameSessions
- DescribePlayerSessions
- SearchGameSessions

**Set Up GameLift Local**

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 GameLift-supported language.

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

- A build of the 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 GameLift Local service. This verifies that your game server is performing as expected with the following:

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

1. **Start 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:

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

   Once Local starts, you see logs indicating that two local servers were started, one listening for your game server and one listening for your game client or the AWS CLI. Logs continue to report activity on the two local servers, including communication to and from your game components.

2. **Start your game server.**

   Start your 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 GameLift Local service. This means that your game server successfully initialized the GameLift Server SDK (with `InitSDK()`). It has called `ProcessReady()` with the log paths shown and, if successful, is ready to host a game session. While the game server is running, 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!
16:50:53,543  INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - onProcessReady data:
    logPathsToUpload: "C:\game\logs"
    logPathsToUpload: "C:\game\error"
    port: 1935
16:50:53,544  INFO || - [HostProcessManager] nioEventLoopGroup-3-1 - Registered new process true, true,
16:50:53,558  INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - onReportHealth received from /127.0.0.1:64247 with health status: healthy
```

Potential error and warning messages include the following:

- Error: “ProcessReady did not find a process with pID: `<process ID>`! Was `InitSDK()` invoked?”
- Warning: “Process state already exists for process with pID: `<process ID>`! Is `ProcessReady(...)` invoked more than once?”

3. **Start the AWS CLI.**

   Once your game server successfully calls `ProcessReady()`, you can start making client calls. Open another command prompt window and start the AWS CLI tool. Get and install the AWS Command Line Interface tool. The AWS CLI by default uses the GameLift web service endpoint. You must override this with the Local endpoint in every request using the `--endpoint-url` parameter, as shown in the following example request.

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

   In the AWS CLI command prompt window, `AWS gamelift` commands result in responses as documented in the [AWS CLI Command Reference](https://docs.aws.amazon.com/cli/latest/reference/gamelift/).

4. **Create a game session.**

   With the AWS CLI, submit a `CreateGameSession()` request. The request should follow the expected syntax. For Local, the `FleetId` parameter can be set to any valid string (`^fleet-\S+$`).

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

   In the Local command prompt window, log messages indicate that GameLift Local has sent your game server an `onStartGameSession` callback. If a game session is successfully created, your game server responds by invoking `ActivateGameSession`.  

---

**Version**

67
Testing Your Integration

In the AWS CLI window, GameLift responds with a game session object including a game session ID. Notice that the new game session's status is Activating. The status changes to Active once your game server invokes ActivateGameSession. If you want to see the changed status, use the AWS CLI to call DescribeGameSessions().

```json
{
    "GameSession": {
        "Status": "ACTIVATING",
        "MaximumPlayerSessionCount": 2,
        "FleetId": "fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d",
        "GameSessionId": "arn:aws:gamelift:local::gamesession/fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d/gsess-abcdef12-3456-7890-abcd-ef1234567890",
        "IpAddress": "127.0.0.1",
        "Port": 1935
    }
}
```

Test a Game Server and Client

To check your full game integration, including connecting players to games, you can run both your game server and client locally. This allows you to test programmatic calls from your game client to the GameLift Local. You can verify the following actions:

- The game client is successfully making AWS SDK requests to the 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 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
   ./gamelif-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 GameLift-integrated game server locally. See Test a Game Server (p. 66) for more detail on message logs.

3. **Configure your game client for Local and start it.**

To use your game client with the GameLift Local service, you must make the following changes to your game client's setup, as described in Set Up GameLift on a Client or Service (p. 58):

- 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 GameLift service.

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

For GameLift to manage player demand and correctly report metrics, your game server must report various statuses back to 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** – 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** – GameLift Local log messages should show that your game server calls TerminateGameSession().

  **Note**
  Previous guidance was to call TerminateGameSession() when ending a game session. This method is deprecated with GameLift Server SDK v4.0.1. See End a game session (p. 54).

- **Server process is terminated** – GameLift Local log messages should show that your game server calls ProcessEnding(). An AWS CLI call to DescribeGameSessions() should reflect a status change from Active to Terminated (or Terminating).

### Variations with Local

When using GameLift Local, keep in mind the following:

- Unlike the 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

### Integrating Games with Amazon GameLift Realtime Servers

Amazon GameLift Realtime Servers offers a lightweight server solution that is designed for games that don't need a complex, custom-built game server. Games such as mobile games, turn-based games, message-based games, etc., can use ready-to-go Realtime game servers that require minimal configuration but can also be customized with game-specific logic.

**Tip**

Learn more about ways to explore Amazon GameLift features, including Realtime Servers, using sample games (p. 27). The topics in this section describe how to enable your multiplayer game clients to connect to and use Realtime Servers game servers that are running on GameLift. For a complete roadmap to getting your game up and running with Realtime Servers, see Get Started with Realtime Servers (p. 41).

**Learn more about Realtime Servers:**

- How Realtime Servers Work (p. 8)

**Get started integrating your game with Realtime Servers:**

- Integrating a Game Client for Realtime Servers (p. 71)
- Creating a Realtime Script (p. 74)

- Get Started with Realtime Servers (p. 41)
Integrating a Game Client for Realtime Servers

This topic describes how to prepare your game client to be able to join and participate in Amazon GameLift-hosted game sessions.

There are two sets of tasks needed to prepare your game client:

- Set up your game client to acquire information about existing games, request matchmaking, start new game sessions, and reserve game session slots for a player.
- Enable your game client to join a game session hosted on a Realtime server and exchange messages.

Find or Create Game Sessions and Player Sessions

Set up your game client to find or start game sessions, request FlexMatch matchmaking, and reserve space for players in a game by creating player sessions. As a best practice, create a client service and use it to make the direct requests to the Amazon GameLift service when triggered by a game client action. The client service then relays relevant responses back to the game client.

1. Add the AWS SDK to your client service project, initialize an Amazon GameLift client, and configure it to use the hosting resources in your fleets and/or queues. The AWS SDK is available in several languages; see the Amazon GameLift SDKs For client services (p. 23).

2. Add GameLift functionality to your client service. For more detailed instructions, see Add Amazon GameLift to Your Game Client (p. 58) and Adding FlexMatch matchmaking. The best practice is to use game session placements to create new game sessions. This method lets you take full advantage of GameLift's ability to quickly and intelligently place new game sessions, as well as use player latency data to minimize game lag. At a minimum, your client service must be able to request new game sessions and handle game session data in response. You may also want to add functionality to search for and get information on existing game sessions, and request player sessions, which effectively reserve a player slot in an existing game session.

3. Convey connection information back to the game client. The backend game service receives game session and player session objects in response to requests to the Amazon GameLift service. These objects contain information, in particular connection details (IP address and port) and player session ID, that the game client needs to connect to the game session running on a Realtime Server.

Connect to Games on Realtime Servers

Enable your game client to connect directly with a hosted game session on a Realtime server and exchange messages with the server and with other players.

1. Get the Realtime Client SDK, build it, and add it to your game client project. See the README file for more information on SDK requirements and instructions on how to build the client libraries.

2. Call Client() (p. 209) with a client configuration that specifies the type of client/server connection to use.

   **Note**
   If you’re connecting to a Realtime server that is running on a secured fleet with a TLS certificate, you must specify a secured connection type.

3. Add the following functionality to your game client. See the Realtime Servers Client API (C#) Reference (p. 208) for more information.
   - Connect to and disconnect from a game
     - Connect() (p. 209)
     - Disconnect() (p. 210)
   - Send messages to target recipients
• SendMessage() (p. 211)
• Receive and process messages
  • OnDataReceived() (p. 214)
• Join groups and leave player groups
  • JoinGroup() (p. 211)
  • RequestGroupMembership() (p. 212)
  • LeaveGroup() (p. 211)

4. Set up event handlers for the client callbacks as needed. See Realtime Servers Client API (C#) Reference: Asynchronous Callbacks (p. 212).

When working with Realtime fleets that have TLS certificate generation enabled, the server is automatically authenticated using the TLS certificate. TCP and UDP traffic is encrypted in flight to provide transport layer security. TCP traffic is encrypted using TLS 1.2, and UDP traffic is encrypted using DTLS 1.2.

Game Client Examples

Basic Realtime Client (C#)

This example illustrates a basic game client integration with the Realtime Client SDK (C#). As shown, the example initializes a Realtime client object, sets up event handlers and implements the client-side callbacks, connects to a Realtime server, sends a message, and disconnects.

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

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

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

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

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

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

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

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

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

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

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

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

Creating a Realtime Script

To use Realtime Servers for your game, you need to provide a script (in the form of some JavaScript code) to configure and optionally customize a fleet of Realtime Servers. This topic covers the key steps in creating a Realtime script. Once the script is ready, upload it to the Amazon GameLift service and use it to create a fleet (see Upload a Realtime Servers script to GameLift (p. 103)).

To prepare a script for use with Realtime Servers, add the following functionality to your Realtime script.

Manage Game Session Life-Cycle (required)

At a minimum, a Realtime script must include the `Init()` function, which prepares the Realtime server to start a game session. It is also highly recommended that you also provide a way to terminate game sessions, to ensure that new game sessions can continue to be started on your fleet.
The `Init()` callback function, when called, is passed a Realtime session object, which contains an interface for the Realtime server. See Realtime Servers Interface (p. 220) for more details on this interface.

To gracefully end a game session, the script must also call the Realtime server's `session.processEnding` function. This requires some mechanism to determine when to end a session. The script example code illustrates a simple mechanism that checks for player connections and triggers game session termination when no players have been connected to the session for a specified length of time.

Realtime Servers with the most basic configuration—server process initialization and termination—essentially act as stateless relay servers. The Realtime server relays messages and game data between game clients that are connected to the game, but takes no independent action to process data or perform logic. You can optionally add game logic, triggered by game events or other mechanisms, as needed for your game.

### Add Server-Side Game Logic (optional)

You can optionally add game logic to your Realtime script. For example, you might do any or all of the following. The script example code provides illustration. See Amazon GameLift Realtime Servers Script Reference (p. 218).

- **Add event-driven logic.** Implement the callback functions to respond to client-server events. See Script Callbacks for Realtime Servers (p. 218) for a complete list of callbacks.

- **Trigger logic by sending messages to the server.** Create a set of special operation codes for messages sent from game clients to the server, and add functions to handle receipt. Use the callback `onMessage`, and parse the message content using the `gameMessage` interface (see `gameMessage.opcode` (p. 223)).

- **Enable game logic to access your other AWS resources.** For details, see Communicate with other AWS resources from your fleets (p. 55).

- **Allow game logic to access fleet information for the instance it is running on.** For details, see Get fleet data for a GameLift instance (p. 56).

### Realtime Servers Script Example

This example illustrates a basic script needed to deploy Realtime Servers plus some custom logic. It contains the required `Init()` function, and uses a timer mechanism to trigger game session termination based on length of time with no player connections. It also includes some hooks for custom logic, including some callback implementations.

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

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

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

// Defines how long to wait in Seconds before beginning early termination check in the example tick loop
const minimumElapsedTime = 120;
```
var session;                        // The Realtime server session object
var logger;                         // Log at appropriate level
    via .info(), .warn(), .error(), .debug()
var startTime;                      // Records the time the process started
var activePlayers = 0;              // Records the number of connected players
var onProcessStartedCalled = false; // Record if onProcessStarted has been called

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

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

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

// On Process Started is called when the process has begun and we need to perform any
// bootstrapping. This is where the developer should insert any code to prepare
// the process to be able to host a game session, for example load some settings or set
// state

// Return true if the process has been appropriately prepared and it is okay to invoke the
// GameLift ProcessReady() call.
function onProcessStarted(args) {
    onProcessStartedCalled = true;
    logger.info("Starting process with args: ", args);
    logger.info("Ready to host games...");
    return true;
}

// Called when a new game session is started on the process
function onStartGameSession(gameSession) {
    // Complete any game session set-up
    // Set up an example tick loop to perform server initiated actions
    startTime = getTimeInS();
    tickLoop();
}

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

// Return true if the process is healthy
function onHealthCheck() {
    return true;
}

// On Player Connect is called when a player has passed initial validation
// Return true if player should connect, false to reject
function onPlayerConnect(connectMsg) {
    // Perform any validation needed for connectMsg.payload, connectMsg.peerId
    return true;
// Called when a Player is accepted into the game
function onPlayerAccepted(player) {
    // This player was accepted -- let’s send them a message
    const msg = session.newTextGameMessage(OP_CODE_PLAYER_ACCEPTED, player.peerId,
        "Peer " + player.peerId + " accepted");
    session.sendReliableMessage(msg, player.peerId);
    activePlayers++;
}

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

    session.getPlayers().forEach((player, playerId) => {
        if (playerId != peerId) {
            session.sendReliableMessage(outMessage, peerId);
        }
    });
    activePlayers--;
}

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

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

// Return true if the send to group should be allowed
function onSendToGroup(gameMessage) {
    return true;
}

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

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

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

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

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

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

---

Adding FlexMatch matchmaking

Use GameLift FlexMatch to add player matchmaking functionality to your GameLift hosted games. You can use FlexMatch with either custom game servers or Realtime Servers.

FlexMatch pairs the matchmaking service with a customizable rules engine. You design how to match players together based on player attributes and game modes that make sense for your game. FlexMatch manages the nuts and bolts of evaluating players who are looking for a game, forming matches with one or more teams, and starting game sessions to host the matches.

To use the full FlexMatch service, you must have your hosting resources set up with queues. GameLift uses queues to locate the best possible hosting locations for games across multiple regions and computing types. In particular, GameLift queues can use latency data, when provided by game clients, to place game sessions so that players experience the lowest possible latency when playing.

For more information on FlexMatch including detailed help with integrating matchmaking into your games, see these GameLift FlexMatch Developer Guide topics:

- How GameLift FlexMatch works
- FlexMatch integration steps
Integrating Games with the Amazon GameLift Plug-in for Unity

Amazon GameLift provides tools for preparing your multiplayer games and custom game servers to run on the GameLift service. The GameLift SDKs contain libraries needed to enable game clients and servers to communicate with the GameLift service. The Amazon GameLift Plug-in for Unity makes it easier to access GameLift resources and integrate GameLift into your Unity game. You can use the Plug-in for Unity to access GameLift APIs and deploy AWS CloudFormation templates for common gaming scenarios.

The topics in this section contain more information about how to obtain and configure the Plug-in for Unity.

Topics
- Installing the Plug-in for Unity (p. 79)
- Configuring .NET Settings (p. 81)
- Testing Your Game Locally (p. 82)
- Deploying a Scenario (p. 83)
- Importing and Running a Sample Game (p. 86)

Installing the Plug-in for Unity

In this section, you download and install the Amazon GameLift Plug-in for Unity. The Plug-in for Unity requires Unity for Windows version 2019.4 LTS and 2020.3 LTS. To build and test games using GameLift Local, the GameLift Managed Servers SDK is required. You will also need to install the current version of Java and .NET 4.x.

To download and install the Plug-in for Unity

1. Download the Amazon GameLift Plug-in for Unity. You can find the latest version on the Amazon GameLift Plug-in for Unity Repository page. Under the latest release, select Assets, and then download com.amazonaws.gamelift-version.tgz.
2. Launch Unity and select a project.
3. On the menu, select Window, and then choose Package Manager:
4. In the **Package Manager** tab, under the tab, select +, and then choose **Add package from tarball...**:

5. In the **Select packages on disk** window, navigate to the `com.amazonaws.gamelift` folder, select the file `com.amazonaws.gamelift-version.tgz`, and then choose **Open**:
6. Once the Plug-in for Unity is loaded, GameLift will be added as a new item on the Unity menu. It may take a few minutes to install and recompile scripts. The GameLift Plugin Settings tab will automatically open:

7. Close the window. To configure .NET settings, proceed to Configuring .NET Settings (p. 81).

**Configuring .NET Settings**

This section describes how to configure your project to use .NET 4.x. The Amazon GameLift Plug-in for Unity requires .NET 4.x.
To configure .NET settings

1. In Unity, in the Plug-in for Unity tab, select the SDK tab.

2. In the SDK pane, choose Use .NET 4.x. This will override the current .NET settings for the project. GameLift Local requires .NET 4.x. You must select .NET 4.x to use the GameLift Local SDK and to test locally unless you have already built your game executable.

When configured, the status will change from Not Configured to Configured.

Testing Your Game Locally

You can test builds of your game server on your local machine. Once you launch your game server, you can launch your game client and debug GameLift methods and other interactions.

Topics

• Configuring Local Testing (p. 82)

• Testing Your Game Locally (p. 82)

Configuring Local Testing

Amazon GameLift Local is a client-side debugging tool that emulates a subset of the GameLift API on your local development machine. You can use GameLift Local to verify code changes in seconds, without a network connection.

To configure GameLift Local

1. In Unity, in the Plug-in for Unity tab, select the Test tab.

2. In the Test pane, select Download GameLift Local. The Plug-in for Unity will open a browser window and download GameLift_06_03_2021.zip to your downloads folder. Your browser window may automatically close.

   The Server SDK zip file includes the C# Server SDK, .NET source files and a pre-built .NET component suitable for Unity.

3. Unzip the downloaded file GameLift_06_03_2021.zip.

4. In the GameLift Plugin Settings window, select GameLift Local Path window, navigate to the unzipped folder, select the file GameLiftLocal.jar, and then choose Open.

   When configured, local testing status will change from Not Configured to Configured.

5. Verify the status of the JRE. If it is Not Configured select Download JRE and install the recommended Java version.

   Once the Java environment has been installed and configured, the status will change to Configured.

6. The Plug-in for Unity can launch your game server. Select Open Local Test UI and then specify the Game Server .exe File Path, and then choose Deploy and Run. You can stop your game server by choosing Stop or by closing the game server windows.

Testing Your Game Locally

The Amazon GameLift Plug-in for Unity makes it easy to launch your game server for local testing. You can launch as many game clients as you need to test your game.
To test your game locally

1. In Unity, in the Plug-in for Unity tab, select the Test tab.
2. In the Test pane, select Open Local Test UI.
3. In the Local Testing window, specify a Server executable path. Select ... to select the path and executable name of your server application.
4. In the Local Testing window, specify a GL Local port.
5. Select Deploy & Run to deploy and run the server.
6. When local testing is completed, select Stop.

You can also manually close the game server windows.

Deploying a Scenario

The Amazon GameLift Plug-in for Unity includes the following pre-built sample scenarios you can customize for your game:

- **Auth Only** — This scenario creates a game backend service that performs only player authentication and no game server capability. It creates a Amazon Cognito user pool to store player authentication information and an Amazon API Gateway REST endpoint-backed AWS Lambda handlers to start a game and view game connection information. The Lambda handler always returns a 501 Error (Unimplemented)

- **Single-Region Fleet** — This scenario creates a game backend service with a single GameLift fleet. After the player authenticates and starts a game (with a POST request to /start_game), an AWS Lambda handler searches for an existing viable game session with an open player slot on the fleet via gamelift::SearchGameSession. If an open slot is not found, the Lambda creates a new game session via gamelift::CreateGameSession. Once a game start is requested, the game client should poll the backend with POST requests to /get_game_connection to receive a viable game session.

- **Multi-Region Fleets with Queue and Custom Matchmaker** — In this scenario, Amazon GameLift queues are used in conjunction with a custom matchmaker. The custom matchmaker forms matches by grouping up the oldest players in the waiting pool. Once the placement is done, GameLift publishes a message to the Amazon Simple Notification Service topic in the backend service, triggering a Lambda function to store placement details along with game conection details to a Amazon DynamoDB table. Subsequent GetGameConnection calls read from this table and return the connection information to the game client.

- **SPOT Fleets with Queue and Custom Matchmaker** — This scenario is the same as Multi-Region Fleets with Queue and Custom Matchmaker except it configures three fleets. Two of the fleets are Spot fleets containing nuanced instance types to provide durability for Spot unavailabilities. The third fleet is an On-Demand fleet to serve as a backup in case the other Spot fleets go unavailable. Using a GameLift queue can keep availability high and cost low. For more information and best practices about queues, see Setting up GameLift queues for game session placement (p. 137).

- **FlexMatch** — On StartGame requests, a Lambda creates a matchmaking ticket via gamelift::StartMatchmaking. A separate Lambda listens to FlexMatch Match events similar to the queue example above. This deployment scenario also uses a low frequency poller to describe incomplete tickets via gamelift::DescribeMatchmaking. The incomplete tickets are periodically described so they are not discarded by GameLift. This is a best practice recommended by Track Matchmaking Events. For more information about FlexMatch, see What is GameLift FlexMatch.

Each sample scenario uses an AWS CloudFormation template to create a stack with all of the resources needed for the sample game. You can remove the resources by deleting the corresponding AWS CloudFormation stack.
Updating AWS Credentials

The Amazon GameLift Plug-in for Unity requires security credentials to deploy a scenario. You can choose to create new credentials or using existing credentials.

For more information about configuring credentials, see Understanding and getting your AWS credentials.

To update AWS credentials

1. In Unity, in the Plug-in for Unity tab, select the Deploy tab.
2. In the Deploy pane, select AWS Credentials.
3. You can create new credentials or choose existing credentials.
   - To create credentials, select Create new credentials profile and then specify the New Profile Name, AWS Access Key ID, AWS Secret Key, and AWS Region.
   - To choose existing credentials, select Choose existing credentials profile and then select a profile name and AWS Region.
4. In the Update AWS Credentials window, select Update Credentials Profile.

Updating Account Bootstrap

The bootstrap location is an Amazon S3 bucket used during deployment. It is used to store game server assets and other dependencies. The AWS Region you select for the bucket must be the same Region you will use for the sample scenario deployment.

For more information about Amazon S3 buckets, see Creating, configuring, and working with Amazon Simple Storage Service buckets.

To update the account bootstrap location

1. In Unity, in the Plug-in for Unity tab, select the Deploy tab.
2. In the Deploy pane, select Update Account Bootstrap.
3. In the Account Bootstrapping window, you can choose an existing Amazon S3 bucket or create a new Amazon S3 bucket:
   - Select Choose existing Amazon S3 bucket to choose an existing bucket. Choose Update to save your selection.
   - Select Create new Amazon S3 bucket to create a new Amazon Simple Storage Service bucket, then select a Policy. The policy specifies when the Amazon S3 bucket will be expire. Choose Create to create the bucket.

Deploying a Sample Game Scenario

You can use a sample scenario to test your game in the cloud. Each scenario uses a AWS CloudFormation template to create a stack with all of the required resources. Many of the scenarios require a game server
executable and build path. When the scenario is deployed, some game assets are copied to the bootstrap location as part of deployment.

You must configure AWS credentials and an AWS account bootstrap to deploy a scenario.

To deploy a scenario

1. In Unity, in the Plug-in for Unity tab, select the Deploy tab.
2. In the Deploy pane, select Open Deployment UI.
3. In the Deployment window, select a scenario. The Auth Only scenario does not require a server executable and can deploy quickly. All other scenarios require a server path and server executable and can take about 30 minutes to deploy.
4. Specify a Game Name. It must be unique. It will be used as part of the AWS CloudFormation stack name when the scenario is deployed. For example, if you specify MySampleGame, the corresponding AWS CloudFormation stack will be named "GameLiftPluginForUnity-MySampleGame".
5. Select the Game Server Build Folder Path. The build folder path points to the folder containing the server executable and dependencies. For example, c:/SampleGame/GameServer

You will not be able to select a build folder path if it is not required by the chosen scenario.
6. Select the Game Server Build .exe File Path. The build executable file path points to the game server executable. For example, c:/SampleGame/GameServer/SampleGame.exe

You will not be able to select a build executable file path if it is not required by the chosen scenario.
7. Select Start Deployment to initiate deployment of the scenario. You can follow the status of the update in the Deployment window under Current State:

```
Current State
Deployment Status: CREATE_IN_PROGRESS
Scenario: Single-Region Fleet
Game Name: MySampleGame
Region: us-west-2
Last Updated: 2021-09-08 10:59 AM
```

8. When the scenario completes deployment, Current State will be updated to include the Cognito Client ID and API Gateway Endpoint you can copy and paste into the sample game.

```
Deployment Status
Deployment Status: CREATE_COMPLETE
Scenario: Single-Region Fleet
Game Name: TestGame
AWS region: us-west-2
Last Updated: 2021-09-13 02:06 PM
Cognito Client ID 484jb5l7oh24vvkd3rof2kqj1nt
API Gateway Endpoint https://8igpepx7g1.execute-api.us-west-2.amazonaws.com/v1/
View AWS CloudFormation Console
```

9. To update sample game settings, on the Unity menu, choose Go To Client Connection Settings. This will display an Inspector tab on the right side of the Unity screen.
Make sure Local Testing Mode is not selected.

Use the API Gateway endpoint value to specify API Gateway Endpoint and the Amazon Cognito client ID to specify the Cognito Client ID. Select the same AWS Region you used for the scenario deployment. You can then rebuild and run the sample game client using the deployed scenario resources.

Deleting Resources Created by the Scenario

To delete the resources created for the scenario, you must delete the corresponding AWS CloudFormation stack.

To delete resources created by the scenario

1. In the Amazon GameLift Plug-in for Unity Deployment window, select View AWS CloudFormation Console to open the AWS CloudFormation console. You can also open the AWS CloudFormation console directly with the URL https://console.aws.amazon.com/cloudformation.
2. In the AWS CloudFormation console, select Stacks, and then choose the stack that includes the game name specified during deployment. For example, if your game name is MySampleGame, the AWS CloudFormation stack name will be GameLiftUnityPlugin-MySampleGame.
3. Select Delete to delete the stack. It may take a few minutes to delete a stack. When the stack used by the sample scenario is deleted, its status will be ROLLBACK_COMPLETE.

Importing and Running a Sample Game

The Amazon GameLift Plug-in for Unity includes a sample game you can use to explore the basics of integrating your game with Amazon GameLift. In this section, you build the game server and client and then test locally using GameLift Local. You will see GameLift messages from the game server and game client in the log window.

To build and run the sample game client

1. In Unity, on the menu, select GameLift, and then choose Import Sample Game.
2. In the Import Sample Game window, choose Import to import the game and all of its assets and dependencies.
3. Build the game server. In Unity, on the menu, select GameLift, and then choose Apply Sample Server Build Settings. After the game server settings are configured, Unity will recompile assets.
4. In Unity, on the menu, select File, and then choose Build Settings..., confirm Server Build is checked, choose Build, and then select a build folder.
   Unity will build the sample game server, placing the executable and required assets in the specified build folder.
5. Close the build window.
6. In Unity, on the menu, select GameLift, and then choose Apply Sample Client Build Settings. After the game client settings are configured, Unity will recompile assets.
7. In Unity, on the menu, select Go To Client Settings. This will display an Inspector tab on the right side of the Unity screen. In the GameLift Client Settings tab, choose Local Testing Mode.
8. Build the game client. In Unity, on the menu, select File, and then choose Build Settings..., confirm Server Build is not checked, choose Build, and then select a build folder.
   Unity will build the sample game client, placing the executable and required assets in the specified build folder.
9. Close the build window.

The game server and client are built. In the next few steps, you run the game and see how it interacts with GameLift.

10. In Unity, in the Plug-in for Unity tab, select the Deploy tab.

11. In the Test pane, select Open Local Test UI.

12. In the Local Testing window, specify a Game Server .exe File Path. The path must include the executable name. For example, C:/MyGame/GameServer/MyGameServer.exe.

13. Select Deploy and Run. The Plug-in for Unity will launch the game server and open a GameLift Local log window. The windows will contain log messages including messages sent between the game server and GameLift Local.

14. Launch the game client. You can find it in the build location you specified when building the sample game client.

15. In the GameLift Sample Game, provide an email and password and then select Log In. The email and password are not validated and are not used.

16. In the GameLift Sample Game, choose Start. The game client will look for a game session. If one cannot be found, it will create a game session. The game client then starts the game session. You can see game activity in the logs.

The messages below are from the sample game server:

```
... 2021-09-15T19:55:3495 PID:20728 Log :) GAMELIFT AWAKE
2021-09-15T19:55:3514 PID:20728 Log :) SDK VERSION: 4.0.2
2021-09-15T19:55:3556 PID:20728 Log :) SERVER IS IN A GAMELIFT FLEET
2021-09-15T19:55:3577 PID:20728 Log :) GAMELIFT HEALTH CHECK REQUESTED (HEALTHY)
...
2021-09-15T19:55:3636 PID:20728 Log :) LISTENING ON PORT 33430
...
2021-09-15T19:56:2464 PID:20728 Log :) GAMELIFT SESSION REQUESTED
2021-09-15T19:56:2468 PID:20728 Log :) GAME SESSION ACTIVATED
2021-09-15T19:56:3578 PID:20728 Log :) GAMELIFT HEALTH CHECK REQUESTED (HEALTHY)
2021-09-15T19:57:3584 PID:20728 Log :) GAMELIFT HEALTH CHECK REQUESTED (HEALTHY)
2021-09-15T19:58:0334 PID:20728 Log SERVER: Frame: 8695 Connection accepted: playerIdx 0 joined
2021-09-15T19:58:0335 PID:20728 Log SERVER: Frame: 8696 Connection accepted: playerIdx 1 joined
```

The messages below are messages from GameLift Local:

```
12:55:26,000 INFO || - [SocketIOServer] main - Session store / pubsub factory used: MemoryStoreFactory (local session store only)
```
17. In the game client, choose Quit or close the window to stop the game client.

18. In Unity, in the Local Testing window, choose Stop or close the game server windows to stop the server.
Managing GameLift Hosting Resources

This section provides detailed information on setting up GameLift managed resources to run your game servers and enable them to host game sessions for players. Whether you’re deploying a fully custom game server or working with Realtime Servers, you need to configure and deploy resources, scale capacity to meet player demand, and set up a way to locate available resources to host new game sessions.

**Tip**

Learn more about ways to explore Amazon GameLift features, including Realtime Servers, using sample games (p. 27).

Here’s a summary of process required to set up GameLift resources for managed hosting. The topics in this section discuss how to design and build each resource.

1. Start by uploading your game server code to the GameLift service in the AWS Regions where you plan to set up fleets. Your code might be a custom game server build or a Realtime configuration script.

2. Next, create a **fleet** of computing resources in one or more Regions. For each fleet, you specify the build/script to deploy, choose the type of instances (virtual computing machine) to use, configure them to run your game servers, and select the location(s) to deploy them to. When a fleet is created, the build/script is installed on each instance in the fleet, ready to host game sessions.

3. Adjust fleet capacity to host game sessions as needed to meet player demand. Optionally, set up an auto-scaling policy.

4. Create an **alias** for each fleet. Although this step is optional, it is a best practice to abstract fleet identifiers to maintain uninterrupted game server availability then upgrading game builds and fleets.

5. With a set of fleets deployed, set up a game session placement mechanism to find available game servers to host new game sessions. The most common method is to set up a **game session queue**. A queue is able to search for available game servers across multiple locations, fleet types, and instance types. It uses FleetIQ algorithms to select the best possible game server, based on a defined set of priorities, and starts a new game session.

About GameLift hosting resources

This topic describes the key resources that you work with when using the GameLift service to host your games. Learn more about how GameLift works in *How GameLift works* (p. 3).

The following diagram illustrates the basic structure of GameLift resources and how they relate to each other.
About GameLift hosting resources

Game server code

- **Build** – Your custom-built game server software that runs on GameLift and hosts game sessions for your players. A game build represents the set of files that run your game server on a particular operating system. You can have many different builds, such as for different flavors of your game. The game build must be integrated with the GameLift service. You upload game build files to the GameLift service in the Regions where you plan to set up fleets. See more details in Uploading builds and scripts to GameLift (p. 97).

- **Script** – Your configuration and custom game logic for use with Realtime Servers. Realtime Servers are provided by GameLift to use instead of a custom-built game server. You configure Realtime Servers for your game clients by creating a script using JavaScript, and add custom game logic as appropriate to host game sessions for your players. You upload the Realtime script to the GameLift service in the Regions where you plan to set up fleets. See more details about builds and scripts in Uploading builds and scripts to GameLift (p. 97).

Fleet

A collection of virtual computing resources, called instances, that run your game servers and host game sessions for your players. You define the type of instances to use (hardware/software configuration, Spot or On-Demand availability), select a custom game build or a Realtime server script to deploy, and specify one or more locations to deploy instances to. You also provide a runtime configuration, which controls how game servers are run on each instance, and configure other game management properties for the fleet. You manage a fleet's capacity, which determines how many game sessions and players it can support, using either manual or automatic scaling. You can create multiple single-location fleets, which deploy instances only in the Region where they are created (referred to as the "home" Region). Alternatively, you can create a multi-location fleet, which can deploy instances to any other GameLift-supported Regions as "remote" locations as well as the home Region. For information on AWS Regions that support single-location and multi-location fleets, see Using Amazon GameLift in AWS Regions (p. 25). See more details about fleets in Setting up GameLift fleets (p. 106).

Game session queue

A game session placement mechanism that receives requests for new game sessions and searches for an available game server to host the new game session. When an available game server is located, the game session queue prompts the server to start a new game session and returns connection information back to the requester. You provide a list of fleets where the queue can search for available game servers, which can vary by instance type, availability types (Spot or On-Demand) and geographical locations. You can also provide a set of placement priorities, based on player latency, hosting cost, and location preference, which FleetIQ can use to select the "best possible" game server to host a requested game session. The game session queue is an essential element of the
Create resources with AWS CloudFormation

You can use AWS CloudFormation to manage your GameLift resources (p. 89). The GameLift console and CLI commands are useful tools to create and update individual resources. But with AWS CloudFormation you can manage an entire set of resources to support your game hosting. In AWS CloudFormation, you create a template that models each resource and then use the template to create your resources. To update resources, you make the changes to your template and use AWS CloudFormation to implement the updates. You can organize your resources into logical groups, called stacks and stack sets.

Using AWS CloudFormation to maintain your GameLift hosting resources offers a more efficient way to manage sets of AWS resources. You can use version control to track template changes over time and coordinate updates made by multiple team members. You can also reuse templates. For example, when deploying a game across multiple Regions, you might use the same template to create identical resources in each Region. You can also use these templates to deploy the same sets of resources in another partition.

For more information about AWS CloudFormation, see the AWS CloudFormation User Guide. To view template information for GameLift resources, see the Amazon GameLift Resource Type Reference.

The following topics cover best practices with using AWS CloudFormation with GameLift and present some recommendations for structuring your resource templates.

Best Practices

For detailed guidance on using AWS CloudFormation, see the AWS CloudFormation Best Practices in the AWS CloudFormation User Guide. In addition, these best practices have special relevance with GameLift.

- **Consistently manage your resources using through AWS CloudFormation only.** This is a core AWS CloudFormation best practice, but it bears repeating. If you change your resources outside of AWS CloudFormation, such as using the GameLift console, GameLift API calls or CLI commands, your resources will get out of sync with your resource templates. This may result in unexpected consequences the next time you update your resources using the AWS CloudFormation templates.

- **Use AWS CloudFormation stacks and stack sets to efficiently manage multiple resources.**
  - Use stacks to manage groups of connected resources. AWS CloudFormation intelligently updates resources in a stack that reference each other based on whether resource properties are mutable. For example, suppose you have a stack that contains a build, a fleet that references the build, and an alias that references the fleet. In GameLift, the relationship between builds and fleets is immutable. If you update your template to replace a build, AWS CloudFormation also replaces the fleets that are connected to the replaced build. AWS CloudFormation then updates the existing aliases to point to the new fleets. For more information, see Working with Stacks in the AWS CloudFormation User Guide.
• Use AWS CloudFormation stack sets if you're deploying identical stacks across multiple regions or AWS accounts. For more information, see Working with Stack Sets in the AWS CloudFormation User Guide.

• If you are using Spot Instances, include an On-Demand Fleet as a back-up. We recommend setting up your templates with two fleets in each region, one fleet with Spot Instances, and one fleet with On-Demand Instances. GameLift's FleetIQ feature ensures that game sessions are always placed first with viable Spot Instances. The On-Demand fleet acts as fallback in the event that the Spot Fleet is not available.

• Group your Region-specific resources and global resources into separate stacks when you are managing resources in multiple Regions. Some resources, such as GameLift fleets, can only reference other resources in the same Region. Other resources, such as GameLift queues, can reference resources in other regions. Placing them in separate stacks gives you more flexibility in where you place your global resources.

• Place your global resources in close proximity to the services that use it. When you are placing global resources, keep in mind how these resources are being accessed. Resources like queues and matchmaking configurations tend to receive a high volume of requests from specific sources, such as a backend service. By placing your resources in close proximity to the source of those requests, you minimize the request travel time and can improve overall performance.

• Place your matchmaking configuration in the same Region as the game session queue that it uses. Matchmaking configurations send requests for new game sessions to the queue, so placing these resources together also helps to optimize system performance.

• Create a separate alias for each fleet in the stack. Aliases make it much easier to transition player traffic when replacing game builds and fleets.

Using AWS CloudFormation Stacks

The following are recommended structures to use when setting up AWS CloudFormation stacks for GameLift-related resources. Your optimal stack structure varies depending on whether you are deploying your game in only one Region or across multiple Regions.

Stacks for a Single Region

To manage GameLift resources in a single Region, we recommend a two-stack structure:

• Support stack – This stack contains resources that your GameLift resources depend on. At a minimum, this stack should include the S3 bucket where you store your custom game server or Realtime script files. The stack should also include an IAM role that gives GameLift permission to retrieve your files from the S3 bucket when creating a GameLift build or script resource. This stack might also contain other AWS resources that are used with your game, such as DynamoDB tables, Amazon Redshift clusters, and Lambda functions.

• GameLift stack – This stack contains all of your GameLift resources, including the build or script, a set of fleets, aliases, and game session queue. AWS CloudFormation creates a build or script resource with files stored in the S3 bucket location and deploys the build or script to one or more fleet resources. Each fleet should have a corresponding alias. The game session queue references some or all of the fleet aliases. If you are using FlexMatch for matchmaking, this stack also contains a matchmaking configuration and rule set.

The diagram below illustrates a two-stack structure for deploying resources in a single AWS Region.
Stacks for Multiple Regions

When deploying your game in more than one Region, keep in mind how resources can interact across Regions. Some resources, such as GameLift fleets, can only reference other resources in the same Region. Other resources, such as a GameLift queue, are Region agnostic. To manage GameLift resources in multiple Regions, we recommend the following structure.

- **Regional support stacks** – These stacks contain resources that your GameLift resources depend on. This stack must include the S3 bucket where you store your custom game server or Realtime script files. It might also contain other AWS resources for your game, such as DynamoDB tables, Amazon Redshift clusters, and Lambda functions. Many of these resources are Region specific, so you must create them in every Region. GameLift also needs an IAM role that allows access to these support resources. Because an IAM role is Region agnostic, you only need one role resource, placed in any Region and referenced in all of the other support stacks.

- **Regional GameLift stacks** – This stack contains the GameLift resources that must exist in each region where your game is being deployed, including the build or script, a set of fleets, and aliases. AWS CloudFormation creates a build or script resource with files in an S3 bucket location, and deploys the build or script to one or more fleet resources. Each fleet should have a corresponding alias. The game session queue references some or all of the fleet aliases. You can maintain one template to describe this type of stack and use it to create identical sets of resources in every Region.

- **Global GameLift stack** – This stack contains your game session queue and matchmaking resources. These resources can be located in any Region and are usually placed in the same Region. The queue can reference fleets or aliases that are located in any Region. To place additional queues in different Regions, create additional global stacks.

The diagrams below illustrate a multistack structure for deploying resources in several AWS Regions. The first diagram shows a structure for a single game session queue. The second diagram shows a structure with multiple queues.
Updating Builds

GameLift builds are immutable, as is the relationship between a build and a fleet. As a result, when you update your hosting resources to use a new set of game build files, the following need to happen:

- Create a new build using the new set of files (replacement).
- Create a new set of fleets to deploy the new game build (replacement).
- Redirect aliases to point to the new fleets (update with no interruption).
For more information, see Update Behaviors of Stack Resources in the AWS CloudFormation User Guide.

**Deploy Build Updates Automatically**

When updating a stack containing related build, fleet and alias resources, the default AWS CloudFormation behavior is to automatically perform these steps in sequence. You trigger this update by first uploading the new build files to a new S3 location. Then you modify your AWS CloudFormation build template to point to the new S3 location. When you update your stack with the new S3 location, this triggers the following AWS CloudFormation sequence:

1. Retrieves the new files from S3, validates the files, and creates a new GameLift build.
2. Updates the build reference in the fleet template, which triggers new fleet creation.
3. After the new fleets are active, updates the fleet reference in the alias, which triggers the alias to update to target the new fleets.
4. Deletes the old fleet.
5. Deletes the old build.

If your game session queue uses fleet aliases, player traffic is automatically switched to the new fleets as soon as the aliases are updated. The old fleets are gradually drained of players as game sessions end. Auto-scaling handles the task of adding and removing instances from each set of fleets as player traffic fluctuates. Alternatively, you can specify an initial desired instance count to quickly ramp up for the switch and enable auto-scaling later.

You can also have AWS CloudFormation retain resources instead of deleting them. For more information, see RetainResources in the AWS CloudFormation API Reference.

**Deploy Build Updates Manually**

If you want to have more control over when new fleets go live for players, you have some options. You can choose to manage aliases manually using the GameLift console or the CLI. Alternatively, instead of updating your build template to replace the build and fleets, you can add a second set of build and fleet definitions to your template. When you update the template, AWS CloudFormation creates a second build resource and corresponding fleets. Since the existing resources are not replaced, they are not deleted, and the aliases remain pointing at original fleets.

The main advantage with this approach is that it gives you the flexibility. You can create separate resources for the new version of your build, test the new resources, and control when the new fleets go live to players. A potential drawback is that it requires twice as many resources in each Region for a brief period of time.

The following diagram illustrates this process.
How Rollbacks Work

When executing a resource update, if any step is not completed successfully, AWS CloudFormation automatically initiates a rollback. This process reverses each step in sequence, deleting the newly created resources.

If you need to manually trigger a rollback, change the build template's S3 location key back to the original location and update your stack. A new GameLift build and fleet are created, and the alias switches over to the new fleet after the fleet is active. If you are managing aliases separately, you need to switch them to point to the new fleets.

For more information about how to handle a rollback that fails or gets stuck, see Continue Rolling Back an Update in the AWS CloudFormation User Guide.

Uploading builds and scripts to GameLift

Before deploying your GameLift-enabled multiplayer game servers for hosting with the GameLift service, you need to upload your game server files. The topics in this section provide guidance on preparing and uploading custom game server build files or Realtime Servers server script files. You can When you upload files, you create a GameLift build or script resource, which you then deploy on fleets of hosting resources.

Topics
- Upload a custom server build to GameLift (p. 97)
- Upload a Realtime Servers script to GameLift (p. 103)

Upload a custom server build to GameLift

Once your game server has been integrated with GameLift, upload the build files to the managed GameLift service so that it can be deployed for game hosting. This topic covers how to package your game's build files, create an optional build install script, and then upload the files using either the AWS CLI or the AWS SDK.

Add a build 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 to the instance file structure. 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). Permission failures related to the install script generate an event message that indicates 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 GameLift fleet instances (p. 126).
- The install script cannot rely on a VPC peering connection. If you're planning to set up VPC peering when creating fleets for this build, this connection is not available until after the build is installed on fleet instances.

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.

```
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.

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

This example install.sh file shows how you can use the Amazon CloudWatch agent to collect system-level metrics, custom metrics, and handle log rotation.

```
sudo yum install -y amazon-cloudwatch-agent
sudo yum install -y collectd
cat <<'EOF' > /tmp/config.json
{
  "agent": {
    "metrics_collection_interval": 60,
    "run_as_user": "root",
    "credentials": {
      "role_arn": "arn:aws:iam::account#:role/rolename"
    }
  }
```

```
Since GameLift is running in a service VPC, you must grant permissions for GameLift to assume the role on your behalf. You must create a role that includes CloudWatchAgentAdminPolicy and use the role when you create a fleet.

**Package your game build files**

Before uploading your GameLift-enabled game server to the GameLift service for hosting, you need to package the game build files into a build directory. This directory must include all components required to run your game servers and host game sessions, 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. 22)).
- **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 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.

You can set up any application in your build, including your install script, to securely access your resources on other AWS services. Learn more about possible ways to do this in [Communicate with other AWS resources from your fleets](#) (p. 55).

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.
Note

If you’re storing your game build files in an Amazon S3 bucket for uploading, you need to package the build files into a `.zip` file. See instructions for uploading using this method in Create a build with files in Amazon S3 (p. 101).

Create a GameLift build

When creating a build and uploading your files, you have a couple of options:

- Create a build from a file directory (p. 100). This is the simplest and most commonly used method.
- Create a build with files in Amazon S3 (p. 101). With this option, you can manage your build versions in S3 under your AWS account.

With both methods, a new build resource is created with a unique build ID and other metadata. The build is placed in **Initialized** status. Once GameLift successfully acquires the game server files, the build moves to **Ready** status. At this point, you can deploy it by creating a new GameLift fleet (see Deploy a GameLift fleet with a custom game build (p. 112)).

When you create a fleet, you specify which build to deploy to the fleet. When GameLift sets up the new fleet, it downloads the build files to each fleet instance, and installs it based on the build install script (if one is provided). Build files are installed on the instances in the following locations:

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

Create a build from a file 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 GameLift and uploads files from a location that you specify.

1. **Send an upload request.** In a command line window, enter 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** – Specify the game server build's OS. When a new fleet is created for this build, fleet instances are configured with the appropriate OS. GameLift supports several Windows and Linux varieties. This parameter is optional. If an operating system is not specified, GameLift uses the default value (`WINDOWS_2012`). Once the build is created, this value cannot be updated later.
   - **Name** – Provide a descriptive name for the new build. Build name does not need to be unique, and you can update this value at any time by updating the build resource.
   - **Build version** – Use this optional field to specify version details for the build files. Since each new version of your game server requires a new build resource, this information can provide a valuable differentiator.
   - **Region** – Identify the GameLift-supported region where you want to create your build. You must create the build in the region where you plan to deploy fleets. If you’re deploying your game in multiple regions, you must create a build in each region.

   **Note**

   If you work in multiple regions, it is always a good idea to check your current default region. In the AWS console, the current region is always displayed in the upper right
corner, with a dropdown list of available regions to select from. If you're using the AWS CLI, check your current default using the configure get (AWS configure get region). Use the command configure set (AWS configure set region [region name]) to change your default region.

In response to your upload request, the 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:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS gamelift upload-build --operating-system AMAZON_LINUX --build-root ~/mygame --name &quot;My Game Nightly Build&quot; --build-version &quot;build 255&quot; --region us-west-2</td>
<td>Upload a custom server build with Amazon Linux operating system in the US West 2 region</td>
</tr>
<tr>
<td>AWS gamelift upload-build --operating-system WINDOWS_2012 --build-root &quot;C:\mygame&quot; --name &quot;My Game Nightly Build&quot; --build-version &quot;build 255&quot; --region us-west-2</td>
<td>Upload a custom server build with Windows 2012 operating system in the US West 2 region</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS gamelift describe-build --build-id [build ID returned with the upload request] --region [region name]</td>
<td>View the build record including current status</td>
</tr>
<tr>
<td>Example</td>
<td>AWS gamelift describe-build --build-id &quot;build-3333cccc-44dd-55ee-66ff-7777aaaa88bb&quot; --region us-west-2</td>
</tr>
</tbody>
</table>

   In response to your request, the 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

To create a game build with packaged game server files that are stored in an Amazon S3 bucket under your AWS account, use the AWS CLI command create-build. This operation creates a new build in GameLift and acquires your build files from the Amazon S3 bucket that you specify. GameLift will report the SizeOnDisk as 0. The size of the file can be found in Amazon S3.

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 under your AWS account. Take note of the bucket label and the file name; you’ll need these when creating a GameLift build.

2. **Give GameLift access to your build files.** Follow the instructions at Set up a role for GameLift access (p. 21) to create an IAM role. A role specifies which entities (such as the GameLift service) can assume the role and defines a set of permissions for limited access to your AWS resources. 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.

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 GameLift where your build files are stored. You must have IAM PassRole permission, as described in IAM policy examples for GameLift (p. 18).
In this request, specify an Amazon S3 location, including the following information, which you collected when setting up your bucket and access role. Enter the following command and parameters.

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

- **Name** – Provide a descriptive name for the new build. Build name does not need to be unique, and you can update this value at any time by updating the build resource.
- **Operating system** – Specify the game server build's OS. When a new fleet is created for this build, fleet instances are configured with the appropriate OS. GameLift supports several Windows and Linux varieties. This parameter is optional. If an operating system is not specified, GameLift uses the default value (WINDOWS_2012). Once the build is created, this value cannot be updated later.
- **Build version** – Use this optional field to specify version details for the build files. Since each new version of your game server requires a new build resource, this information can provide a valuable differentiator.
- **S3 location**
  - **Bucket** – Name of the S3 bucket that contains your build. Example: "my_build_files".
  - **Key** – Name of the .zip file that contains your build files. Example: "my_game_build_7.0.1, 7.0.2".
- **Role ARN** – ARN assigned to the IAM role that you created. Example: "arn:aws:iam::111122223333:role/GameLiftAccess". For an example policy, see Access a game build file in Amazon S3 (p. 198).
- **Region** – Identify the GameLift-supported region where you want to create your build. You must create the build in the region where you plan to deploy fleets. If you're deploying your game in multiple regions, you must create a build in each region.

**Note**
It is always a good idea to check your current default AWS Region. In the AWS console, the current Region is displayed in the upper right corner with a dropdown list of available regions to select from. If you're using the AWS CLI, check your current default using `aws configure get`. Use the command `aws configure set region [region name]` to change your default Region.

**Example:**
```
AWS gamelift create-build --operating-system WINDOWS_2012 --storage-location "Bucket=my_game_build_files,Key=mygame_build_101.zip,RoleArn=arn:aws:iam::111122223333:role/gamelift" --name "My Game Nightly Build" --build-version "build 101" --region us-west-2
```

In response to your request, the GameLift service returns the newly created build record, including the build's current status.

### Update your build files

Once a GameLift build has been created, the build files associated with it cannot be changed. Instead, you must create a new 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 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.
Upload a Realtime Servers script to GameLift

When you're ready to deploy Realtime Servers for your game, upload completed Realtime server script files to GameLift. This is done by creating a GameLift script resource and specifying the location of your script files. You can also update server script files that are already deployed by uploading new files for an existing script resource.

When you create a new script resource, GameLift assigns a unique script ID (example: `script-1111aaaa-22bb-33cc-44dd-5555eeee66ff`) and uploads a copy of the script files. Upload time depends on the size of your script files and connection speed.

Once the script resource is created, it can be deployed with a new GameLift Realtime Servers fleet. GameLift installs your server script onto each instance in the fleet, placing the script files at the following location: `/local/game`.

To troubleshoot fleet activation problems that might be related to the server script, see Debug GameLift fleet issues (p. 124).

Package script files

Your server script can include one or multiple files combined into a single `.zip` file for uploading. The zip file must contain all files that your script depends on to run.

For uploading, you can store your zipped script files in either a local file directory or in an Amazon S3 bucket.

Upload script files from a local directory

If you have your script files stored locally, you can opt to upload them to GameLift from there. Use either the GameLift console or the AWS CLI tool to create the script resource.

GameLift Console

To create a script resource

1. Open the Amazon GameLift console at https://console.aws.amazon.com/gamelift/
2. Use the GameLift main menu to open the Scripts: Create script page. This page contains the script creation form.
3. In Script configuration, enter a script name and version information. Because the content of a script can be updated, version data can be helpful in tracking the updates.
4. In Script code, choose the Script type "Zip file". This option lets you specify a zip file that is stored in a local directory.
5. Browse for the zip file that contains your script and select it.

6. After you finish defining the new script record, click Submit. GameLift assigns an ID to the new script and begins uploading the designated zip file. You can view the new script record, including status, on the console's Scripts page.

AWS CLI

To create a script with the AWS CLI, open a command line window and use the create-script command to define the new script and upload your server script files. See complete documentation on this command in the AWS CLI Command Reference. Get and install the AWS Command Line Interface tool.

To create a script resource

1. Place the zip file into a directory where you can make calls to the AWS CLI.
2. In a command line window, switch to the directory where the zip file is located.
3. Enter the create-script command and parameters. For the -zip-file parameter, be sure to prepend the string "fileb://" to the name of the zip file. It identifies the file as binary and ensures that the compressed content will processed correctly.

   AWS gamelift create-script --name [user-defined name of script] --script-version [user-defined version info] --zip-file fileb://[name of zip file] --region [region name]

   In response to your request, the Amazon GameLift service returns the new script object.

   Examples:

   AWS gamelift create-script --name My_Realtime_Server_Script_1 --script-version 1.0.0 --zip-file fileb://myrealtime_script_1.0.0.zip --region us-west-2

   You can view the new script by calling describe-script or by viewing it in the Amazon GameLift console.

Upload script files in Amazon S3

You can opt to store your script files in an Amazon S3 bucket and upload them to GameLift from there. When you create your script, you specify the S3 bucket location and Amazon GameLift acquires your script files directly from S3.

To create a script resource

1. Store your script files in an Amazon S3 bucket. Create a .zip file containing your server script files and upload it to an Amazon S3 bucket in an AWS account that you control. Take note of the bucket name and the file name (also called the "key"); you'll need these when creating a GameLift script.
   
   Note
   GameLift currently does not support uploading from Amazon S3 buckets with names that contain a dot (.)

2. Give GameLift access to your script files. Follow the instructions at Set up a role for GameLift access (p. 21) to create an IAM role that allows GameLift to access the Amazon S3 bucket containing your server script. Once you've created the role, take note of the new role's Amazon Resource Name (ARN), which you'll need when creating a script.
3. **Create a script.** Use either the Console or the AWS CLI to create a new script record. To make this request, you must have IAM PassRole permission, as described in IAM policy examples for GameLift (p. 18).

**GameLift Console**

1. Use the GameLift main menu to open the **Scripts: Create script** page to access the script creation form.
2. In **Script configuration**, enter a script name and version information. Because the content of a script can be updated, version data can be helpful in tracking the updates.
3. In **Script code**, choose the **Script type** "User storage". This option lets you specify the S3 bucket containing your script file.
4. Enter the storage location information for your S3 bucket:
   - S3 bucket – The bucket name.
   - S3 key – The name of the file (zipped file containing your server script) in the bucket.
   - S3 role ARN – The ARN value for the IAM role that you created in Step 2.
   - S3 object version – (optional) A specific version number for the S3 file. This is used only when the S3 bucket has object versioning turned on AND when you want to specify a version other than the latest one.
5. Once you finished defining the new script record, click **Submit**. GameLift assigns an ID to the new script and begins uploading the designated zip file. You can view the new script record, including status, on the console's **Scripts** page.

**AWS CLI**

Use the `create-script` command to define the new script and upload your server script files. See complete documentation on this command in the [AWS CLI Command Reference](https://docs.aws.amazon.com/cli/latest/reference/gamelift/create-script.html). Get and install the AWS Command Line Interface tool.

1. Open a command line window and switch to a directory where you can use the AWS CLI tool.
2. Enter the `create-script` command with the following parameters: `--name`, `--script-version`, and `--storage-location`. The storage location parameter specifies the Amazon S3 bucket location of your script files.

```bash
AWS gamelift create-script --name [user-defined name of script] --script-version [user-defined version info] --storage-location "Bucket=[S3 bucket name],Key=[name of zip file in S3 bucket],RoleArn=[Access role ARN]" --region [region name]
```

In response to your request, the Amazon GameLift service returns the new script object.

*Examples:*

```bash
AWS gamelift create-script --name My_Realtime_Server_Script_1 --script-version 1.0.0 --storage-location "Bucket=gamelift-script,Key=myrealtime_script_1.0.0.zip,RoleArn=arn:aws:iam::123456789012:role/S3Access" --region us-west-2
```

You can view the new script by calling `describe-script` or by viewing it in the Amazon GameLift console.
Update script files

You can update the metadata for a script resource using either the GameLift Console or the AWS CLI command update-script.

You can also update the script content for a script resource. Updated script content is deployed to all fleet instances that use the updated script resource. Once deployed, the updated script is used when starting new game sessions. It does not affect game sessions that are already running at the time of the update.

To update script files:

- For script files stored locally: Use either the GameLift Console or the AWS CLI command update-script to upload the updated script .zip file.
- For script files stored in an Amazon S3 bucket: Upload the updated script files to the bucket. Amazon GameLift periodically checks for updated script files and acquires them directly from the S3 bucket.

You also can change the location where script content is located without having to create a new script record.

Setting up GameLift fleets

This section provides detailed help with designing, building, and maintaining fleets for use with a GameLift managed hosting solution. GameLift managed fleets are used to deploy custom game servers and Realtime Servers.

A fleet represents your hosting resources in the form of a set of EC2 virtual computing machines, called instances. A fleet's locations determines where, geographically, instances are deployed to host game sessions for your players. The size of a fleet, and the number of game sessions and players it can support, depends on the number of instances you give it, which you can adjust either manually or by auto-scaling.

Many games in production use more than one fleet. You need multiple fleets to, for example, have more than one version of your game server running simultaneously, provide back-up capacity for Spot fleets, or to build in redundancy.

To learn about how to create fleets that best suit your game needs, start with GameLift fleet design guide (p. 106). Once you get a fleet up and running, you can manage fleet capacity (p. 129), create a fleet alias (p. 122), and add the fleet to a game session queue (p. 137).

Topics

- GameLift fleet design guide (p. 106)
- Create a new GameLift fleet (p. 112)
- Manage your GameLift fleets (p. 120)
- Add an alias to a GameLift fleet (p. 122)
- Debug GameLift fleet issues (p. 124)
- Remotely access GameLift fleet instances (p. 126)

GameLift fleet design guide

This design guide covers key decision points when creating a fleet of hosting resources for use with a managed GameLift solution. Select the right combination of hosting resources and learn how to configure them to best suit to your game.
Tip
Learn more about ways to explore Amazon GameLift features, including Realtime Servers, using sample games (p. 27).

Topics
- Choosing computing resources (p. 107)
- Managing how game servers are launched for hosting (p. 109)
- Using Spot Instances with GameLift (p. 110)

Choosing computing resources

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 (using a runtime configuration). When 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 based on demand. All instances in a fleet use the same type of resources and the same runtime configuration. You can edit a fleet's runtime configuration and other fleet properties, but the type of resources cannot be changed.

When choosing resources for a fleet, you need to consider several factors, including game operating system, instance type (the computing hardware), and whether to use On-Demand or Spot Instances. Keep in mind that hosting costs with GameLift primarily depend on the type of instances you use. Learn more about GameLift pricing.

Operating systems

GameLift supports game server builds that run on either Microsoft Windows or Amazon Linux (see supported game server operating systems (p. 22)). When uploading a game build to GameLift, you specify the operating system for the game. When you create a fleet to deploy the game build, 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 GameLift you have a wide range of instance type options to choose from. To learn more about the capabilities of each instance type, see Amazon EC2 Instance Types.

Consider the geographic locations where you plan to deploy your game servers. Instance type availability varies by AWS Region. In addition, check the service limits on the number of instance an AWS account can use, which are applied per instance type, per Region. You can view a list of available instance types and service limits using the methods described in Instance service quotas (p. 108).

When choosing from available instance types 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. 109).
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 availability 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 you can 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 can offer a highly cost-efficient alternative to On-Demand Instances by taking advantage of currently unused AWS computing capacity. The availability of Spot Instances varies, making them less expensive but also generally less viable for game hosting without GameLift. When using Spot Instances, it is important to understand the impact of these characteristics. Spot prices fluctuate based on the supply and demand for each instance type in each Region. Also, Spot Instances can be interrupted by AWS (with a two-minute notification) whenever AWS needs the capacity back.

GameLift provides a set of FleetIQ algorithms to boost the viability of Spot Instances for game hosting. FleetIQ is used in the game session placement process to find the best available location to place each new game session (see Setting up GameLift queues for game session placement (p. 137)). When a queue has Spot fleets, FleetIQ can prioritize low-cost Spot fleets but eliminates fleets and locations where there’s an elevated chance of interruption based historical patterns.

You can view pricing history for any instance type in the GameLift console. The Spot history page graphs On-demand and Spot pricing and calculates the relative cost savings with Spot Instances. Use the controls to select an instance type, operating system, and a time range.

You can also evaluate FleetIQ performance using queue metrics, as well as instance-specific metrics on Spot Instances. Learn more about GameLift metrics (p. 170).

Learn more about how to use Spot Instances in the Using Spot Instances with GameLift (p. 110).

Instance service quotas

AWS limits the total number of Amazon EC2 instances (On-Demand or Spot) that your AWS account can use with GameLift. In addition, there’s a maximum limit on each instance type, which varies by Region and location. You can extend these quotas by request.

You can access information on service quotas using the following methods:

- **AWS endpoints and quotas** lists general quota information for GameLift, as well as other AWS services.
- Look up specific quota values for instance types per Region in the GameLift console. Go to the Service Limits page and select a Region to view quotas for the available instance types. This page also displays current usage values for each instance type in the Region.
- Retrieve quota values for instance types per Region with the AWS CLI command describe-ec2-instance-limits. This action also returns the number for instances currently active in the Region.

For multi-location fleets, information on instance availability and limits depends on a combination of the fleet’s home Region and (if relevant) a selected remote location. Information will be different, for example, for m5.large instances that are deployed to the sa-east-1 Region in (1) a fleet that resides in that Region, versus (2) a fleet that resides in another Region and deploys instances to sa-east-1 as a remote location.
Managing how game servers are launched for hosting

You can set up a fleet’s runtime configuration to run multiple game server processes per instance. By running multiple processes on each instance can help you use your hosting resources more efficiently and potentially reduce overall hosting costs.

How a fleet manages multiple processes

GameLift uses a fleet’s runtime configuration to determine the type and number of processes to run on each instance in the fleet. At a minimum, a runtime configuration contains one server process configuration that represents one game server executable. You can also define additional server process configurations to run other types of processes related to your game. Each server process configuration contains the following information:

- The file name and path of an executable in your game build.
- (Optional) Parameters to pass to the process on launch.
- The number of processes to run concurrently.

When an instance in the fleet is activated, it immediately launches the set of server processes that are defined in the runtime configuration. With multiple processes, each process launch is staggered by a few seconds, so instances with many processes might take a few minutes to achieve full activation. Processes have a limited life span. As they terminate, instances continual launch new processes in order to maintain the number and type of server processes as defined in the current runtime configuration.

You can change the runtime configuration at any time by adding, changing, or removing server process configurations. Each instance regularly checks for updates to the fleet’s runtime configuration, ensuring that changes are implemented quickly. Here’s how GameLift adopts runtime configuration changes.

1. Before an instance checks that it is running the correct type and number of server processes, the instance sends a request to the GameLift service for the latest version of the runtime configuration.
2. The instance checks its active processes against the latest runtime configuration and handles updates as follows:
   - If the updated runtime configuration removes a server process type: Active server processes of this type continue to run until they end, and are not replaced when they terminate.
   - If the updated runtime configuration decreases the number of concurrent processes for a server process type: Excess server processes of this type continue to run until they end, and are not replaced when they terminate.
   - If the updated runtime configuration adds a new server process type or increases the concurrent processes for an existing type: New server processes are started immediately, up to the GameLift maximum for server processes on an instance. In this case, new server processes are launched only as existing processes end.

Optimizing a fleet for multiple processes

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

- Create a build (p. 97) that contains all of the game server executables that you want to deploy to a fleet and upload it to GameLift. All game servers in a build must run on the same platform and be integrated with GameLift using the 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 that connect to games hosted on this fleet must be integrated using the AWS SDK version 2016-08-04 or later.
In addition, follow these game server integration recommendations to optimize fleet performance:

- Handle server process shutdown scenarios to ensure that GameLift can recycle processes efficiently. Without these, server processes won't shut down until they fail, and runtime configuration updates may be slow to implement.
- Add a shutdown procedure to your game server code that calls server API `ProcessEnding()`.
- Implement the callback function `OnProcessTerminate()` in your game server code to gracefully handle termination requests from GameLift.
- Make sure that unhealthy server processes are shut down and relaunched quickly. Define what "healthy" and "unhealthy" mean for your game and maintain each process's health status. Report this status back to GameLift by implementing the `OnHealthCheck()` callback function in your game server code. GameLift automatically shuts down server processes that are reported unhealthy for three consecutive reports. If you don't implement `OnHealthCheck()`, GameLift assumes that a server process is healthy unless the process fails to respond to a communication. As a result, poorly performing server processes can continue to exist, using up resources until they finally fail.

### Choosing the number of processes per instance

When deciding on the number of concurrent processes to run on an instance, there are three limits to keep in mind:

- 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 in order to maintain 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 GameLift will never run more concurrent processes than the total number configured. This means that the transition from old runtime configuration to new might happen gradually, with new processes starting only as existing processes end. For example, say 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`. Because instances in the fleet are already running 10 processes, each instance cannot start a process with the new launch parameters until a process with the old launch parameters ends.

### Using Spot Instances with GameLift

When setting up your hosting resources, you have the option of using Spot Instances, On-Demand Instances, or a combination. Learn more about how GameLift uses Spot Instances in On-Demand versus Spot Instances (p. 108). 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 add Spot fleets to your existing game session queues for matchmaking placements.

### To set up game and hosting resources for Spot fleets:

1. **Design a game session queue with a "Spot-optimized" configuration.** Managing game session placement with a queue is always a best practice, and it's required when using Spot Instances. For this step, you need to identify which Regions/locations the queue will place game sessions into, and
select the right combination of instance types (hardware) for your game servers. This information tells you the types of fleets you'll need to set up. For help designing a queue that optimizes Spot availability and resiliency, see Best practices for queues with Spot fleets (p. 139).

2. **Create the fleets for your Spot-optimized queue.** Based on your queue design, create fleets to deploy your game servers to your desired locations and instance types. Spot fleets and On-Demand fleets can have identical configurations other than the fleet type designation. See Deploy a GameLift fleet with a custom game build (p. 112) 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 Spot or On-Demand fleets in a long list of fleet names, such as when adding fleets to a queue.

3. **Create your game session queue.** After you've set up the fleets based on your Spot-optimized queue design, you can create the queue. Add the fleet destinations, configure the game session placement process, and define placement priorities. See Create a game session queue (p. 148) for help creating and configuring the new queue.

4. **Enable your game client or client service to request new game sessions using the Spot-optimized queue.** If you're not already doing so, update your game client code to request new game sessions using a queue for game session placement. This approach enables the use of FleetIQ to avoid resources with a high chance of interruption and select the best available location based on your defined priorities (such as player latency, hosting cost, and geographical location). Queues also provide fallback hosting resources in the unlikely event that a preferred location is not available. For help implementing game session placements in your game client, see Create Game Sessions (p. 59).

5. **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 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 and players to a new placement. See Respond to a server process shutdown notification (p. 54) for help implementing `onProcessTerminate()`.

   **Note**
   AWS makes every effort to provide the notification as soon as an instance is chosen for termination, but there is the possibility that the Spot Instance will be terminated before the warning arrives. Your game server should be prepared to handle unexpected interruptions.

6. **Evaluate the performance of your Spot fleets and queues.** Once the new queue is actively placing new game sessions, use GameLift metrics to evaluate performance. Key metrics include:

   - ** Interruption rate** – Track the number and frequency of Spot-related interruptions for instances and game sessions in a Spot fleet. These fleet metrics (`InstanceInterruptions` and `GameSessionInterruptions`) can be viewed in the GameLift console or by using Amazon CloudWatch (see GameLift metrics for fleets (p. 171)). If you aggregate fleet metrics in a metrics group, you can also view interruptions by instance type and operating system in CloudWatch. Game sessions that were terminated for Spot-related reasons have a status of "TERMINATED" and a status reason of "INTERRUPTED".

   - **Queue effectiveness** – Track metrics for your queues, including placement success rates, average wait time, and queue depth to verify that the use of Spot fleets has no impact on queue performance. Queue metrics can be viewed in the GameLift console or by using Amazon CloudWatch.

   - **Fleet utilization** – Monitor usage rates for your fleets, including data on instances, game sessions and player sessions. Usage for your On-Demand fleets can be an indicator that FleetIQ is choosing to avoid risky placements into your Spot fleets and falling back to the On-Demand fleets. Fleet utilization metrics can be viewed in the GameLift console or by using Amazon CloudWatch.
Create a new GameLift fleet

Create a new fleet and deploy your custom game server build or Realtime Servers for hosting. You can deploy any game build or script resource that has been successfully uploaded to the GameLift service.

Tip
Learn more about ways to explore Amazon GameLift features, including Realtime Servers, using sample games (p. 27).

Topics
• Deploy a GameLift fleet with a custom game build (p. 112)
• Deploy a Realtime Servers Fleet (p. 116)
• How GameLift fleet creation works (p. 119)

Deploy a GameLift fleet with a custom game build

If you're using Realtime Servers for your game, see Deploy a Realtime Servers Fleet (p. 116).

You can create and deploy a new fleet to host game servers for any game build that has been uploaded to the GameLift service and is in a Ready status.

To create a fleet with a custom game build

Use either the GameLift console or the AWS Command Line Interface (CLI) to create a fleet.

After you create a new fleet, the fleet's status passes through several stages as the fleet is deployed and game servers installed and started up. Once the fleet reaches ACTIVE status, it is ready to host game sessions. For help with fleet creation issues, see Debug GameLift fleet issues (p. 124).

Console

2. On the Builds page, find the build that you want to create a fleet for and verify that its status is Ready. Select the build (use the option button to the left of the build status) and click Create fleet from build.
3. On the Create fleet page, enter the Fleet Details:
   • 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 Choosing computing resources (p. 107).
   • Metric group – (Optional) Enter the name of a new or existing fleet metric group. When using Amazon CloudWatch to track your GameLift metrics, you can aggregate the metrics for multiple fleets by adding them to the same metric group.
   • Instance role ARN – (Optional) Enter the ARN value for an IAM role that you want to associated with this fleet. This setting allows all instances in the fleet to assume the role, which extends access to a defined set of AWS services. Learn more about how to Communicate with other AWS resources from your fleets (p. 55). When creating a fleet with an instance role ARN, you must have IAM PassRole permission, as described in IAM policy examples for GameLift (p. 18).
   • Certificate type – Choose whether to have GameLift generate a TLS certificate for the fleet. You can use a fleet TLS certificate to have your game client authenticate a game server when connecting, and encrypt all client/server communication. For each instance in a TLS-enabled fleet, GameLift also creates a new DNS entry with the certificate. Use these resources to set
Create a new fleet

1. Set up authentication and encryption for your game. Once the fleet is created, you cannot change the certificate type.
   - **Binary type** – Select the binary type "Build".
   - **Build** – If you used the Create fleet from build feature, the build information, including name, ID and operating system, is automatically filled in. Otherwise, select a valid build from the dropdown list.
   - **Home region** – Indicates the Region where the fleet is being created. It is also where the selected build is located. You can change the home Region by selecting a different Region in the Console's header. Fleet instances are deployed in the fleet's home Region.

2. **Instance type**. Select an Amazon EC2 instance type from the list. The instance types listed vary depending on 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 Choosing computing resources (p. 107). Once the fleet is created, you cannot change the instance type.

3. **Location management**. Select one or more additional remote locations to deploy instances to. This option is available when creating a fleet in an AWS Region that supports multi-location fleets (see Using Amazon GameLift in AWS Regions (p. 25)). The fleet deploys instances to its home Region, which is the Region where the fleet is being created. If you select additional locations, fleet instances are also deployed in these locations. Locations are disabled if the selected instance type is not available in that location.

4. **Process management**. Configure how you want server processes to run on each instance.
   - **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 your game build has multiple server executables, 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 the GameLift 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 cannot activate.

The collection of server process configurations is called the fleet's runtime configuration. It describes all 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 GameLift allows for a new game session activate. If the game session does not complete activation and move to status ACTIVE, the game session activation is terminated.

7. **EC2 port settings.** 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.)

8. 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.

9. Once you’ve finished setting the configuration for your new fleet, click **Initialize fleet**. GameLift assigns an ID to the new fleet and begins the fleet activation process. You can track the new fleet’s status on the **Fleets** page.

You can update the fleet’s metadata and configuration at any time, regardless of fleet status (see **Manage your GameLift fleets**). You can update fleet capacity only once the fleet has reached ACTIVE status (see **Scaling GameLift hosting capacity**). You can also add or remove remote locations.

**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.** Get and install the AWS Command Line Interface tool.

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

- The fleet will use c5.large On-Demand Instances with the operating system that is appropriate for the selected game build.
- It will deploy the specified game server build, which must be in a **Ready** status to the following locations:
  - us-west-2 (home Region)
  - sa-east-1 (remote location)
- TLS certificate generation is enabled.
- 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.
Create a new fleet

- On each instance, 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 EMEAfleets metric group, which (in this example) combines metrics for all fleets in EMEA regions.

```
# AWS gamelift create-fleet
   --name "SampleFleet123"
   --description "The sample test fleet"
   --ec2-instance-type "c5.large"
   --region us-west-2
   --locations "Location=sa-east-1"
   --fleet-type "ON_DEMAND"
   --build-id "build-92f061ed-27c9-4a02-b1f4-6f85b2385620"
   --certificate-configuration "CertificateType=GENERATED"
   --runtime-configuration "GameSessionActivationTimeoutSeconds=300,"
   MaxConcurrentGameSessionActivations=2,
   ServerProcesses=[{LaunchPath=C:\game\Bin64.dedicated\MultiplayerSampleProjectLauncher_Server.exe,Parameters=+sv_port 33435 +start_lobby,
   ConcurrentExecutions=10}]
   --new-game-session-protection-policy "FullProtection"
   --resource-creation-limit-policy "NewGameSessionsPerCreator=3,"
   PolicyPeriodInMinutes=15"
   --ec2-inbound-permissions
   "FromPort=33435,ToPort=33435,IpRange=0.0.0.0/0,Protocol=UDP"
   "FromPort=33235,ToPort=33235,IpRange=0.0.0.0/0,Protocol=UDP"
   --metric-groups "EMEAfleets"
```

Copiable version:

```
AWS gamelift create-fleet --name "SampleFleet123" --description "The sample test fleet"
   --ec2-instance-type "c5.large" --region us-west-2 --locations "Location=sa-east-1"
   --fleet-type "ON_DEMAND" --build-id "build-92f061ed-27c9-4a02-b1f4-6f85b2385620"
   --certificate-configuration "CertificateType=GENERATED" --runtime-configuration
   "GameSessionActivationTimeoutSeconds=300, MaxConcurrentGameSessionActivations=2,"
   ServerProcesses=[{LaunchPath=C:\game\Bin64.dedicated\MultiplayerSampleProjectLauncher_Server.exe,Parameters=+sv_port 33435 +start_lobby,
   ConcurrentExecutions=10}]
   --new-game-session-protection-policy "FullProtection"
   --resource-creation-limit-policy "NewGameSessionsPerCreator=3,"
   PolicyPeriodInMinutes=15"
   --ec2-inbound-permissions
   "FromPort=33435,ToPort=33435,IpRange=0.0.0.0/0,Protocol=UDP"
   "FromPort=33235,ToPort=33235,IpRange=0.0.0.0/0,Protocol=UDP"
   --metric-groups "EMEAfleets"
```

If the create-fleet request is successful, GameLift returns a set of fleet attributes that includes the configuration settings you requested and a new fleet ID. GameLift immediately initiates the fleet activation process and sets the fleet status and the location statuses to **New**. You can track the fleet's status and view other fleet information using these CLI commands:

- `describe-fleet-events`
- `describe-fleet-attributes`
- `describe-fleet-capacity`
Create a new fleet

- describe-fleet-port-settings
- describe-fleet-utilization
- describe-runtime-configuration
- describe-fleet-location-attributes
- describe-fleet-location-capacity
- describe-fleet-location-utilization

You can change the fleet's capacity and other configuration settings as needed using these commands:

- update-fleet-attributes
- update-fleet-capacity
- update-fleet-port-settings
- update-runtime-configuration
- create-fleet-locations
- delete-fleet-locations

Deploy a Realtime Servers Fleet

You can create a new fleet of Realtime game servers to host game sessions for your game. Realtime Servers fleets require that you create a Realtime script and upload it to Amazon GameLift. If you have a custom game server build, see Deploy a GameLift fleet with a custom game build (p. 112) for help creating a fleet with it. 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. 120).

Create a Realtime Fleet (Console)

To create a Realtime fleet with the Amazon GameLift console:

1. Open the Amazon GameLift console at https://console.aws.amazon.com/gamelift/. Go to the Fleets: Create fleet page to configure a new fleet.

2. Fleet Details.

   - **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 Choosing computing resources (p. 107).
   - **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.
   - **Instance role ARN** – (Optional) Enter the ARN value for an IAM role that you want to associated with this fleet. This setting allows all instances in the fleet to assume the role, which extends access to a defined set of AWS services. Learn more about how to Communicate with other AWS resources from your fleets (p. 55). When creating a fleet with an instance role ARN, you must have IAM PassRole permission, as described in IAM policy examples for GameLift (p. 18).
   - **Certificate type** – Choose whether to have GameLift generate a TLS certificate for the fleet. With this feature enabled for a Realtime fleet, GameLift automatically authenticates the client/server connection and encrypts all communication between game client and server. Once the fleet is created, you cannot change the certificate type.
   - **Binary type** – Select the binary type "Script".
   - **Script** – Select the Realtime script you want to deploy from the dropdown list.
• **Home region** – Indicates the Region where the fleet is being created. It is also where the selected build is located. You can change the home Region by selecting a different Region in the Console’s header. Fleet instances are deployed in the fleet's home Region.

3. **Instance type.** 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 *Choosing computing resources (p. 107).* Once the fleet is created, you cannot change the instance type.

4. **Location management.** Select one or more additional remote locations to deploy instances to. This option is available when creating a fleet in an AWS Region that supports multi-location fleets (see *Using Amazon GameLift in AWS Regions (p. 25).* The fleet deploys instances to its home Region, which is the Region where the fleet is being created. If you select additional locations, fleet instances are also deployed in these locations. Locations are disabled if the selected instance type is not available in that location.

5. **Process management.** 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 want to launch processes using different files in your uploaded Realtime script, you must have a configuration for each type of process you want to launch.

   • **Launch path** – Type the name of the script file that you want to launch with. A launch script file must call an *Init()* function. During deployment, your uploaded Realtime script files are unzipped and stored in the /local/game/ directory, so you just need to specify the script file name. Example: *MyRealtimeLaunchScript.js*.

   • **Launch parameters** – (Optional) You can pass information to your Realtime script at launch time. Type the information as a set of command line parameters. Example: *+map Winter444*.

   • **Concurrent processes** – Indicate how many server processes with this configuration to run concurrently on each instance in the fleet.

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

   Check the Amazon GameLift limits on the number of concurrent server processes. Limits on concurrent server processes per instance apply to the total of concurrent processes set for all configurations. For example, if you’re limited to one process, you can have only one configuration, and concurrent processes must be set to 1. If a fleet is configured to exceed the limit, the fleet cannot activate.

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

   b. **Game session activation (optional):**

   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 to activate. If the game session does not move to status ACTIVE, the game session activation process is terminated.

6. **Protection policy (optional)**. Indicate whether or not to apply game session protection to instances in this fleet. Protected instances are not terminated during a scale-down event if they are hosting an active game session. Using this setting applies a fleet-wide protection policy; you can also set protection for individual game sessions when creating the game session.

7. Once you’ve finished configuring the 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. 131), runtime configuration, and other configuration settings as needed. You can also add or remove remote locations.

### Create a Realtime Fleet (AWS CLI)

To create a Realtime 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. Get and install the AWS Command Line Interface tool.

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

- The fleet will use c5.large spot instances.
- It will deploy the specified Realtime script to the following locations: `us-west-2` (home Region) and `sa-east-1` (remote location).
- Each instance in the fleet will run ten identical processes of the Realtime script 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 60 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.
- Metrics for this fleet will be added to the EMEAfleets metric group, which (in this example) combines metrics for all fleets in EMEA regions.

**Note**

For Realtime Servers fleets, Amazon GameLift automatically sets TCP and UDP ranges for use by the Realtime servers. You can view the automatic settings by calling the CLI command `describe-fleet-port-settings`.

```bash
$ AWS gamelift create-fleet
   --name "SampleRealtimeFleet123"
   --description "A sample Realtime fleet"
   --ec2-instance-type "c5.large"
   --region us-west-2
   --locations "Location=sa-east-1"
   --fleet-type "SPOT"
   --script-id "script-1111aaaa-22bb-33cc-44dd-5555eeee66ff"
   --certificate-configuration "CertificateType=GENERATED"
   --runtime-configuration "GameSessionActivationTimeoutSeconds=60,
   MaxConcurrentGameSessionActivations=2,
   ServerProcesses=[{LaunchPath=/local/game/
   myRealtimeLaunchScript.js,
   Parameters=+map Winter444,
   ConcurrentExecutions=10}]"
```

Version
118
Create a new fleet

```
--new-game-session-protection-policy "FullProtection"
--resource-creation-limit-policy "NewGameSessionsPerCreator=3,
PolicyPeriodInMinutes=15"
--metric-groups "EMEAfleets"
```

Copiable version:

```
AWS gamelfit create-fleet --name "SampleRealtimeFleet123" --description "A sample Realtime
fleet" --ec2-instance-type "c5.large" --region us-west-2 --locations "Location=sa-
east-1" --fleet-type "SPOT" --script-id "script-1111aaaa-22bb-33cc-44dd-5555eeee66ff"
--certificate-configuration "CertificateType=GENERATED" --runtime-configuration
"GameSessionActivationTimeoutSeconds=60,MaxConcurrentGameSessionActivations=2,ServerProcesses=[{LaunchPath=
local/game/myRealtimeLaunchScript.js,Parameters=+map Winter444,ConcurrentExecutions=10}]
"--new-game-session-protection-policy "FullProtection" --resource-creation-limit-policy
"NewGameSessionsPerCreator=3,PolicyPeriodInMinutes=15" --metric-groups "EMEAfleets"
```

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 and the location statuses to New. You can track the
fleet's status and view other fleet information using these CLI commands:

AWS CLI

- describe-fleet-events
- describe-fleet-attributes
- describe-fleet-capacity
- describe-fleet-port-settings
- describe-fleet-utilization
- describe-runtime-configuration
- describe-fleet-location-attributes
- describe-fleet-location-capacity
- describe-fleet-location-utilization

You can change the fleet's capacity and other configuration settings as needed using these commands:

- update-fleet-attributes
- update-fleet-capacity
- update-fleet-port-settings
- update-runtime-configuration
- create-fleet-locations
- delete-fleet-locations

How GameLift fleet creation works

When you create a new fleet, the GameLift service initiates a workflow that creates a fleet with one
instance in each fleet location. As each step of the workflow is completed, events are emitted and status
is updated. You can track all events, including those for fleet creation, using the GameLift console (see
the Fleet detail page, Events tab) or by calling the GameLift API DescribeFleetEvents. You can track the
status of individual locations using DescribeFleetLocationAttributes.

**Note**

You cannot remotely access an instance in a fleet until the fleet is in ACTIVE status.
Fleet creation workflow:

- GameLift creates a Fleet resource. For the fleet's home Region and each remote location that is defined in the fleet, the desired capacity is set to one (1) instance. Fleet/location status is set to New, and GameLift begins writing events to the fleet event log.
- GameLift allocates requested computing resources for one new instance in each fleet location.
- GameLift downloads the game server files to each instance. The files are either the custom game build or a Realtime Servers build with custom configuration script. Status is set to Downloading.
- The downloaded game server files on each instance are validated to ensure that no errors occurred during downloading. Status is set to Validating.
- The game server is built on each instance. For a custom game build, the game server is installed using an install script if available. For Realtime Servers, the software is installed using the configuration script. Status is set to Building.
- GameLift begins launching server processes on each instance, following instructions in the fleet's runtime configuration. If the fleet is configured to run multiple concurrent server processes per instance, GameLift staggers the process launches by a few seconds. As each process comes online, it reports readiness back to the GameLift service. Status is set to Activating.
- As soon as one server process in any fleet location notifies GameLift that it is ready, the fleet status and the location status is set to Active. As server processes in other fleet locations report readiness, the status of each fleet location is set to Active.

To troubleshoot issues with fleet creation, see Debug GameLift fleet issues (p. 124).

Manage your GameLift fleets

Use the GameLift console or the AWS CLI to update your fleet settings or delete a fleet. You can update the following fleet settings:

- General fleet attributes. Change fleet settings including name and description, game session protection, resource creation limits, and metric group. These attributes apply to the entire fleet and, where relevant, to instances in all fleet locations.
- Port settings. Add or remove inbound permissions for game servers in the fleet. These settings define access permissions for inbound traffic that connects to game servers on the instances in all fleet locations.
- Runtime configuration. Change the instructions that determine which server processes to run concurrently on an instance. The runtime configuration is used by the instances in all fleet locations. Updates are routinely provided to every instance.
- Remote locations. Add or remove remote locations from a fleet. A fleet's locations determine where fleet instances are deployed to host game sessions.

To update a fleet configuration

You can update mutable fleet attributes, port settings and runtime configurations using the GameLift console or the AWS CLI. To change scaling limits, see Auto-scale fleet capacity with GameLift (p. 132).

**Note**

It is possible for an active fleet to be deployed with a build that is currently in Deleted or Error status. This does not affect the fleet's status or ability to host game sessions.

GameLift Console

1. Open the Amazon GameLift console at https://console.aws.amazon.com/gameli/
2. Choose Fleets from the menu bar to view a list of fleets, and click on the name of the fleet you want to update. A fleet must be in ACTIVE status before it can be edited.

4. On the Edit fleet page, you can make the following updates (see Deploy a GameLift fleet with a custom game build (p. 112) for more detailed field descriptions):
   - Change the fleet attributes such as Name and Description.
   - Add or remove Metric groups, which are used in Amazon CloudWatch to track aggregated GameLift metrics for multiple fleets.
   - Change how you want server processes to run and host game sessions by updating the Server process allocation (runtime configuration) and game session activation settings.
   - Update the EC2 port settings used to connect to server processes on this fleet.
   - Update resource creation limit settings.
   - Turn game session protection on or off.

5. Click Submit to save your changes.

**AWS CLI**

Get and install the AWS Command Line Interface tool.

Use the following AWS CLI commands to update a fleet:

- update-fleet-attributes
- update-fleet-port-settings
- update-runtime-configuration

**To update fleet locations**

You can add or remove a fleet's remote locations using the GameLift console or the AWS CLI. A fleet's home Region, which is where the fleet was created and resides, cannot be changed.

**GameLift Console**

1. Open the Amazon GameLift console at https://console.aws.amazon.com/gamelift/.
2. Choose Fleets from the menu bar to view a list of fleets, and click on the name of the fleet you want to update. A fleet must be in ACTIVE status before it can be edited.
3. On the Fleet detail page, open the Locations tab to view the fleet's locations. This list includes the fleet's home Region (labelled) and all remote locations. This view displays each location's current status, and number of active instances, game servers, and game sessions.
4. To add new remote locations, click Add locations and select the locations you want to deploy instances to. This list does not include instances where the fleet's instance type is not available. You can add some or all available locations to the fleet.
5. With new locations selected, click Save. The new locations are added to the list, with status set to NEW. GameLift immediately begins provisioning an instance in each added location and preparing it to host game sessions. As this process moves forward, the location status changes until it is ACTIVE.
6. To remove existing remote locations from the fleet, use the check boxes to select one or more listed locations.
7. With one or more fleets selected, click Actions > Remove locations. The removed locations remain in the list, with status set to DELETING. GameLift immediately begins the process of terminating activity in the removed location. If there are active instances that are hosting game sessions, GameLift used the game server termination process to gracefully end game sessions, terminate game servers, and shut down instances.
AWS CLI

Get and install the AWS Command Line Interface tool.

Use the following AWS CLI commands to update fleet locations:

- create-fleet-locations
- delete-fleet-locations

To delete a fleet

You can delete a fleet when it is no longer needed. Deleting a fleet permanently removes all data associated with game sessions and player sessions, as well as collected metric data. As an alternative, you can retain the fleet, disable auto-scaling, and manually scale the fleet to 0 instances.

Note
If the fleet being deleted has a VPC peering connection, you first need to request authorization by calling CreateVpcPeeringAuthorization. You do not need to explicitly delete the VPC peering connection—this is done as part of the delete fleet process.

You can use either the Amazon GameLift console or the AWS CLI tool to delete a fleet.

GameLift Console

1. Open the Amazon GameLift console at https://console.aws.amazon.com/gamelift/.
2. Choose Fleets from the menu bar to view a list of fleets, and click on the name of the fleet you want to delete. Only fleets in ACTIVE or ERROR status can be deleted.
3. At the top of the Fleet detail page, under Actions, choose Terminate fleet.
4. In the Terminate fleet dialog box, confirm the deletion by typing the name of the fleet.
5. Click Delete.

AWS CLI

Get and install the AWS Command Line Interface tool.

Use the following AWS CLI command to delete a fleet:

- delete-fleet

Add an alias to a GameLift fleet

A GameLift alias is used to abstract a fleet designation. Fleet designations tell Amazon GameLift where to search for available resources when creating new game sessions for players. By using aliases instead of specific fleet IDs, you can more easily and seamlessly switch player traffic from one fleet to another by changing the alias's target location.

There are two types of routing strategies for aliases:

- **Simple** – A simple alias routes player traffic to a specified fleet ID. You can update the fleet ID for an alias 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 direct players who are using an out-of-date client to a location where they can get an upgrade.
Fleets have a finite lifespan, and there are several reasons why you'll need to switch out fleets during the life of a game. Specifically, you can't update a fleet's game server build or change certain computing resource attributes (instance types, spot/on-demand usage) on an existing fleet. Instead, you need to create new fleets with the changes and then switch players to the new fleets. With aliases, switching fleets has minimal impact on your game and is invisible to players.

Aliases are primarily useful in games that do not use queues. Switching fleets in a queue is a simple matter of creating a new fleet, adding it to the queue, and removing the old fleet, none of which is visible to players. In contrast, game clients that don't use queues must specify which fleet to use when communicating with the GameLift service. Without aliases, a fleet switch requires updates to your game code and possibly the need to distribute updated game clients to players. With aliases, you can avoid both.

When updating the fleet-id an alias points to, there is a transition period of up to 2 min during which game sessions being placed on the alias may still end up on the old fleet.

To create a new alias

You can create an alias using either the GameLift console, as described here, or with the AWS CLI command `create-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.

To edit an alias

You can edit an alias using either the GameLift console, as described here, or with the AWS CLI command `update-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.
Debug GameLift fleet issues

This topic provides guidance on fleet configuration issues for a GameLift managed hosting solution. For additional troubleshooting, you can remotely access a fleet instance once the fleet is active. See Remotely access GameLift fleet instances (p. 126).

Fleet creation issues

When a fleet is created, the GameLift service initiates a workflow that deploys a new instance in each of the fleet's locations and prepares it to run your game servers. For a detailed description, see How GameLift fleet creation works (p. 119). A fleet cannot host game sessions and players until it reaches Active status. This section discusses the most common issues that prevent fleets from becoming active.

Downloading and validating

During this phase, fleet creation may fail if there are issues with the extracted build files, the installation script won't run, or if the executable(s) designated in the runtime configuration is not included in the build files. GameLift provides logs related to each of these issues.

If the logs do not reveal an issue, it's possible that the problem is due to an internal service error. In this case, try to create the fleet again. If the problem persists, consider re-uploading the game build (in case the files were corrupted). You can also contact GameLift support or post a question on the forum.

Building

Issues that cause failure during the build phase are almost certainly due to problems with the game build files and/or the installation script. Verify that your game build files, as uploaded to GameLift, can be installed on a machine running the appropriate operating system. Be sure to use a clean OS installation, not an existing development environment.

Activating

The most common fleet creation problems occur during the Activating phase. During this phase, a number of elements are being tested, including the game server's viability, the runtime configuration settings, and the game server's ability to interact with the GameLift service using the Server SDK. Common issues that come up during fleet activation include:

Server processes fail to start.

First check that you've correctly set the launch path and optional launch parameters in the fleet's runtime configuration. You can view the fleet's current runtime configuration using either the GameLift console (see the Fleet detail page, Capacity allocation (p. 166)) tab or by calling the AWS CLI command describe-runtime-configuration. If the runtime configuration looks correct, check for issues with your game build files and/or installation script.

Server processes start but fleet fails to activate.

If server processes start and run successfully, but the fleet does not move to Active status, a likely cause is that the server process is failing to notify GameLift that it is ready to host game sessions. Check that your game server is correctly calling the Server API action ProcessReady() (see Initialize the server process (p. 52)).

VPC peering connection request failed.

For fleets that are created with a VPC peering connection (see To set up VPC peering with a new fleet (p. 157)), VPC peering is done during this Activating phases. If a VPC peering fails for any reason, the new fleet will fail to move to Active status. You can track the success or failure of the peering request by calling describe-vpc-peering-connections. Be sure to check that a valid VPC peering authorization exists (describe-vpc-peering-authorizations, since authorizations are only valid for 24 hours.)
Server process issues

Server processes start but fail quickly or report poor health.

Other than issues with your game build, this outcome can happen when trying to run too many server processes simultaneously on the instance. The optimum number of concurrent processes depends on both the instance type and your game server's resource requirements. Try reducing the number of concurrent processes, which is set in the fleet's runtime configuration, to see if performance improves. You can change a fleet's runtime configuration using either the GameLift console (edit the fleet's capacity allocation settings) or by calling the AWS CLI command update-runtime-configuration.

Fleet deletion issues

Fleet can't be terminated due to max instance count.

The error message indicates that the fleet being deleted still has active instances, which is not allowed. You must first scale a fleet down to zero active instances. This is done by manually setting the fleet's desired instance count to "0" and then waiting for the scale-down to take effect. Be sure to turn off auto-scaling, which will counteract manual settings.

VPC actions are not authorized.

This issue only applies to fleets that you have specifically created VPC peering connections for (see VPC peering for GameLift (p. 156). This scenario occurs because the process of deleting a fleet also includes deleting the fleet's VPC and any VPC peering connections. You must first get an authorization by calling the GameLift service API CreateVpcPeeringAuthorization() or use the AWS CLI command create-vpc-peering-authorization. Once you have the authorization, you can delete the fleet.

Realtime Servers fleet issues

Zombie game sessions: They start and run a game, but they never end.

You might observe this issues as any of the following scenarios:

- Script updates are not picked up by the fleet's Realtime servers.
- The fleet quickly reaches maximum capacity and does not scale down when player activity (such as new game session requests) decreases.

This is almost certainly a result of failing to successfully call processEnding in your Realtime script. Although the fleet goes active and game sessions are started, there is no method for stopping them. As a result, the Realtime server that is running the game session is never freed up to start a new one, and new game sessions can only start when new Realtime servers are spun up. In addition, updates to the Realtime script do not impact already-running game sessions, only ones.

To prevent this from happening, scripts need to provide a mechanism to trigger a processEnding call. As illustrated in the Realtime Servers Script Example (p. 75), one way is to program an idle session timeout where, if no player is connected for a certain amount of time, the script will end the current game session.

However, if you do fall into this scenario, there are a couple workarounds to get your Realtime servers unstuck. The trick is to trigger the Realtime server processes—or the underlying fleet instances—to restart. In this event, GameLift automatically closes the game sessions for you. Once Realtime servers are freed up, they can start new game sessions using the latest version of the Realtime script.

There are a couple of methods to achieve this, depending on how pervasive the problem is:
Remotely access fleet instances

- Scale the entire fleet down. This method is the simplest to do but has a widespread effect. Scale the fleet down to zero instances, wait for the fleet to fully scale down, and then scale it back up. This will wipe out all existing game sessions, and let you start fresh with the most recently updated Realtime script.
- Remotely access the instance and restart the process. This is a good option if you have only a few processes to fix. If you are already logged onto the instance, such as to tail logs or debug, then this may be the quickest method. See Remotely access GameLift fleet instances (p. 126).

If you opt not to include way to call `processEnding` in your Realtime script, there are a couple of tricky situations that might occur even when the fleet goes active and game sessions are started. First, a running game session does not end. As a result, the server process that is running that game session is never free to start a new game session. Second, the Realtime server does not pick up any script updates.

**Remotely access GameLift fleet instances**

You can remotely access any instance that is currently running in your GameLift fleets. Some common reasons to directly access an instance include:

- To troubleshoot issues with:
  - how game server processes are started and stopped based on your runtime configuration.
  - how your game server interacts with the GameLift service.
  - game session and player connection issues.
  - matchmaking backfill issues.
- To get real-time game server activity, such as to track log updates.
- To run benchmarking tools using actual player traffic.
- To investigate specific issues with a game session or game server process.

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

- Only instances in active fleets can be remotely accessed. If your fleet fails to activate, then there is no way to get instance connection information. For help with fleet activation issues, see Debug GameLift fleet issues (p. 124).
- Remotely connecting to an instance doesn't affect instance activity. The instance continues to start and stop server processes, host game sessions when placed on the instance by GameLift, and can be terminated at any time in response to a scale down event or a Spot interruption.
- Any changes you make to an instance can potentially impact the instance's active game sessions and connected players.

**Connecting 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. Get and install the AWS Command Line Interface tool. You can also use the AWS SDK, with documentation available in the GameLift Service API Reference.

1. Open
2. **Find the ID of the instance you want to connect to.** When requesting access, you must specify an instance ID. Use the AWS CLI command `describe-instances` (or the API call `DescribeInstances`) with a fleet ID to get information on all instances in the fleet. For help, including example requests and responses, see the CLI or API reference guides.
3. **Request access credentials for the instance.** Once you have an instance ID, use the command `get-instance-access` (or the API call `GetInstanceAccess`) to request access credentials and other information. For help, including example requests and responses, see the CLI or API reference guides. If successful, 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 values as simple strings, so you can use the returned values as is. Example credentials:

  ```json
  "Credentials": {
  "Secret": "aA1bBB2cCcd3EEE",
  "UserName": "gl-user-remote"
  }
  ```

- **For Linux instances** — To connect to a Linux instance, SSH requires a user name and private key. GameLift issues RSA private keys and returns them as a single string, with the newline character (`
`) 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. Example credentials returned:

  ```json
  "Credentials": {
  "Secret": "-----BEGIN RSA PRIVATE KEY-----
  nEXAMPLEKEYCQAgEAYwWzhaDsrA1w3mRlQtvhvGRX8gnxDaFrt/gx42kWxTs4rXe/xe/b5CpSgle/
  \nvBoU7jLxxz92pHoFn8yP+Dc21eyz6cvjTMwaJwEw5/aK7i0s5Dr6C7QKw2duV5quDeOQW
  \n/aKnxMi1Q66XagnfNjXerqZ+CwqeqUwMkUeLeJFhlyMCyUrPUNMc1Eeh4911l9X1F
  \nG50TCEF0Feb18dd0qC62zPa1j1i19X/az09VR+tp0uZEL+wmnXn13/tNaPB5xvd02U767km6SUpW
  \n0Pzer/0L8V+x+bi4hThfSj9Y7aVQjF1FvWHX1g4bt2C02/wei18/RtIIDAaAbo1BA2GZ1kaEvnrqu
  \n/u/le179v7mSm7iLS5lWt48dJLAW68ut/fztvCH0SkkBoCQXrlNhMq2VgqY3X/k0n2WfYJ5V
  ufscbl11nmb5qwMG1nEpJaZD6QSS3kCI1wAXNYU1GcU0iShbJooat/
  GTLJU05Kfcv30pABUyN5p53v6G7Hxb2\nbabyWyJnfjeL4Me846yd2Y3V2Cm^X+/X/
  BO85hnJj6+hjXPFmpV39NZxEmCdj1a/X15Dymhm/
  t0w5DS/\n810Gk9TopEp7CkIfatEATyy2iVqoQg6k64iuM9JKaA3os2xzQmXeqVXUY1TLZVEH07bh1y9d01ozR
  \nQa/Gs/FIZaXa2ixjWyv0lpjE7s+kCgYEAa9mZtyhhhKkFDpwrSMAPaL80NAbbjWeyJ7ZSMqfl
  +l1p+lYKrLidDzLx1rVAH+yHPR1t2HJOt7U2h4aXv+cpg09quU7B34eEy24B7G/U+GTfbXSxOqX/
  \n+p90lyVwcT7asQ35A8ZHi+mvkJ5OBEKzet9XcWONBYELghnEpe7CGyEAOY6voGye6YhLeHui19khwhu
  \nyavu0 kelc5xkJ9fnHfHR2y11r1r2w2wqpn+948lURpzwVW9Ehv+m=xtfma21sPj/1k75XwnO
  \nAWk9gkmo60G3qJ3y2NBURa3f335LR0i0qGvQe10HLYXpJneEHhv+unzaLzl1W0tS5bBkK0UC
  \nynbT1b0+0SdxGcpf2nbbhj1p1FdBeryf1YRX5gsV2UwNQAwjdp9PEn9y9pQ+8xMNkY1ycOGQWqWbOh
  \nNoQ0yvVyW05DswQa4Qd1Wx24nGd3tIFxWI15eKuAaeOChboy18w8f8wXj57t1sdoXNe0a
  \nArq6Wt/GlC6qpnA92xK9vkVwbKgB1F1O1vlJWbirsGZg9wvKwFpFtRdK3v7ZyY9z0xqZj5SnXkh
  \nNh0v1Hdx4ZctmCqBpIAyMnjr/T0blybxAmN410aXNMMg4K5y11mSzaQoFq+eC3sd4jPf1lj
  \nNjboED/BNy8frlNDAVHNE85SDs2F26EL8yE9RJSrR9xNaAGMwTvYNe4zvzks/F5yU01ociDLoa
  \nNnWU38/vnDcGpIXD5h3q3aEcjuiJmambilwTv+iYzj5hv7U9G8mJwUNGT1d6bns6ysQeMa2srF3Qs
  \nNvRkAKKkY3yj4jgkU4FtYTW0FYJXk6cFrR/CVQF503nHAAJXJw4e4ej3JncTzm2SpYzwApc\\n-----END RSA PRIVATE KEY------",
  "UserName": "gl-user-remote"
  }
  ```

When using the AWS CLI, you can automatically generate a properly formatted .pem file by including the --query and --output parameters to your `get-instance-access` request.

To set permissions on the .pem file, run the following command:

```
$ chmod 400 MyPrivateKey.pem
```

4. **Open a port for the remote connection.** Instances in 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`. Version 127
As a best practice, we recommend opening ports for remote access only when you need them and closing them when you're finished. For example, once a fleet is created but before being activated, its port settings cannot be updated. If you are stuck, re-create the fleet with the port settings open.

Use the command `update-fleet-port-settings` to add a port setting for the remote connection (such as 22 for SSH or 3389 for RDP). For the IP range value, specify the IP addresses for the devices you plan to use to connect (converted to CIDR format). Example:

```bash
$ AWS gamelift update-fleet-port-settings
  --fleet-id  "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa"
  --inbound-permission-authorizations
  "FromPort=22,ToPort=22,IpRange=54.186.139.221/32,Protocol=TCP"
```

The following example opens up port 3389 on a Windows fleet

```bash
$ AWS gamelift update-fleet-port-settings
  --fleet-id  "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa"
  --inbound-permission-authorizations
  "FromPort=3389,ToPort=3389,IpRange=54.186.139.221/32,Protocol=TCP"
```

5. **Open a remote connection client.** Use Remote Desktop for Windows or SSH for Linux instances. Connect to the instance using the IP address, port setting, and access credentials.

   SSH example:

   ```bash
   ssh -i MyPrivateKey.pem gl-user-remote@192.0.2.0
   ```

### Viewing files on 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 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.

- **GameLift hosting resources.** Files used by the 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`).

- **Fleet data.** Attributes of the fleet that the instance belongs to is stored in a JSON file. This information can be used by server processes that are running on instance. The JSON file is stored in the following location:
  - On Windows: `C:\GameMetadata\gamelift-metadata.json`
Scaling GameLift hosting capacity

Hosting capacity, which is measured in instances, represents the number of game sessions that can be hosted concurrently, and by extension the number of concurrent players that those game sessions can accommodate. One of the most challenging tasks with game hosting is to scale capacity so that it meets player demand without wasting money on resources that aren't needed. Learn more about how capacity scaling works (p. 6) in GameLift.

You have full control to manage the hosting capacity of your fleets. Capacity is adjusted at the fleet location level. All fleets have at least one location: the fleet's home Region. When viewing or scaling capacity, the information is listed by location, including the fleet's home Region and any additional remote locations.

You can manually set the number of instances to maintain, or you can set up auto-scaling to dynamically adjust capacity as player demand changes. We recommend that you start by turning on the Target Tracking auto-scaling option. The idea behind target tracking is to maintain just enough hosting resources to accommodate current players plus a little extra to handle unexpected spikes in player demand. For most games, target tracking offers a highly effective scaling solution.

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

- Set minimum and maximum limits for capacity scaling (p. 130)
- Manually set capacity levels (p. 131)
- Turn on auto-scaling with Target Tracking (p. 133)
- Manage rule-based auto-scaling (advanced feature) (p. 134)
- Temporarily disable auto-scaling (p. 131)

Most fleet scaling activities can be done using the GameLift console. You can also use the AWS SDK or AWS CLI with the GameLift Service API.

To manage fleet capacity in the 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. In this tab, you can:
   - View historical scaling metrics for the entire fleet or by individual fleet location.
   - View and update capacity settings for each fleet location, including scaling limits and current capacity settings.
   - Update target tracking auto-scaling, view rule-based auto-scaling policies that are applied to the entire fleet, and enable/disable auto-scaling activity for each location.

Topics

- Set GameLift capacity limits (p. 130)
• Manually set capacity for a GameLift fleet (p. 131)
• Auto-scale fleet capacity with GameLift (p. 132)

Set GameLift capacity limits

When scaling hosting capacity for a fleet location, either manually or by auto-scaling, you must consider the location's scaling limits. All fleet locations have a minimum and maximum limit that define the allowed range for the location's capacity. By default, limits on fleet locations are set to a minimum of 0 instances and a maximum of 1 instance. Before you can scale a fleet location, you'll need to adjust the limits.

If you're using auto-scaling, the maximum limit allows GameLift scale up a fleet location to meet player demand but prevents 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 to maintain some hosting availability at all times, even when player demand is low.

You can set capacity limits for a fleet's locations in the GameLift console or by using the AWS CLI. The location'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. Capacity scaling settings are located in the Scaling Limits section.
4. For each fleet location, set minimum and maximum instance counts. After making a change, you must commit your changes by clicking the check mark button.

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

The new limits are immediately reflected in the Scaling History graph.

AWS CLI

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

```
AWS gamelift describe-fleet-location-capacity --fleet-id <fleet identifier> --location <location name>
```

2. **Update limit settings.** In a command line window, use the update-fleet-capacity command with the following parameters. You can adjust both instance limits and desired instance count with the same command.

```
--fleet-id <fleet identifier>
--location <location name>
```
Manually set fleet capacity

When you create a new fleet, desired capacity is automatically set to one instance in each fleet location. In response, GameLift deploys one new instance in each location. To change fleet capacity, you can either turn on auto-scaling or you can manually set the number of instances you want for a location. Learn more about Scaling fleet capacity (p. 6).

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 level. When setting a desired capacity manually, keep in mind that this action will affect actual fleet capacity only as long as there are no auto-scaling policies in force for the fleet location. If auto-scaling is enabled, it will immediately reset the desired capacity based on its own scaling rules.

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

Disabling auto-scaling

When a fleet is configured with auto-scaling policies, you can opt to turn off all auto-scaling activity for each fleet location. With auto-scaling turned off, the desired number of instances in the fleet location will remain the same unless manually changed. When auto-scaling is turned off for a location, it affects the fleet's current policies and any policies that might be defined in the future.

To manually set fleet capacity

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 and scroll down to the Auto-Scaling Policies section. If there are auto-scaling policies defined for the fleet, this section lists each fleet location with a Disable all scaling policies check box. Disable auto-scaling for the location that you want to set manually.
4. In the **Scaling Limits** section, set the Desired instances field for the location you want to set manually. This value tells 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 falls outside of the location's minimum and maximum limits, an error is generated.

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. 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 in the location changes to match the updated desired instances value. This process may take a little time.

**AWS CLI**

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

   ```bash
   AWS gamelift describe-fleet-location-capacity --fleet-id <fleet identifier> --location <location name>
   ```

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

   ```bash
   --fleet-id <fleet identifier>  
   --location <location name>  
   --desired-instances <fleet capacity as an integer>  
   --max-size <maximum capacity>  [Optional]  
   --min-size <minimum capacity>  [Optional]
   ```

   **Example:**

   ```bash
   AWS gamelift update-fleet-capacity  
   --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa  
   --location us-west-2  
   --desired-instances 5  
   --max-size 10  
   --min-size 1
   ```

   **Copyable version:**

   ```bash
   AWS gamelift update-fleet-capacity --fleet-id  
   fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa --location us-west-2 --desired-instances 5 --max-size 10 --min-size 1
   ```

   If your request is successful, the fleet ID is returned. If the new **desired-instances** setting is outside the minimum/maximum limits, an error is returned.

### Auto-scale fleet capacity with GameLift

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. 6) works in GameLift.

Auto-scaling is done by creating scaling policies that provide instructions to 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 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. 133)
- Auto-scale with rule-based policies (p. 134)

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. 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.

To 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.

**AWS CLI**

1. **Set capacity limits.** Set either or both limit values using the `update-fleet-capacity` command. For help, see Set GameLift capacity limits (p. 130).

2. **Create a new policy.** Open a command-line window and use the `put-scaling-policy` command with your policy's parameter settings. To update an existing policy, specify the policy's name and provide a complete version of the updated policy.

```
--fleet-id <unique fleet identifier>
--name "<unique policy name>"
--policy-type <target- or rule-based policy>
--metric-name <name of metric>
--target-configuration <buffer size>
```

Example:

```
$AWS gamelift put-scaling-policy
--fleet-id "fleet-2222babb-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-2222babb-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 GameLift Service API. You can view all active policies in the 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 Set GameLift capacity limits (p. 130).

2. **Create a new policy.** Open a command-line window and use the `put-scaling-policy` command with your policy's parameter settings. To update an existing policy, specify the policy's name and provide a complete version of the updated policy.

   ```
   --fleet-id <unique fleet identifier>
   --name "<unique policy name>"
   --policy-type <target- or rule-based policy>
   --metric-name <name of metric>
   --comparison-operator <comparison operator>
   --threshold <threshold integer value>
   --evaluation-periods <number of minutes>
   --scaling-adjustment-type <adjustment type>
   --scaling-adjustment <adjustment amount>
   ```

   **Example:**

   ```
   AWS gamelift put-scaling-policy
   --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa
   --name "Scale up when AGS<50"
   --policy-type RuleBased
   --metric-name AvailableGameSessions
   --comparison-operator LessThanThreshold
   --threshold 50
   --evaluation-periods 10
   --scaling-adjustment-type ChangeInCapacity
   --scaling-adjustment 1
   ```

   **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:**

```
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 10 minutes, then change fleet capacity using `ChangeInCapacity` by 1 instances.

Metric name

To trigger a scaling event, link an auto-scaling policy to one of the following fleet-specific metrics. See GameLift metrics for fleets (p. 171) 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 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 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, GameLift waits ten minutes before responding to triggers from the same policy. This wait allows GameLift time to add the new instances, launch the game servers, connect players, and start collecting data from the new instances. During this time, 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.

**Setting up GameLift queues for game session placement**

The game session queue is the primary mechanism for processing new game session requests and locating available game servers to host them. Although it is possible to request a new game session be
hosted on specific fleet or location, game session placement with queues offer significant benefits for game developers and players. These include:

- **Take advantage of GameLift FleetIQ to find the "best possible" placement.** When processing game session placement requests, a queue uses FleetIQ algorithms to prioritize queue locations based on a set of defined preferences. You can configure a queue to prioritize placement based on some combination of lowest player latency, lowest hosting cost, fleet characteristics, and locations.

- **Host games on lower-priced Spot fleets.** Use queues to optimize use of AWS Spot fleets, which offer significantly lower hosting costs. By default, FleetIQ always tries to place new game sessions in Spot fleets.

- **Place new games faster during high demand.** Queues are configured with multiple possible locations for placements. This means that there is always fallback capacity if the preferred placement location is unavailable. In contrast, when placing game sessions on a single fleet location, it is possible for the location to fill up. Even with auto-scaling, it takes time to spin up new instances when needed. As a result, with queues, player wait time is less likely to be impacted during surges in player demand.

- **Make game availability more resilient.** Outages can happen. With a multi-Region queue, a slowdown or outage doesn't have to affect player access to your game. Instead, if one or more preferred locations are unavailable, GameLift can place new game sessions with the next best choice. Auto-scaling can adjust to 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 placing game sessions on a single fleet location, you need to maintain a higher level of idle capacity just in case. In contrast, the fleet locations in a queue provide back-up capacity for each other. Locations scale up or down based on player demand. For example, when demand is high in Asia, demand is typically low in Europe; your European fleets can provide fallback capacity to support surges in Asia, even when they're scaled down during low demand.

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

Use the AWS Command Line Interface (CLI) or the Amazon GameLift console to create and manage game session queues, and view queue metrics.

### Best practices for GameLift game session queues

Here are some best practices that can help you build effective game session queues for game session placement.

#### Best practices for queues with any fleet type

A queue contains a list of fleet destinations where new game sessions can be placed. Each fleet can have instances deployed in multiple geographic locations. When choosing a placement, the queue selects a combination of a fleet and a fleet location. You provide a set of priorities for the queue to use when choosing a placement.

Consider the following guidelines and best practices:

- You can add fleets and aliases that reside in any AWS Region. Whether you're using fleets with a single location or multiple locations, make sure you add fleets that cover every geographical location where you want to support players. This is particularly important if you're making placements based on reported player latency.

- Assign an alias to each fleet in a queue, and use the alias names when setting destinations in your queue.
• Destinations in a queue must be running game builds that are compatible with the game clients that use the queue to get new game sessions. New game session requests that are processed by the queue can potentially be placed on any queue destination.

• A queue should have at least two fleets that reside in different Regions. This design improves hosting resiliency by decreasing the impact of fleet or regional slowdowns. It also enables the queue to more efficiently manage unexpected changes in player demand.

• The list order of destinations in a queue matters. A queue prioritizes placement choices based on several elements, including destination list order. If a queue is configured to prioritize on destination, game sessions will always be placed on the first fleet listed, if there's an available game server to host it.

• When deciding what Region to create your game session queue in, consider where most of its traffic will be coming from. In most games, a game client service, such as a session directory service, manages new game session requests. By positioning your queue in a Region that is geographically close to where your client service is deployed, you can minimize communication latency.

• When choosing fleets to provide hosting in specific locations, you can use multi-location fleets that cover a larger set of locations than you want. Use the queue filter configuration to prevent the queue from placing game sessions in specified locations. You can use at least 2 multi-location fleets with different home regions to mitigate the impact of game placements during a regional outage.

• A queue cannot have fleets with different certificate configurations. All fleets in the queue must have TLS certificate generation either enabled or disabled.

Best practices for queues with Spot fleets

If your queue includes Spot fleets, you want to set up a resilient queue that takes advantage of cost savings with Spot fleets while minimizing the effect of game session interruptions. For help correctly building fleets and game session queues for use with Spot fleets, see the Tutorial: Set up a game session queue for Spot Instances (p. 143). Learn more about Spot instances in Using Spot Instances with GameLift (p. 110).

In addition to the general best practices in the previous section, consider these Spot-specific best practices:

• Include at least one Spot fleet and one On-Demand fleet to cover all the geographic locations where you want to support players. This design is critical to minimizing the impact of potential Spot interruptions. It ensures that game sessions can be placed in any location, even when no Spot fleets are currently viable there. At a minimum, every queue should have at least one On-Demand fleet.

• Include fleets that use different instance types, preferably in the same instance family (c5.large, c5.xlarge, etc.). This design lessens the likelihood that a Spot interruption, which usually affects a specific instance type, will impact all instances in the same location. Check the historical pricing data in the Amazon GameLift console to verify that your preferred instance types typically deliver significant cost savings with Spot. You can view pricing for Spot and On-Demand instances as well as estimated Spot savings per instance.

• When configuring priorities for a queue with Spot fleets, place cost near the top of the list. This will ensure that locations on Spot fleets will always take precedence over locations on On-Demand fleets, when available.

Design a game session queue

A GameLift game session queue is a key component in your game management layer. A queue is responsible for processing new game session requests, locating a game server to host a new game session, and prompting the game server to start the game session. The way you design your queue determines (1) where GameLift can search for available game servers, and (2) how GameLift prioritizes available game servers to find an optimal placement for the request. The goal of this topic is to help you
design a queue that delivers the best possible experience to your players and efficiently uses the hosting resources you provision and pay for.

Queues are required with these GameLift features:

- Matchmaking with FlexMatch (see FlexMatch integration roadmap)
- Spot fleets (see Using Spot Instances with GameLift (p. 110))

Learn more about queues in the section called “Running game sessions” (p. 5). For information on creating a queue, see Create a game session queue (p. 148). For information on creating new game sessions with queues, see Create Game Sessions (p. 59). For help creating queues for Spot fleets, see the Tutorial: Set up a game session queue for Spot Instances (p. 143).

Design multiple queues as needed

Depending on your game and players, you may need to create more than one queue. When your game client service requests a new game session, it specifies which game session queue to use. Some issues to consider:

- If you have more than one variation of your game server, create a separate queue for each variation. All fleets in a queue must deploy compatible game servers, because players who use the queue to join games must be able to play on any of the queue's game servers.
- You might want to set up special queues for certain player groups. This allows you to customize how game sessions are placed based on situation. For example, you might need queues customized for certain game modes that require a special instance type or runtime configuration. Or you might want a special queue to manage placements for a tournament or other event. You can use fleets with different configurations and change how placements are prioritized.
- If you’re not providing player latency data, you might set up queues to serve specific geographic locations.
- You can set up queues based on how you want game session placement metrics to be collected. See GameLift metrics for queues (p. 176).
- In some cases, your fleet configuration may require multiple queues. For example, all fleets in a queue must have the same TLS certificate generation setting.

Build a multi-region queue

A multi-region design is recommended for all queues. This design can improve placement speed and hosting resiliency, and is critical when you want to use player latency data to put players into game sessions with optimal gameplay experiences. If you're building multi-region queues that use Spot fleets, see the Tutorial: Set up a game session queue for Spot Instances (p. 143).

One option is to add a multi-location fleet to a queue. The queue is able to place game sessions in any of the fleet's locations. You don't need to add more fleets to increase regional coverage, although you might want to add other fleets with different configurations or home Regions for redundancy. If you're using multi-location Spot fleet, follow best practices and include an On-Demand fleet that includes the same locations.

The following example outlines the process of designing a basic multi-region queue. For this example, we'll use a game server build that is deployed in two fleets, one Spot and one On-Demand. Each fleet has the following locations: us-east-1, us-east-2, ca-central-1 and us-west-2.

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. Keep in mind that the list order of destinations may affect how game sessions are placed, depending on the queue's prioritization order. In this example, we list our Spot fleet first, and our On-Demand fleet second.

3. If you want to prevent game sessions from being placed in any of the destination fleet's locations, create a filter configuration that lists the locations where game sessions can be placed. In this example, we want to place in all locations, so no filter configuration is required.

4. Choose the queue's priority configuration to determine where the queue will search first for an available game server. In this example, we opt to use the default priority order. Since our game session placement requests do not include player latency data, our queue will prioritize locations based first on destinations, and then on locations. The result of this choice is as follows:
   - The first destination listed in the queue is our Spot fleet.
   - Fleet locations are listed alphabetically as follows: ca-central-1, us-east-1, us-east-2, us-west-2.
   - As a result, new game sessions will always be placed in the ca-central-1 location of the Spot fleet, unless there are no available game servers there. In that event, the queue will then place new game sessions in us-east-1 of the Spot fleet until it is filled, and so on. Game sessions will only be placed in On-Demand fleet locations if all game servers on the Spot fleet are in use.

As our queue is put into action, we can use metric data to determine how well the queue design is holding up.

**Prioritize where game sessions are placed**

GameLift uses FleetIQ algorithms to determine the best possible queue location to place a new game session. It does this by prioritizing queue locations based on an ordered set of criteria.

The priority criteria are: regional player latency data (as reported in a game session placement request), hosting cost, geographical location (based on a provided order, and destination fleet (based on list order). The order these criteria are applied significantly affects how placement occurs. You can opt to use the default order or you can customize it by providing a priority configuration.

**Default priority order**

FleetIQ applies the criteria in the following default order:

- For placement requests that include player latency data, FleetIQ prioritizes placement in locations as follows:
  1. Lowest average latency for all players in the request.
  2. Lowest hosting cost, if latency is equal in multiple locations. Hosting cost is based primarily on a combination of the instance type and location (a combination of the fleet's home Region and remote location).
  3. Destination list order, if latency and cost is equal in multiple locations. Destinations are prioritized based on the order they are listed in the queue configuration.
  4. Location, if latency, cost, and destination are equal in multiple locations. Locations are prioritized alphabetically based on the Region code.

- For placement requests that do **not** include player latency data, FleetIQ prioritizes placement in locations as follows. This approach tends to place all game sessions on the first listed destination fleet, in whatever location comes first alphabetically. Game sessions are only placed in other locations and fleets when the first location fills up:
  1. Destination list order. Destinations are prioritized based on the order they are listed in the queue configuration.
  2. Location, if destination is equal in multiple locations. Locations are prioritized alphabetically based on the Region code.
Custom priority order

When customizing your queue's priority order, create a priority configuration (see PriorityConfiguration). You can create a new queue with this configuration or update an existing queue.

You can include any or all of the prioritization criteria (LATENCY, COST, DESTINATION, LOCATION). Keep in mind that FleetIQ appends any criteria that's not explicitly mentioned to the end of your list, based on the default order. For example, if you only specify destination in your priority configuration, FleetIQ will apply the following order: (1) destination, (2) latency, (3) cost, (4) location.

If you include LOCATION in your priority configuration, you also have to provide an ordered list of locations for FleetIQ to use.

Create a player latency policy

If your game session placement requests include player latency data, FleetIQ works to place game sessions in locations that offer the lowest average regional latency for all players in the request. Making placements based on average player latency protects most players from being put into games with unacceptable latency, but it doesn't account for extreme outliers. To accommodate these outliers, you need to put a latency cap in place.

A latency cap policy ensures that a requested game session is not placed where any individual player in the request would experience latency over the maximum value. Keep in mind, however, that a latency cap can also increase the risk that a game session request with latency outliers might never be fulfilled. Instead of a single cap, you can use a series of policies to gradually raise the latency maximum over time. You can customize how to balance providing 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.

Evaluate queue metrics

Use metrics to evaluate how well your queues are performing. You can view queue-specific metrics in the GameLift console (View Queue Details (p. 154)) or in Amazon CloudWatch. Queue metrics are described in GameLift metrics for queues (p. 176).

Queue metrics can provide insight in three main areas:

- **Overall queue performance** – Metrics indicate how successfully a queue is responding to placement requests and help to identify when and why placements are failing. For queues with manually scaled fleets, metrics average wait times and queue depth can indicate when capacity for a queue might need to be adjusted.
• FleetIQ performance – For placement requests that use FleetIQ's filtering and prioritization (that is, requests with player latency data), metrics indicate how often FleetIQ is able to find optimal placement for new game sessions. Optimal placement may include finding resources with the lowest possible player latency or, when Spot fleets are available, with the lowest cost. There are also error metrics that identify common reasons why optimal placement failed.

• Region-specific placements – For multi-region queues, metrics show successful placements by broken down by Region. With queues that use FleetIQ, this data provides useful insight into where player activity is occurring.

When evaluating metrics for FleetIQ performance, consider the following tips:

• Use the “placements succeeded” metric in combination with the FleetIQ metrics for lowest latency and/or lowest price to track the queue's rate of optimal placement.

• To boost a queue's optimal placement rate, take a look at the error metrics for the following:
  • If the error metric “first choice not available” is high, this is a good indicator that capacity scaling for the queue's fleets needs to be adjusted. It may be that all fleets in the queue are under-scaled, or there may be one particular fleet or Region that is an optimal fit for most placements.
  • If the error metric “first choice not viable” is high, this is an indicator to look at your Spot fleets. Spot fleets are considered “not viable” when the interruption rate for a particular instance type is too high. To resolve this issue, change the queue to use Spot fleets with different instance types. As noted in the best practices for queues with Spot fleets, it is always a good idea to include Spot fleets with different instance types in each Region.

Tutorial: Set up a game session queue for Spot Instances

Introduction

This tutorial describes how to set up game session placement for games that are deployed on low-cost Spot fleets. GameLift uses queues to field game session placement requests, locate available game servers to host them, and start the new game sessions. When building queues for Spot fleets, you need to take additional steps to minimize Spot interruptions and maintain continual game server availability for your players. This tutorial follows best practices and provides tips on building queues for Spot fleets.

Intended audience

This tutorial is for game developers who use GameLift Spot fleets to host a custom game server or Realtime Servers and need to set up game session placement. This tutorial assumes that you have a basic understanding of How GameLift works (p. 3).

What you’ll learn

This tutorial outlines the key decision points and steps required to build a game session placement layer for Spot fleets. In it, you will learn how to do the following:

• Define the group of players (scope) who will be served by your game session queue.
• Build a fleet infrastructure to support the game session queue's scope.
• Assign an alias to each fleet to abstract the fleet ID.
• Create a queue, add fleets, and prioritize where game sessions are placed.
• Add player latency policies to help minimize latency issues for every player.

Prerequisites

Before creating fleets and queues for game session placement, you should complete the following tasks:
• Complete the required game server integration tasks. Your game server must be able to receive prompts from the GameLift service to start and stop game sessions.
• Upload your game server build or Realtime script to the GameLift service for deployment.
• Plan your fleet configuration. At a minimum, you need to know which instance types you want to use (choose at least one, preferably two or three), and whether your fleets require an Instance role ARN and/or a TLS certificate. These properties can't be changed after a fleet is created.

**Step 1: Define the scope of your queue**

Your game's player population might have groups of players who will never play together in the same game sessions. For example, if your game is published in two languages, you need to place one player segment into Japanese game servers and another segment into English game servers.

To set up game session placement for your player population, you create a separate queue for each player segment. Each queue must be scoped appropriately in order to place players into the correct game servers. Some common ways that queues are scoped include:

• By geographic locations. When deploying your game servers in multiple geographic areas, you might build separate queues for players in each smaller area to limit the impact on player latency.
• By build/script variations. If you have more than one variation of your game server, you might be supporting player groups that can't play in the same game sessions. For example, game server builds/scripts might support different languages, device types, etc.
• By event types. You might create a special queue to manage games for participants in tournaments or other special events.

**Tutorial example**

In this tutorial, we are designing a queue for a game that has only one game server build variation. For launch, the game is being released only in two Asia Regions: ap-northeast-2 (Seoul) and ap-southeast-1 (Singapore). Because these Regions are relatively close to each other, latency isn't a significant issue for our player population. For this example, we have only one player segment, which means we need to create just one queue. At some future point, when the game is being released in North America, we'll create a second queue that is scoped for North American players.

**Step 2: Create Spot fleet infrastructure**

Build a set of fleets to host games for your player segment. Create fleets in Regions and with game server builds/scripts that fit the scope that you defined in Step 1. Your infrastructure should fit the following criteria:

• **Create fleets in at least two Regions.** It's possible for a Region to have a temporary outage. By having game servers hosted in at least one other region, you mitigate the impact of regional outages on your players. Even if you're supporting a player base in only one Region, place back-up fleets in a second Region nearby. You can keep your back-up fleets scaled down, and use auto scaling to increase capacity if usage increases.
• **Create at least one On-Demand fleet in each Region.** In each Region, create at least one Spot fleet and one On-Demand fleet. Since Spot fleets can become unavailable on short notice, On-Demand fleets provide back-up game servers for your players. You can keep your backup fleets scaled down until they're needed, and use auto scaling to increase On-Demand capacity when Spot fleets are unavailable.
• **Select different instance types across multiple Spot fleets in a Region.** If you are creating more than one Spot fleet in a region, vary the instance types of the fleets. If one Spot Instance type becomes temporarily unavailable, the interruption affects only one Spot fleet in the Region. Best practice is to choose widely available instance types, and use instance types in the same family (for example,
m5.large, m5.xlarge, m5.2xlarge). Use the GameLift console to check historical pricing data for instance types.

- **Use the same or a similar game build or script for all fleets.** The queue might put players into game sessions on any fleet in the queue. Players must be able to play in any game session on any fleet.
- **Use the same TLS certificate setting for all fleets.** Game clients that connect to game sessions in your fleets need to have compatible communication protocols.

When you’re ready to build your fleets, see Deploy a GameLift fleet with a custom game build (p. 112) or Deploy a Realtime Servers Fleet (p. 116) for detailed instructions on using the GameLift console or the AWS CLI to create new fleets.

**Tip**

We highly recommend including the fleet type (Spot or On-demand) in your fleet names. This makes it much easier to identify fleet types when viewing a list of fleets.

**Tutorial example**

As determined in Step 1, our queue needs to support players in two Regions. We need to create a two-Region infrastructure with at least one Spot fleet and one On-Demand fleet in each Region. Every fleet deploys the same game server build. In addition, we anticipate that player traffic will be heavier in the Seoul Region, so we want to double up on our Spot fleets there. By adding another Spot fleet with a different instance type, we increase the likelihood that games in that Region will be hosted on low-cost Spot instances even if Spot interruptions occur.

Our resulting fleet infrastructure includes five fleets in total:

**ap-northeast-2 (Seoul)**
- Spot fleet, c5.large instance type
- Spot fleet, c5.xlarge instance type
- On-Demand fleet, c5.large instance type

**ap-southeast-1 (Singapore)**
- Spot fleet, c5.large instance type
- On-Demand fleet, c5.large instance type

**Step 3: Assign aliases for each fleet**

Create a new alias for each fleet in your infrastructure. Aliases are used to abstract fleet identities; because fleets need to be replaced periodically for a variety of reasons, aliases make this process simple and efficient. When your fleets are created, see Add an alias to a GameLift fleet (p. 122) for instructions on using the GameLift console or the AWS CLI to create new aliases and point them to the appropriate fleet. You must create the alias in the same Region as the fleet it will point to.
Tip
We highly recommend including the fleet type ("spot" or "on-demand") in your alias names. This makes it much easier to identify the fleet type when viewing a list of aliases.

Tutorial example
Our fleet infrastructure has five fleets, so we need to create five aliases, choosing the "simple" routing strategy and selecting one of our five fleets. We need three aliases in the ap-northeast-2 (Seoul) Region, and two aliases in the ap-southeast-1 (Singapore) Region.

Step 4: Create a queue with destinations
With your fleets and aliases prepared, you’ve done the hard part of setting up the infrastructure for game session placement. Now you need to create the game session queue itself and add your fleet destinations. See Create a game session queue (p. 148) for detailed instructions on using the GameLift console or the AWS CLI to create your new queue.

When creating your queue:

- Keep the default timeout of 10 minutes to start with. You'll want to test out how your queue timeout affects your players' wait times for getting into games.

  Tip
  You'll want to monitor your game's queue timeout rate. It's a good indicator when something is slowing down your game session placement pipeline.

- Skip the section on player latency policies for now. We'll cover this in the next step.

- Prioritize the fleets in your queue. Fleet prioritization determines where the queue looks first when searching for available resources to host a new game session. You might choose to prioritize by Region, instance types, fleet type, and so on. When working with Spot fleets, we recommend either of the following approaches:

  - If your infrastructure uses a primary Region with fleets in a second Region for back-up only, you want to prioritize fleets first by region, and then by fleet type. With this approach, all fleets in the primary Region are placed at the top of the list, with Spot fleets followed by On-Demand fleets.
  
  - If your infrastructure uses multiple Regions equally, you want to prioritize fleets by fleet type, placing Spot fleets at the top of the list.

  Note
  When a game session placement request contains player latency information, your default prioritization may change. In this case GameLift FleetIQ re-prioritizes the queue's fleets by Region and tries to place game sessions where players are reporting the lowest latency.

Tutorial example
For this example, we create a new queue with the name "MBG_ASIA_spot", and add the aliases for all five of our fleets. Our game session requests do include latency information, but we still need to implement a prioritization strategy. For this queue, we prioritize placements first by Region and second by fleet type. The following list details the queue destinations in prioritized order:
• Alias for ap-northeast-2 (Seoul), c5.large Spot fleet
• Alias for ap-northeast-2 (Seoul), c5.xlarge Spot fleet
• Alias for ap-northeast-2 (Seoul), c5.large On-Demand fleet
• Alias for ap-southeast-1 (Singapore), c5 large Spot fleet
• Alias for ap-southeast-1 (Singapore), c5 large On-Demand fleet

Based on this configuration, this queue will always attempt to place new game sessions into a Spot fleet in Seoul. When those fleets are full, the queue will use available capacity on the Seoul On-Demand fleet as a backup. Only once all three Seoul fleets are unavailable will game sessions be placed on the Singapore fleets.

Step 5: Add latency limits to the queue

If your game includes regional player latency information in game session placement requests, FleetIQ works to place players into game sessions with the lowest possible latency experience. For game session requests that include multiple players, FleetIQ uses the average latency values of all players in the request. While this approach works most of the time, it occasionally puts one or more of the players into a game session with unacceptable latency.

To handle these outlier cases, you can set up a player latency policy in your queue to enforce a hard latency limit for all individual players in a game session placement request. This policy ensures that no individual player gets put into a game session with an unreasonably high latency. A latency policy can increase player wait times for game session placements, as the queue searches for an acceptable game server, or it can prevent a placement altogether. Your policy can also include relaxable rules that increase the maximum allowed latency over time. See Create a player latency policy (p. 142) for additional help with player latency policies.

**Tip**

If you want to manage player latency more specifically, such as to require similar latency across all players in a group, you can use GameLift FlexMatch and create latency-based matchmaking rules.

**Tutorial example**

Our game includes latency information in our game session placement requests, and we have a player party feature that creates a game session for a group of players. We are willing to have players wait a little longer to get into games with the best possible gameplay experience. Our game tests show that latencies under 50 milliseconds are optimal, and the game becomes unplayable at latencies over 250 milliseconds. Players’ tolerance for wait times starts to fall off at about one minute. We can use this information to set the parameters of our player latency policy.

The GameLift console provides a simple UI for building player latency policy statements. For our queue, which has a five-minute timeout (300 seconds), we add the following policy statements. (You can also set player latency policies using the AWS CLI command update-game-session-queue.)

1. Spend 60 seconds searching for a destination where all player latencies are less than 50 milliseconds, then...
2. Spend 30 seconds searching for a destination where all player latencies are less than 125 milliseconds, then...
3. Spend the remaining queue timeout searching for a destination where all player latencies are less than 250 milliseconds.

With this policy, our queue looks for placements with optimal latency (under 50ms) for the first minute, and then relaxes the limit fairly quickly after that. The queue does not make any placements where one player would have a latency of 250ms or higher.

Summary

Congratulations! If you've followed the steps in this tutorial, you now have a game session queue that is scoped for a particular segment of your player population. It uses Spot fleets effectively and is resilient when rare Spot interruptions occur. In addition, your queue's fleets are strategically prioritized and you've added hard latency limits to protect players from bad gameplay experiences.

You can now use the queue to place game sessions for the player segment it serves. When making game session placement requests for these players, you must reference this game session queue name in the request. For more information on making game session placement requests, see Create Game Sessions (p. 59) or Integrating a Game Client for Realtime Servers (p. 71).

Create a game session 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. 139).

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. 59).

To create a queue

Console

Game session queues that are created using the Console have the following characteristics:

- Game sessions can be placed in any location that is part of any destination fleet in the queue. If you want to exclude one or more locations, use the AWS CLI to update a queue, using update-game-session-queue, to add a filter configuration.
- FleetIQ automatically applies the default prioritization when placing game sessions. If you want to customize how your queue handles prioritization, use the AWS CLI to update a queue, using update-game-session-queue, to add a priority configuration.

1. Open the GameLift console at https://console.aws.amazon.com/gamelift/, and choose the region you want to create your queue in.
2. From the 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 GameLift to try to place a new game session before stopping. 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, 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 Create a player latency policy (p. 142).

   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 polices 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. All fleets in the queue must have the same certificate configuration (either GENERATED or DISABLED). All fleets should 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 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. Optionally, add the ARN for an SNS topic where you want to receive placement-related event notifications. Learn more about Set up event notification for game session placement (p. 151).

7. 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.

**AWS CLI**

You can use the AWS Command Line Interface (AWS CLI) to create a queue. Get and install the AWS Command Line Interface tool.
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 game session queue that has two fleet destinations and allows up to five minutes for the placement process. Fleets are listed as destinations and identified by either a fleet ARN or alias ARN. All fleets and aliases must already exist. The queue is configured with a filter configuration, a custom priority configuration, and a notification target for tracking placement events. The priority configuration looks at hosting cost first, followed location, applied by the defined order.

**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.

```bash
$ 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
--filter-configuration "AllowedLocations=us-east-1, ca-central-1, us-east-2, us-west-2"
--priority-configuration "PriorityOrder=COST,LOCATION LocationOrder=us-east-2,ca-central-1,us-east-1,us-west-2,us-east-1"
```

**Copiable version:**

```bash
```

If the `create-game-session-queue` request is successful, 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.

```bash
# 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
```

Version
150
Set up event notification

If you’re using queues to manage game session placement in your game, you need a way to monitor the status of individual placement requests and take action as appropriate. Implementing event notifications is a fast and efficient method for tracking placement activity. If your game is in production, or in pre-production with high-volume placement activity, you should be using event notifications.

There are two options for setting up event notifications. You can set up an SNS topic and have GameLift publish event notifications on placement activity by referencing the topic ID in a game session queue. Alternatively, you can use Amazon CloudWatch Events, which has a suite of tools available for managing events and taking action on them.

See the list of game session placement events emitted by GameLift in Game session placement events (p. 264). Each placement event is identified by the service (GameLift), the game session queue, and the placement request ID.

Set up an SNS topic

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

To set up an SNS topic for 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 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 topic ARN to open it. Choose the topic action Edit topic policy, and go to the Advanced view tab.
You can optionally set additional access control for the topic. By adding conditions to your topic resource policy, you can control which queues can publish to the topic. Specify queues by their ARNs. You can only add queues that belong to the same AWS account as the topic. Cross-account notification publishing is not supported.

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": [
    {
      "Sid": "__default_statement_ID",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "SNS:GetTopicAttributes",
        "SNS:SetTopicAttributes",
        "SNS:AddPermission",
        "SNS:RemovePermission",
        "SNS:DeleteTopic",
        "SNS:Subscribe",
        "SNS:ListSubscriptionsByTopic",
        "SNS:Publish",
        "SNS:Receive"
      ],
      "Resource": "arn:aws:sns:your_region:your_account:your_topic_name",
      "Condition": {
        "StringEquals": {
          "AWS:SourceAccount": "your_account"
        }
      }
    },
    {
      "Sid": "__console_pub_0",
      "Effect": "Allow",
      "Principal": {
        "Service": "gamelift.amazonaws.com"
      },
      "Action": "sns:Publish",
      "Resource": "arn:aws:sns:your_region:your_account:your_topic_name",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:gamelift:your_region:your_account:gamesessionqueue/your_queue_name"
        }
      }
    }
  ]
}
```

Set up CloudWatch Events

GameLift automatically posts all game session placement events to CloudWatch Events. With CloudWatch Events you can set up rules to have 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 “PlacementFulfilled” to an AWS Lambda function that handles tasks that precede connecting to a game.
session. 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 queues, you can leave the notification target field empty, or reference an SNS topic if you want to use both options.

To access GameLift placement events in CloudWatch Events, go to the Amazon CloudWatch console and open Events. Be sure that you're in the AWS Region where your game session queue resides. For more information about getting account credentials to access CloudWatch Events, see Sign in to the Amazon CloudWatch Console.

You'll want to set up CloudWatch Events rules to initiate action in response to a queue event. Learn more about this in Creating a CloudWatch Events Rule. Here are some relevant examples:

• CloudWatch Events rule that matches events from all GameLift queues

```json
{
    "source": [
        "aws.gamelift"
    ],
    "detail-type": [
        "GameLift Queue Placement Event"
    ]
}
```

• Example CloudWatch Events rule that matches events from a specific queue

```json
{
    "source": [
        "aws.gamelift"
    ],
    "detail-type": [
        "GameLift Queue Placement Event"
    ],
    "resources": [
        "arn:aws:gamelift:your_region:your_account:gamesessionqueue/your_queue_name"
    ]
}
```

View Your Queues

You can view information on all existing game session placement queues (p. 137). 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. 154).

**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.

Queue metrics include a range of information describing placement activity across the entire queue, as well as successful placements broken down by region. The region-specific data is useful for tracking

Version

154
where your games are being hosted. With queues that use FleetIQ and prioritize placements to minimize player latency and hosting costs, regional placement metrics can help to detect issues with the overall queue design.

Queue metrics are also available in Amazon CloudWatch. You can view the descriptions of all metrics at GameLift metrics for queues (p. 176).

**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.
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. You can refresh the graph any time by clicking the refresh button in the graph's upper right corner.
   - **Time zone** – 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.

**How GameLift FleetIQ works**

FleetIQ is the GameLift algorithm that manages how a game session queue selects hosting resources for game session placements. A queue is configured with specific fleet destinations where game sessions can be placed. FleetIQ relies on 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 game session placement request provides player latency data, FleetIQ evaluates the queue's player latency policies and removes all fleets in Regions where the latency that is reported by a player in the request exceeds a policy's maximum limit.
   - FleetIQ removes any Spot fleets that are not currently viable due to unacceptable interruption rates.
2. FleetIQ prioritizes the remaining queue destinations based on the following:
   - If player latency data is provided, FleetIQ re-orders the queue destinations by Region, those with lowest average player latency listed first.
   - If player latency data is not provided, FleetIQ uses the original list of queue destinations.
3. FleetIQ selects a destination from the prioritized list.
   - If the destination list was prioritized by Region, FleetIQ selects the fleet in the lowest-latency Region with the lowest price. If there are no viable Spot fleets, any fleet in that Region may be selected.
   - If the destination list was not prioritized, FleetIQ selects the first viable fleet from the original list, even if there are lower priced Spot fleets on the list.
4. FleetIQ evaluates whether the selected fleet has a server process available to host a new game session. It is considered an "optimal" placement when the new game session is placed in a fleet with the lowest possible latency and/or the lowest price.
5. If the selected fleet has no available resources, FleetIQ moves to the next listed destination and repeats until it finds a fleet to host the new game session.
VPC peering for GameLift

This topic provides guidance on how to set up a VPC peering connection between your GameLift-hosted game servers and your other non-GameLift resources. Use Amazon Virtual Private Cloud (VPC) peering connections to enable your game servers to communicate directly and privately with your other AWS resources, such as a web service or a repository. You can establish VPC peering with any resources that run on AWS and are managed by an AWS account that you have access to.

Note
VPC peering is an advanced feature. To learn about preferred options for enabling your game servers to communicate directly and privately with your other AWS resources, see Communicate with other AWS resources from your fleets (p. 55).

If you're already familiar with Amazon VPCs and VPC peering, understand that setting up peering with GameLift game servers is somewhat different. You don't have access to the VPC that contains your game servers—it is controlled by the GameLift service—so you can't directly request VPC peering for it. Instead, you first pre-authorize the VPC with your non-GameLift resources to accept a peering request from the GameLift service. Then you trigger GameLift to request the VPC peering that you just authorized. GameLift handles the tasks of creating the peering connection, setting up the route tables, and configuring the connection.

To set up VPC peering for an existing fleet

1. Get AWS Account ID(s) and credentials.

   You need an ID and sign-in credentials for the following AWS accounts. You can find AWS account IDs by signing into the AWS Management Console and viewing your account settings. To get credentials, go to the IAM console.
   
   • AWS account that you use to manage your GameLift game servers.
   
   • AWS account that you use to manage your non-GameLift resources.

   If you're using the same account for GameLift and non-GameLift resources, you need ID and credentials for that account only.

2. Get identifiers for each VPC.

   Get the following information for the two VPCs to be peered:
   
   • VPC for your GameLift game servers – This is your GameLift fleet ID. Your game servers are deployed in GameLift on a fleet of EC2 instances. A fleet is automatically placed in its own VPC, which is managed by the GameLift service. You don't have direct access to the VPC, so it is identified by the fleet ID.
   
   • VPC for your non-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. Once you have created a VPC, you can find the VPC ID by signing into the AWS Management Console for Amazon VPC and viewing your VPCs.

   Note
   When setting up a peering, both VPCs must exist in the same region. The VPC for your GameLift fleet game servers is in the same region as the fleet.
3. **Authorize a VPC peering.**

   In this step, you are pre-authorizing a future request from GameLift to peer the VPC with your game servers with your VPC for non-GameLift resources. This action updates the security group for your VPC.

   To authorize the VPC peering, call the GameLift service API `CreateVpcPeeringAuthorization()` or use the AWS CLI command `create-vpc-peering-authorization`. Make this call using the account that manages your non-GameLift resources. Identify the following information:
   - Peer VPC ID – This is for the VPC with your non-GameLift resources.
   - GameLift AWS account ID – This is the account that you use to manage your 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 connection.**

   With a valid authorization, you can request that GameLift establish a peering connection.

   To request a VPC peering, call the GameLift service API `CreateVpcPeeringConnection()` or use the AWS CLI command `create-vpc-peering-connection`. Make this call using the account that manages your GameLift game servers. Use the following information to identify the two VPCs that you want to peer:
   - Peer VPC ID and AWS account ID – This is the VPC for your non-GameLift resources and the account that you use to manage them. The VPC ID must match the ID on a valid peering authorization.
   - Fleet ID – This identifies the VPC for your GameLift game servers.

5. **Track the peering connection status.**

   Requesting a VPC peering connection is an asynchronous operation. To track the status of a peering request and handle success or failure cases, use one of the following options:
   - Continuously poll with `DescribeVpcPeeringConnections()`. This operation retrieves the VPC peering connection record, including the status of the request. If a peering connection is successfully created, the connection record also contains a CIDR block of private IP addresses that is assigned to the VPC.
   - Handle fleet events associated with VPC peering connections with `DescribeFleetEvents()`, including success and failure events.

   Once the peering connection is established, you can manage it using the following operations:
   - `DescribeVpcPeeringConnections()` (AWS CLI `describe-vpc-peering-connections`).
   - `DeleteVpcPeeringConnection()` (AWS CLI `delete-vpc-peering-connection`).

**To set up VPC peering with a new fleet**

You can create a new GameLift fleet and request a VPC peering connection at the same time.
1. **Get AWS Account ID(s) and credentials.**

   You need an ID and sign-in credentials for the following two AWS accounts. You can find AWS account IDs by signing into the AWS Management Console and viewing your account settings. To get credentials, go to the IAM console.
   - AWS account that you use to manage your GameLift game servers.
   - AWS account that you use to manage your non-GameLift resources.

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

2. **Get the VPC ID for your non-GameLift AWS resources.**

   If you haven’t already created a VPC for these resources, do so now (see Getting Started with Amazon VPC). Be sure that you create the new VPC in the same region where you plan to create your new fleet. If your non-GameLift resources are managed under a different AWS account or user/user group than the one you use with GameLift, you’ll need to use these account credentials when requesting authorization in the next step.

   Once you have created a VPC, you can locate the VPC ID in Amazon VPC console by viewing your VPCs.

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

   When GameLift creates the new fleet and a corresponding VPC, it also sends a request to peer with the VPC for your non-GameLift resources. You need to pre-authorize that request. This step updates the security group for your VPC.

   Using the account credentials that manage your non-GameLift resources, call the GameLift service API `CreateVpcPeeringAuthorization()` or use the AWS CLI command `create-vpc-peering-authorization`. Identify the following information:
   - Peer VPC ID – ID of the VPC with your non-GameLift resources.
   - GameLift AWS account ID – ID of the account that you use to manage your 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. **Follow the instructions for creating a new fleet using the AWS CLI (p. 112).** Include the following additional parameters:
   - `peer-vpc-aws-account-id` – ID for the account that you use to manage the VPC with your non-GameLift resources.
   - `peer-vpc-id` – ID of the VPC with your non-GameLift account.

   A successful call to `create-fleet` with the VPC peering parameters generates both a new fleet and a new VPC peering request. The fleet's status is set to **New** and the fleet activation process is initiated. The peering connection request's status is set to **initiating-request**. You can track the success or failure of the peering request by calling `describe-vpc-peering-connections`.

   When requesting both a new fleet and a VPC peering connection, both actions either succeed or fail. If a fleet fails during the creation process, the VPC peering connection will not be established. Likewise, if a VPC peering connection fails for any reason, the new fleet will fail to move from status **Activating** to **Active**.
Note
The new VPC peering connection is not completed until the fleet is ready to become active. This means that the connection is not available and can't be used during the game server build installation process.

The following example creates both a new fleet and a peering connection between a pre-established VPC and the VPC for the new fleet. The pre-established VPC is uniquely identified by the combination of your non-GameLift AWS account ID and the VPC ID.

```bash
$ AWS gamelift create-fleet
   --name "My_Fleet_1"
   --description "The sample test fleet"
   --fleet-type "ON_DEMAND"
   --build-id "build-1111aaaa-22bb-33cc-44dd-5555eeee66ff"
   --runtime-configuration "GameSessionActivationTimeoutSeconds=300,
                            MaxConcurrentGameSessionActivations=2,
                            ServerProcesses=[[LaunchPath=C:\\game\\Bin64.dedicated
                                    MultiplayerSampleProjectLauncher_Server.exe,
                                    Parameters=+sv_port 33435 +start_lobby,
                                    ConcurrentExecutions=10]]"
   --new-game-session-protection-policy "FullProtection"
   --resource-creation-limit-policy "NewGameSessionsPerCreator=3,
                                       PolicyPeriodInMinutes=15"
   --ec2-inbound-permissions "FromPort=33435,ToPort=33435,IpRange=0.0.0.0/0,Protocol=UDP"
                             "FromPort=33235,ToPort=33235,IpRange=0.0.0.0/0,Protocol=UDP"
   --metric-groups "EMEAfleets"
   --peer-vpc-aws-account-id "111122223333"
   --peer-vpc-id "vpc-a11a11a"
```

Troubleshooting VPC peering issues

If you're having trouble establishing a VPC peering connection for your GameLift game servers, consider these common root causes:

- An authorization for the requested connection was not found:
  - Check the status of a VPC authorization for the non-GameLift VPC. It might not exist or it might have expired.
  - Check the regions of the two VPCs you're trying to peer. If they're not in the same region, they can't be peered.
- The CIDR blocks (see Invalid VPC Peering Connection Configurations) of your two VPCs are overlapping. The IPv4 CIDR blocks that are assigned to peered VPCs cannot overlap. The CIDR block of the VPC for your GameLift fleet is automatically assigned and can't be changed, so you'll need to change the CIDR block for of the VPC for your non-GameLift resources. To resolve this issue:
• Look up this CIDR block for your GameLift fleet by calling `DescribeVpcPeeringConnections()`.
• Go to the Amazon VPC console, find the VPC for your non-GameLift resources, and change the CIDR block so that they don't overlap.
• The new fleet did not activate (when requesting VPC peering with a new fleet). If the new fleet failed to progress to **Active** status, there is no VPC to peer with, so the peering connection cannot succeed.
Viewing Your Game Data in the Console

The managed GameLift service 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. 161)
- View Your Builds (p. 162)
- View Your Fleets (p. 163)
- View fleet details (p. 163)
- View Data on Game and Player Sessions (p. 167)
- View Your Aliases (p. 168)

View Your Current Amazon GameLift Status

The Dashboard provides a grid view showing the following:

- Builds
- Fleets in all statuses
- 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:

- 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 a 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 build. Select a build and click Delete build.
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. 163).

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 a selected fleet. A fleet's detail page (p. 163) contains usage information and metrics as well as game session and player session data; you can also edit the fleet record or terminate the fleet.

To view the Fleets page, choose Fleets from the Amazon GameLift console's menu bar.

The Fleets page displays the following summary information by default. You can customize which information to display by using the Settings (gear) button.

- **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 instances** – The number of EC2 instances in use for the fleet.
- **Maximum instances** – The current maximum number of EC2 instances allowed on the fleet. This value is configurable (within service limits) from the Fleet detail page, Scaling tab.
- **Game sessions** – The number of active 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 dashboard or the Fleets page by clicking the fleet name.

On this page you can also take the following actions:

- Update a fleet's attributes, port settings, and runtime configuration. Choose Actions: Edit fleet.
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.
- **OS** – Operating system on each instances in the fleet. A fleet’s OS is determined by the build deployed to it.
- **Fleet type** – Indicates whether the fleet uses on-demand or spot instances.
- **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.
- **Locations** – Number of locations where fleet instances are deployed, including the fleet’s home Region and additional remote locations.
- **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.
- **Game sessions** – Number of active game sessions running on instances in the fleet. The data has a five-minute delay.
- **Player sessions** – Number of players 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. 6) 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 displays a graphical representation of fleet metrics over time. To view metrics using Amazon CloudWatch, see Monitor GameLift with Amazon CloudWatch (p. 170).

To display metrics information in the graph

1. From the list shown to the left of the graph area, click a metric name to add it to the graph display. Metric names that are gray are currently not being graphed. The color key identifies which line matches each graphed metric. Descriptions of individual metrics can be found at GameLift metrics for fleets (p. 171). 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 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.

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

   • **Location** – View aggregate metrics for an entire fleet, or view metrics for an individual fleet location.
   • **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.
   • **Time zone** – 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.

#### Scaling history

View a graphical representation of capacity changes over time. Use the following filters:

   • **Location** – View aggregate metrics for an entire fleet, or view metrics for an individual fleet location.
   • **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.

#### Scaling limits

View current fleet capacity settings, broken out by fleet locations. You can also change limits and desired capacity in this section (see Scaling GameLift hosting capacity (p. 129)).
• **Location** – Name of a location where fleet instances are deployed.

• **Status** – Hosting status of the fleet location. Location status must be **ACTIVE** to be able to host games.

• **Minimum** – Hard lower limit on the number of instances to that can be deployed in the location. Fleet capacity cannot drop below this minimum.

• **Desired** – The target number of active instances to maintain the location. When there's a disparity between a fleet location's active instances and desired instances, a scaling event is initiated to either start or shut down instances as needed until active instances equals desired instances.

• **Maximum** – Hard upper limit on the number of instances that can be deployed in the location. Fleet capacity cannot exceed this maximum.

• **Available** – The service limit on instances (for the fleet instance type, home Region, and remote location) minus the number of instances currently in use. This value tells you the maximum number of instances that you can add to the location.

**Auto-scaling policies**

This section covers information about auto-scaling policies that are applied to the fleet identifies whether auto-scaling is enabled or disabled for each fleet location. You can set up or update a target-tracking policy. The fleet's rule-based policies, which must be defined using the AWS SDK or CLI, are displayed here. See **Auto-scale fleet capacity with GameLift** (p. 132).

**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. See **View Data on Game and Player Sessions** (p. 167) for more details.

**Capacity allocation**

The **Capacity allocation** tab displays the runtime configuration for the fleet, which specifies what server processes to launch on each instance. 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.

**ARNs**

The **Locations** tab lists all locations that are configured for this fleet, including the fleet's home Region and all remote locations. In addition, you can use controls in this tab to add or remove locations from the fleet.

• **Fleet ARN** – The identifier assigned to this fleet. A fleet's ARN identifies it as an GameLift resource and specifies the region and AWS account, to ensure that it is a unique identifier.

• **Instance Role ARN** – An identifier for an AWS IAM role that manages access to your other AWS resources, if one was provided during fleet creation. When a role ARN is associated with the fleet, the game servers and other applications that are running on the fleet are able to access those other AWS resources. Learn more at **Communicate with other AWS resources from your fleets** (p. 55).
Locations

The Locations tab lists all locations where fleet instances are deployed. Locations include the fleet's home Region and any remote locations that have been added. You can add or remove locations directly in this tab (see To update fleet locations (p. 121)).

- **Location** – Name of a location where fleet instances are deployed.
- **Status** – Hosting status of the fleet location. Location status tracks the process of activating the first instances in the location. Location status must be ACTIVE to be able to host games.
- **Active instances** – The number of instances with server processes running that are deployed to the fleet location.
- **Active servers** – The number of game server processes able to host game sessions that are currently running on instances in the fleet location.
- **Game sessions** – The number of game sessions currently being hosted on instances in the fleet location.
- **Player sessions** – The number of player sessions, which represent individual players, that are participating in game sessions that are currently being hosted in the fleet location.

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.

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. 12).

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** – 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.

### 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

If you're using GameLift FleetIQ as a standalone feature with Amazon EC2, also refer to Security in Amazon EC2 in the Amazon EC2 User Guide for Linux Instances.

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 GameLift with Amazon CloudWatch (p. 170)
- Logging Amazon GameLift API Calls with AWS CloudTrail (p. 183)

Monitor GameLift with Amazon CloudWatch

You can monitor 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 to provide a historical perspective on how your game server hosting with GameLift is performing. You can set alarms that watch for certain thresholds and send notifications or take actions when those thresholds are met. For more information, see the Amazon CloudWatch User Guide.

The following tables list the metrics and dimensions for GameLift. All metrics that are available in CloudWatch are also available in the GameLift console, which provides the data as a set of customizable graphs. To access CloudWatch metrics for your games, use the AWS Management Console, the AWS CLI, or the CloudWatch API.

If a metric does not have a location, it uses the home location.

**Dimensions for GameLift metrics**

GameLift supports filtering metrics by the following dimensions.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Filter metrics for a fleet deployment location. If a metric does not have a location, it uses the home location.</td>
</tr>
</tbody>
</table>
### GameLift metrics for fleets

The AWS/GameLift namespace includes the following metrics related to activity across a fleet or a group of fleets. Fleets are used with a managed GameLift solution. The 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 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></td>
<td>Dimensions: Location</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DesiredInstances</td>
<td>Target number of active instances that 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></td>
<td>Dimensions: Location</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></td>
<td>Dimensions: Location</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></td>
<td>Dimensions: Location</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></td>
<td>Dimensions: Location</td>
</tr>
</tbody>
</table>
### GameLift metrics for fleets

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PercentIdleInstances</td>
<td>Percentage of all active instances that are idle (calculated as ( \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></td>
<td>Dimensions: Location</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>
<tr>
<td></td>
<td>Dimensions: Location</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></td>
<td>Dimensions: Location</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></td>
<td>Dimensions: Location</td>
</tr>
<tr>
<td>PercentHealthyServerProcesses</td>
<td>Percentage of all active server processes that are reporting healthy (calculated as ( \text{HealthyServerProcesses} / \text{ActiveServerProcesses} )).</td>
</tr>
<tr>
<td></td>
<td>Units: Percent</td>
</tr>
</tbody>
</table>
### Metric

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Relevant CloudWatch statistics:</strong></td>
<td>Average, Minimum, Maximum</td>
</tr>
<tr>
<td><strong>Dimensions:</strong></td>
<td>Location</td>
</tr>
</tbody>
</table>

**ServerProcessAbnormalTerminations**

Server processes that were shut down due to abnormal circumstances since the last report. This metric includes terminations that were initiated by the GameLift service. This occurs when a server process stops responding, consistently reports failed health checks, or does not terminate cleanly (by calling `ProcessEnding()`).

- **Units:** Count
- **Relevant CloudWatch statistics:** Sum, Average, Minimum, Maximum
- **Dimensions:** Location

### Game sessions

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ActivatingGameSessions</strong></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><strong>Units:</strong></td>
<td>Count</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td></td>
<td>Dimensions: Location</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>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td></td>
<td>Dimensions: Location</td>
</tr>
<tr>
<td>AvailableGameSessions</td>
<td>Active, healthy server processes that are not currently being used to host a game session and can start a new game session without a delay to spin up new server processes or instances. This metric can be used with automatic scaling.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong></td>
</tr>
<tr>
<td></td>
<td>For fleets that limit concurrent game session activations, use the metric ConcurrentActivatableGameSessions. That metric more accurately represents the number of new game sessions that can start without any type of delay.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td></td>
<td>Dimensions: Location</td>
</tr>
<tr>
<td>ConcurrentActivatableGameSessions</td>
<td>Active, healthy server processes that are not currently being used to host a game session and can immediately start a new game session.</td>
</tr>
<tr>
<td></td>
<td>This metric differs from AvailableGameSessions in the following way: it does not count server processes that currently cannot activate a new game session because of limits on game session activations. (See the fleet RuntimeConfiguration optional setting MaxConcurrentGameSessionActivations). For fleets that don't limit game session activations, this metric is identical to AvailableGameSessions.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td></td>
<td>Dimensions: Location</td>
</tr>
</tbody>
</table>
### GameLift metrics for queues

The GameLift namespace includes the following metrics related to activity across a game session placement queue. Queues are used with a managed GameLift solution. The GameLift service sends metrics to CloudWatch every minute.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PercentAvailableGameSessions</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.</td>
</tr>
<tr>
<td></td>
<td>Units: Percent</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average</td>
</tr>
<tr>
<td></td>
<td>Dimensions: Location</td>
</tr>
<tr>
<td>GameSessionInterruptions</td>
<td>Number of game sessions on 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>
<tr>
<td></td>
<td>Dimensions: Location</td>
</tr>
</tbody>
</table>

### Player sessions

#### Metric

<table>
<thead>
<tr>
<th>CurrentPlayerSessions</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td>PlayerSessionActivations</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.</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>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AverageWaitTime</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, Sum</td>
</tr>
<tr>
<td></td>
<td>Dimensions: Location</td>
</tr>
<tr>
<td>FirstChoiceNotViable</td>
<td>Game sessions that were successfully placed but NOT in the first-choice fleet, because that fleet was considered not viable (such as a spot fleet with a high interruption rate). This metric is based on cost, not latency. The first-choice fleet is either the first fleet listed in the queue or—when a placement request includes player latency data—it is the first fleet chosen by FleetIQ prioritization. If there are no viable spot fleets, any fleet in that region may be selected.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum</td>
</tr>
<tr>
<td>FirstChoiceOutOfCapacity</td>
<td>Game sessions that were successfully placed but NOT in the first-choice fleet, because that fleet had no available resources. The first-choice fleet is either the first fleet listed in the queue or—when a placement request includes player latency data—it is the first fleet chosen by FleetIQ prioritization.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum</td>
</tr>
<tr>
<td>LowestLatencyPlacement</td>
<td>Game sessions that were successfully placed in a region that offers the queue's lowest possible latency for the players. This metric is emitted only when player latency data is included in the placement request, which triggers FleetIQ prioritization.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum</td>
</tr>
<tr>
<td>LowestPricePlacement</td>
<td>Game sessions that were successfully placed in a fleet with the queue's lowest possible price for the chosen region. (FleetIQ prioritization first chooses the region with the lowest latency for the players and then finds the lowest cost fleet within that region.) This fleet can be either a spot fleet or an on-demand instance if the queue has no spot instances. This metric is emitted only</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>when player latency data is included in the placement request.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum</td>
</tr>
<tr>
<td>Placement &lt;region name&gt;</td>
<td>Game sessions that are successfully placed in fleets located in the specified region. This metric breaks down the PlacementsSucceeded metric by region.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Sum</td>
</tr>
<tr>
<td>PlacementsCanceled</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, Sum</td>
</tr>
<tr>
<td>PlacementsFailed</td>
<td>Game session placement requests that failed for any reason since the last report.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum</td>
</tr>
<tr>
<td>PlacementsStarted</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, Sum</td>
</tr>
<tr>
<td>PlacementsSucceeded</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, Sum</td>
</tr>
<tr>
<td>PlacementsTimedOut</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, Sum</td>
</tr>
</tbody>
</table>
### GameLift metrics for matchmaking

The GameLift namespace includes metrics on FlexMatch activity for matchmaking configurations and matchmaking rules. FlexMatch matchmaking is used with a managed GameLift solution. The GameLift service sends metrics to CloudWatch every minute.

For more information on the sequence of matchmaking activity, see [How Amazon GameLift FlexMatch Works](#).

#### Matchmaking configurations

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum</td>
</tr>
<tr>
<td></td>
<td>Dimensions: Location</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CurrentTickets</strong></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, Sum</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MatchAcceptancesTimedOut</strong></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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MatchesAccepted</strong></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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MatchesCreated</strong></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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MatchesPlaced</strong></td>
<td>Matches that were successfully placed into a game session since the last report.</td>
</tr>
</tbody>
</table>

Version 179
<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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></td>
<td>Relevant CloudWatch statistics: Sum</td>
</tr>
<tr>
<td>PlayersStarted</td>
<td>Players in matchmaking tickets that were added 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>TicketsFailed</td>
<td>Matchmaking requests that resulted in failure 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>TicketsStarted</td>
<td>New matchmaking requests 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>TicketsTimedOut</td>
<td>Matchmaking requests that reached the timeout limit 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>TimeToMatch</td>
<td>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.</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>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>
</tbody>
</table>
### GameLift metrics for FleetIQ

The `GameLift` namespace includes metrics for FleetIQ game server group and game server activity as part of a FleetIQ standalone solution for game hosting. The GameLift service sends metrics to CloudWatch every minute. Also see Monitoring Your Auto Scaling Groups and Instances Using Amazon CloudWatch in the Amazon EC2 Auto Scaling User Guide.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AvailableGameServers</td>
<td>Game servers that are available to run a game execution and are not currently occupied with gameplay. This number includes game servers that have been claimed but are still in AVAILABLE status.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Sum</td>
</tr>
<tr>
<td></td>
<td>Dimensions: GameServerGroup</td>
</tr>
<tr>
<td>UtilizedGameServers</td>
<td>Game servers that are currently occupied with gameplay. This number includes game servers that are in UTILIZED status.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
</tbody>
</table>

### Matchmaking rules

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RuleEvaluationsPassed</td>
<td>Rule evaluations during the matchmaking process that passed since the last report. This metric is limited to the top 50 rules.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Sum</td>
</tr>
<tr>
<td>RuleEvaluationsFailed</td>
<td>Rule evaluations during matchmaking that failed since the last report. This metric is limited to the top 50 rules.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Sum</td>
</tr>
<tr>
<td>Metric</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DrainingAvailableGameServers</td>
<td>Game servers on instances scheduled for termination that are currently not supporting gameplay. These game servers are the lowest priority to be claimed in response to a new claim request.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Sum</td>
</tr>
<tr>
<td></td>
<td>Dimensions: GameServerGroup</td>
</tr>
<tr>
<td>DrainingUtilizedGameServers</td>
<td>Game servers on instances scheduled for termination that are currently supporting gameplay.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Sum</td>
</tr>
<tr>
<td></td>
<td>Dimensions: GameServerGroup</td>
</tr>
<tr>
<td>PercentUtilizedGameServers</td>
<td>Portion of game servers that are currently supporting game executions. This metric indicates the amount of game server capacity that is currently in use. It is useful for driving an Auto Scaling policy that can dynamically add and remove instances to match with player demand.</td>
</tr>
<tr>
<td></td>
<td>Units: Percent</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Average, Minimum, Maximum</td>
</tr>
<tr>
<td></td>
<td>Dimensions: GameServerGroup</td>
</tr>
<tr>
<td>GameServerInterruptions</td>
<td>Game servers on Spot Instances that were interrupted due to limited Spot availability.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Sum</td>
</tr>
<tr>
<td></td>
<td>Dimensions: GameServerGroup, InstanceType</td>
</tr>
<tr>
<td>InstanceInterruptions</td>
<td>Spot Instances that were interrupted due to limited availability.</td>
</tr>
<tr>
<td></td>
<td>Units: Count</td>
</tr>
<tr>
<td></td>
<td>Relevant CloudWatch statistics: Sum</td>
</tr>
<tr>
<td></td>
<td>Dimensions: GameServerGroup, InstanceType</td>
</tr>
</tbody>
</table>
Logging Amazon GameLift API Calls with AWS CloudTrail

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.

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.

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.

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.

```json
{
   "Records": [
      {
         "eventVersion": "1.04",
         "userIdentity": {
            "type": "IAMUser",
            "principalId": "AIDACKVECSQ6C2EXAMPLE",
            "arn": "arn:aws:iam::123456789012:user/AmazonGameLiftUser",
            "accountId": "123456789012",
            "resourceAccount": "123456789012",
            "invokingAccount": "123456789012",
            "service": "AmazonGameLift",
            "type": "AWS User"
         },
         "eventSource": "amazon_game_lift",
         "eventTime": "2019-01-01T00:00:00Z",
         "requestId": "123456789012",
         "messageId": "123456789012",
         "detailType": "TaskCompleted",
         "eventCategory": "GameLift",
         "responseElements": {
            "CreateFleetResponse": {
               "createFleetResult": {
                  "fleetId": "123456789012",
                  "tags": {
                     "env": "test",
                     "cluster": "abc"
                  }
               }
            }
         },
         "protocols": ["http", "https"],
         "sourceIPAddress": "192.0.2.1",
         "userAgent": "Amazon GameLift/1.2.3.4.5",
         "customerRequestToken": "123456789012",
         "awsRegion": "us-west-2",
         "eventBusName": "default",
         "awsAccountId": "123456789012",
         "eventName": "CreateFleet",
         "lambdaExecutionRoleArn": "arn:aws:iam::123456789012:role/AmazonGameLiftLambdaExecutionRole",
         "invokeResult": {
            "invocationId": "123456789012",
            "result": "SUCCESS",
            "error": null,
            "message": null
         }
      }
   ]
}```
"arn": "arn:aws:iam::111122223333:user/myUserName",
"accountId": "111122223333",
"accessKeyId": "AKIAIOSFODNN7EXAMPLE",
"userName": "myUserName"
},
"eventTime": "2015-12-29T23:40:15Z",
"eventSource": "gamelift.amazonaws.com",
"eventName": "CreateFleet",
"awsRegion": "us-west-2",
"sourceIPAddress": "192.0.2.0",
"userAgent": "[]",
"requestParameters": {
  "buildId": "build-92b6e8af-37a2-4c10-93bd-4698ea23de8d",
  "eC2InboundPermissions": [
    {
      "ipRange": "10.24.34.0/23",
      "fromPort": 1935,
      "protocol": "TCP",
      "toPort": 1935
    }
  ],
  "logPaths": [
    "C:\game\serverErr.log",
    "C:\game\serverOut.log"
  ],
  "eC2InstanceType": "c5.large",
  "serverLaunchPath": "C:\game\MyServer.exe",
  "description": "Test fleet",
  "serverLaunchParameters": "-paramX=baz",
  "name": "My_Test_Server_Fleet"
},
"responseElements": {
  "fleetAttributes": {
    "fleetId": "fleet-0bb84136-4f69-4bb2-bfec-a9b9a7c3d52e",
    "serverLaunchPath": "C:\game\MyServer.exe",
    "status": "NEW",
    "logPaths": [
      "C:\game\serverErr.log",
      "C:\game\serverOut.log"
    ],
    "description": "Test fleet",
    "serverLaunchParameters": "-paramX=baz",
    "creationTime": "Dec 29, 2015 11:40:14 PM",
    "name": "My_Test_Server_Fleet",
    "buildId": "build-92b6e8af-37a2-4c10-93bd-4698ea23de8d"
  }
},
"requestID": "824a2a4b-ae85-11e5-88d6-6d5ca9b525f2",
"eventID": "c6b7f9a1-5b9a-4c10-93bd-4698ea23de8d",
"eventType": "AwsApiCall",
"recipientAccountId": "111122223333"
},
{
  "eventVersion": "1.04",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::111122223333:user/myUserName",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "myUserName"
  },
  "eventTime": "2015-12-29T23:40:15Z",
  "eventSource": "gamelift.amazonaws.com",
  "eventName": "DescribeFleetAttributes",
  "awsRegion": "us-west-2",
  "version": "184"
"sourceIPAddress": "192.0.2.0",
"userAgent": "[]",
"requestParameters": {
"fleetIds": [
  "fleet-0bb84136-4f69-4bb2-bfbc-a9b9a7c3d52e"
]
},
"responseElements": null,
"requestID": "82e7f0ec-ae05-e1e5-a8d6-61d5cafb25f2",
"eventId": "11daabcb-0094-49f2-8b3d-3a63c8bad86f",
"eventType": "AwsApiCall",
"recipientAccountId": "111122223333"}
Security in Amazon GameLift

If you're using GameLift FleetIQ as a standalone feature with Amazon EC2, also refer to Security in Amazon EC2 in the Amazon EC2 User Guide for Linux Instances.

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to Amazon GameLift, see AWS Services in Scope by Compliance Program.

- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company’s requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using GameLift. The following topics show you how to configure GameLift to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your GameLift resources.

**Topics**
- Data Protection in GameLift (p. 186)
- Identity and access management for GameLift (p. 188)
- Logging and Monitoring with GameLift (p. 200)
- Compliance validation for GameLift (p. 201)
- Resilience in GameLift (p. 201)
- Infrastructure security in GameLift (p. 202)
- Configuration and vulnerability analysis in GameLift (p. 202)
- Security Best Practices for Amazon GameLift (p. 203)

Data Protection in GameLift

If you're using GameLift FleetIQ as a standalone feature with Amazon EC2, also refer to Security in Amazon EC2 in the Amazon EC2 User Guide for Linux Instances.

The AWS shared responsibility model applies to data protection in Amazon GameLift. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:
• Use multi-factor authentication (MFA) with each account.
• Use SSL/TLS to communicate with AWS resources. We recommend TLS 1.2 or later.
• Set up API and user activity logging with AWS CloudTrail.
• Use AWS encryption solutions, along with all default security controls within AWS services.
• Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.
• If you require FIPS 140-2 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers’ email addresses, into tags or free-form fields such as a Name field. This includes when you work with GameLift or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

GameLift-specific data is handled as follows:

• Game server builds and scripts that you upload to GameLift are stored in Amazon S3. There is no direct customer access to this data once it is uploaded. An authorized user can get temporary access to upload files, but can’t view or update the files in Amazon S3 directly. To delete scripts and builds, use the GameLift console or the service API.
• Game session log data is stored in Amazon S3 for a limited period of time after the game session is completed. Authorized users can access the log data by downloading it via a link in the GameLift console or by calls to the service API.
• Metric and event data is stored in GameLift and can be accessed through the GameLift console or by calls to the service API. Data can be retrieved on fleets, instances, game session placements, matchmaking tickets, game sessions, and player sessions. Data can also be accessed through Amazon CloudWatch and CloudWatch Events.
• Customer-supplied data is stored in GameLift. Authorized users can access it by calls to the service API. Potentially sensitive data might include player data, player session and game session data (including connection info), matchmaker data, and so on.

  **Note**
  If you provide custom player IDs in your requests, it is expected that these values are anonymized UUIDs and contain no identifying player information.

For more information about data protection, see the AWS Shared Responsibility Model and GDPR blog post on the AWS Security Blog.

### Encryption at rest

At-rest encryption of GameLift-specific data is handled as follows:

• Game server builds and scripts are stored in Amazon S3 buckets with server-side encryption.
• Customer-supplied data is stored in GameLift in an encrypted format.

### Encryption in Transit

Connections to the GameLift APIs are made over a secure (SSL) connection and authenticated using AWS Signature Version 4 (when connecting through the AWS CLI or AWS SDK, signing is handled
Internetwork traffic privacy

You can remotely access your GameLift instances securely. For instances that use Linux, SSH provides a secure communications channel for remote access. For instances that are running Windows, use a remote desktop protocol (RDP) client. With GameLift FleetIQ, remote access to your instances using AWS Systems Manager Session Manager and Run Command is encrypted using TLS 1.2, and requests to create a connection are signed using SigV4. For help with connecting to a managed GameLift instance, see Remotely access GameLift fleet instances (p. 126).

Identity and access management for GameLift

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be authenticated (signed in) and authorized (have permissions) to use GameLift resources. IAM is an AWS service that you can use with no additional charge.

Topics
- Audience (p. 188)
- Authenticating with identities (p. 189)
- Managing access using policies (p. 190)
- How GameLift works with IAM (p. 192)
- GameLift identity-based policy examples (p. 195)
- Troubleshooting GameLift identity and access (p. 199)

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in GameLift.

Service user – If you use the GameLift service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more GameLift features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in GameLift, see Troubleshooting GameLift identity and access (p. 199).

Service administrator – If you're in charge of GameLift resources at your company, you probably have full access to GameLift. It's your job to determine which GameLift features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of
your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with GameLift, see How GameLift works with IAM (p. 192).

IAM administrator – If you’re an IAM administrator, you might want to learn details about how you can write policies to manage access to GameLift. To view example GameLift identity-based policies that you can use in IAM, see GameLift identity-based policy examples (p. 195).

## Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. For more information about signing in using the AWS Management Console, see Signing in to the AWS Management Console as an IAM user or root user in the IAM User Guide.

You must be authenticated (signed in to AWS) as the AWS account root user, an IAM user, or by assuming an IAM role. You can also use your company’s single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access AWS using credentials from another company, you are assuming a role indirectly.

To sign in directly to the AWS Management Console, use your password with your root user email address or your IAM user name. You can access AWS programmatically using your root user or IAM users access keys. AWS provides SDK and command line tools to cryptographically sign your request using your credentials. If you don’t use AWS tools, you must sign the request yourself. Do this using Signature Version 4, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 signing process in the AWS General Reference.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, AWS recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see Using multi-factor authentication (MFA) in AWS in the IAM User Guide.

### AWS account root user

When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account root user and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

### IAM users and groups

An IAM user is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing access keys for IAM users in the IAM User Guide. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An IAM group is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named IAMAdmins and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the IAM User Guide.
IAM roles

An IAM role is an identity within your AWS account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the AWS Management Console by switching roles. You can assume a role by calling an AWS CLI or AWS API operation or by using a custom URL. For more information about methods for using roles, see Using IAM roles in the IAM User Guide.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as federated users. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the IAM User Guide.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some AWS services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the IAM User Guide.
- **Cross-service access** – Some AWS services use features in other AWS services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
- **Principal permissions** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see Actions, Resources, and Condition Keys for Amazon GameLift in the Service Authorization Reference.
- **Service role** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the IAM User Guide.
- **Service-linked role** – A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the IAM User Guide.

To learn whether to use IAM roles or IAM users, see When to create an IAM role (instead of a user) in the IAM User Guide.

Managing access using policies

You control access in AWS by creating policies and attaching them to IAM identities or AWS resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions.
You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, AWS evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in AWS as JSON documents. For more information about the structure and contents of JSON policy documents, see Overview of JSON policies in the IAM User Guide.

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the AWS Management Console, the AWS CLI, or the AWS API.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the IAM User Guide.

Identity-based policies can be further categorized as **inline policies** or **managed policies**. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your AWS account. Managed policies include AWS managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see Choosing between managed policies and inline policies in the IAM User Guide.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM role trust policies and Amazon S3 bucket policies. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

Resource-based policies are inline policies that are located in that service. You can’t use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the Amazon Simple Storage Service Developer Guide.

Other policy types

AWS supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.
• **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the Principal field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see Permissions boundaries for IAM entities in the IAM User Guide.

• **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in AWS Organizations. AWS Organizations is a service for grouping and centrally managing multiple AWS accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each AWS account root user. For more information about Organizations and SCPs, see How SCPs work in the AWS Organizations User Guide.

• **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see Session policies in the IAM User Guide.

### Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the IAM User Guide.

### How GameLift works with IAM

Before you use IAM to manage access to GameLift, you should understand what IAM features are available to use with GameLift. To get a high-level view of how GameLift and other AWS services work with IAM, see AWS Services That Work with IAM in the IAM User Guide.

If you're using GameLift FleetIQ as a standalone feature with Amazon EC2, also refer to Security in Amazon EC2 in the Amazon EC2 User Guide for Linux Instances.

### GameLift identity-based policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. GameLift supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see IAM JSON Policy Elements Reference in the IAM User Guide.

### Actions

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

The Action element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated AWS API operation. There are some exceptions, such as permission-only actions that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called dependent actions.

Include actions in a policy to grant permissions to perform the associated operation.
Policy actions in GameLift use the following prefix before the action: `gamelift:`. For example, to grant permission to request a new game session with the GameLift `StartGameSessionPlacement` API operation, you include the `gamelift:StartGameSessionPlacement` action in their policy. Policy statements must include either an `Action` or `NotAction` element. GameLift defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [
   "gamelift:action1",
   "gamelift:action2"
]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "gamelift:Describe*"
```

To see a list of GameLift actions, see Actions Defined by Amazon GameLift in the IAM User Guide.

The following GameLift actions have additional permission dependencies:

- The VPC peering actions `CreateVpcPeeringAuthorization` and `CreateVpcPeeringConnection` require access to the relevant VPC. The VPC can be owned by the same AWS account that you're using to manage GameLift, or it can be owned by a different AWS account. For more information on VPC peering with GameLift, see VPC peering for GameLift (p. 156).

**Resources**

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

The Resource JSON policy element specifies the object or objects to which the action applies. Statements must include either a Resource or a NotResource element. As a best practice, specify a resource using its Amazon Resource Name (ARN). You can do this for actions that support a specific resource type, known as resource-level permissions.

For actions that don't support resource-level permissions, such as listing operations, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

Long-lived GameLift resources have ARN values, which allows the resources to have their access managed using IAM policies. The GameLift fleet resource has an ARN with the following syntax:

```
arn:${Partition}:gamelift:${Region}:${Account}:fleet/${FleetId}
```

For more information about the format of ARNs, see Amazon Resource Names (ARNs) and AWS Service Namespaces.

For example, to specify the `fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa` fleet in your statement, use the following ARN:

```
```
To specify all fleets that belong to a specific account, use the wildcard (*):

```
```

Some GameLift actions, such as those for creating resources, cannot be performed on a specific resource. In those cases, you must use the wildcard (*).

```
"Resource": "*"
```

To view a list of GameLift resource types with ARNs, see About GameLift hosting resources (p. 89). To learn with which actions you can specify the ARN of each resource, see Actions Defined by Amazon GameLift.

### Condition keys

Administrators can use AWS JSON policies to specify who has access to what. That is, which principal can perform actions on what resources, and under what conditions.

The Condition element (or Condition block) lets you specify conditions in which a statement is in effect. The Condition element is optional. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple Condition elements in a statement, or multiple keys in a single Condition element, AWS evaluates them using a logical AND operation. If you specify multiple values for a single condition key, AWS evaluates the condition using a logical OR operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see IAM policy elements: variables and tags in the IAM User Guide.

AWS supports global condition keys and service-specific condition keys. To see all AWS global condition keys, see AWS global condition context keys in the IAM User Guide.

GameLift supports using some global condition keys. To see all AWS global condition keys, see AWS Global Condition Context Keys in the IAM User Guide.

Many GameLift actions support the `aws:RequestedTag` condition key. To see a list of GameLift condition keys, see Condition Keys for Amazon GameLift in the IAM User Guide. To learn with which actions and resources you can use a condition key, see Actions Defined by Amazon GameLift.

### Examples

To view examples of GameLift identity-based policies, see GameLift identity-based policy examples (p. 195).

### GameLift resource-based policies

GameLift does not support resource-based policies.

### Authorization based on GameLift tags

You can attach tags to GameLift resources or pass tags in a request to GameLift. To control access based on tags, you provide tag information in the condition element of a policy using the `gamelift:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys. For more information about tagging GameLift resources, see TagResource() in the Amazon GameLift API Reference.
To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see View GameLift fleets based on tags (p. 198).

**GameLift IAM roles**

An IAM role is an entity within your AWS account that has specific permissions.

**Using temporary credentials with GameLift**

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling AWS STS API operations such as AssumeRole or GetFederationToken.

GameLift supports using temporary credentials.

**Service-linked roles**

Service-linked roles allow AWS services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view but not edit the permissions for service-linked roles.

GameLift does not support service-linked roles.

**Service roles**

This feature allows a service to assume a service role on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

GameLift supports the use of service roles for the following scenarios:

- Allow your GameLift-hosted game servers to access other AWS resources, such as an AWS Lambda function or an Amazon DynamoDB database. Because game servers are hosted on fleets that are managed by GameLift, you need a service role that gives GameLift limited access to your other AWS resources. For more information, see Communicate with other AWS resources from your fleets (p. 55).

**GameLift identity-based policy examples**

By default, IAM users and roles don't have permission to create or modify GameLift resources. They also can't perform tasks using the AWS Management Console, AWS CLI, or AWS API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see Creating Policies on the JSON Tab in the IAM User Guide.

**Topics**

- Policy best practices (p. 196)
- Using the GameLift console (p. 196)
- Allow users to view their own permissions (p. 196)
- Allow player access for game sessions (p. 197)
- Allow access to one GameLift queue (p. 197)
- View GameLift fleets based on tags (p. 198)
- Access a game build file in Amazon S3 (p. 198)
Policy best practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete GameLift resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started using AWS managed policies** – To start using GameLift quickly, use AWS managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by AWS. For more information, see Get started using permissions with AWS managed policies in the IAM User Guide.

- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see Grant least privilege in the IAM User Guide.

- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see Using multi-factor authentication (MFA) in AWS in the IAM User Guide.

- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see IAM JSON policy elements: Condition in the IAM User Guide.

Using the GameLift console

To access the GameLift console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the GameLift resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

To ensure that those entities can still use the GameLift console, add an inline policy to users and groups with the following policy syntax. For more information, see Adding Permissions to a User in the IAM User Guide. You don’t need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API, such as players using game clients. Instead, allow access to only the actions that match the API operation that you're trying to perform.

- Permissions required to use all GameLift console features: see inline policy syntax for administrators in IAM policy examples for GameLift (p. 18).

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
            ]
        }
    ]
}
```
Allow player access for game sessions

You want to grant access that is needed by game clients (or client services that manage requests from game clients) to create new game sessions and request player placement in an available game session. There are several ways that your game might accomplish this task, including using game session placements with queues, matchmaking, or manual placement. To view policy examples for these scenarios, see the inline policy syntax for players in IAM policy examples for GameLift (p. 18).

Allow access to one GameLift queue

In this example, you want to grant an IAM user in your AWS account access to one of your GameLift queues, gamesessionqueue/examplequeue123, including the ability to add, update, and delete queue destinations.

This policy grants permissions to the user for the following actions: gamelift:UpdateGameSessionQueue, gamelift:DeleteGameSessionQueue, and gamelift:DescribeGameSessionQueues. As shown, this policy uses the resource element to limit access to a single queue.
View GameLift fleets based on tags

You can use conditions in your identity-based policy to control access to GameLift resources based on tags. This example shows how you might create a policy that allows viewing a fleet. However, permission is granted only if the fleet tag `Owner` has the value of that user's user name. This policy also grants the permissions necessary to complete this action on the console.

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Sid": "ListFleetsInConsole",
           "Effect": "Allow",
           "Action": "gamelift:ListFleets",
           "Resource": "*"
       },
       {
           "Sid": "ViewFleetIfOwner",
           "Effect": "Allow",
           "Action": "gamelift:DescribeFleetAttributes",
           "Resource": "arn:aws:gamelift::*::*:fleet/*",
           "Condition": {
               "StringEquals": {
                   "gamelift:ResourceTag/Owner": "${aws:username}"}}
       }
   ]
}
```

You can attach this policy to the IAM users in your account. If a user named `richard-roe` attempts to view a GameLift fleet, the fleet must be tagged `Owner=richard-roe` or `owner=richard-roe`. Otherwise he is denied access. The condition tag key `Owner` matches both `Owner` and `owner` because condition key names are not case-sensitive. For more information, see IAM JSON Policy Elements: Condition in the IAM User Guide.

Access a game build file in Amazon S3

Once your game server has been integrated with GameLift, you must upload the build files to the managed GameLift service. You must grant permissions to GameLift to access the build files in Amazon S3. You can do this by attaching the following policy to an IAM role:

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

For more information about uploading GameLift game files, see Upload a custom server build to GameLift (p. 97).
Troubleshooting GameLift identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with GameLift and IAM.

Topics

- I am not authorized to perform an action in GameLift (p. 199)
- I am not authorized to perform iam:PassRole (p. 199)
- I want to view my access keys (p. 199)
- I'm an administrator and want to allow others to access GameLift (p. 200)
- I want to allow people outside of my AWS account to access my GameLift resources (p. 200)

I am not authorized to perform an action in GameLift

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the mateojackson IAM user tries to use the console to view details about a queue but does not have gamelift:DescribeGameSessionQueues permissions.

User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: gamelift:DescribeGameSessionQueues on resource: examplequeue123

In this case, Mateo asks his administrator to update his policies to allow him to access the examplequeue123 resource using the gamelift:DescribeGameSessionQueues action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the iam:PassRole action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to GameLift.

Some AWS services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named marymajor tries to use the console to perform an action in GameLift. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole

In this case, Mary asks her administrator to update her policies to allow her to perform the iam:PassRole action.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, AKIAIOSFODNN7EXAMPLE) and a secret access key (for example, wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.
**Important**
Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

**I’m an administrator and want to allow others to access GameLift**

To allow others to access GameLift, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access AWS. You must then attach a policy to the entity that grants them the correct permissions in GameLift.

To get started right away, see Creating your first IAM delegated user and group in the *IAM User Guide*.

**I want to allow people outside of my AWS account to access my GameLift resources**

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether GameLift supports these features, see How GameLift works with IAM (p. 192).
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

**Logging and Monitoring with GameLift**

Monitoring is an important part of maintaining the reliability, availability, and performance of GameLift and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs.

AWS and GameLift provide several tools for monitoring your game hosting resources and responding to potential incidents.

**Amazon CloudWatch Alarms**

Using Amazon CloudWatch alarms, you watch a single metric over a time period that you specify. If the metric exceeds a given threshold, a notification is sent to an Amazon SNS topic or AWS Auto Scaling policy. CloudWatch alarms are triggered when their state changes and is maintained for a specified
Compliance validation

GameLift is not in scope of any AWS compliance programs.

Third-party auditors assess the security and compliance of AWS services as part of multiple AWS compliance programs, such as SOC, PCI, FedRAMP, and HIPAA.

To learn whether GameLift or other AWS services are in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- **Security and Compliance Quick Start Guides** – These deployment guides discuss architectural considerations and provide steps for deploying baseline environments on AWS that are security and compliance focused.
- **Architecting for HIPAA Security and Compliance Whitepaper** – This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.

  **Note**
  Not all services are compliant with HIPAA.

- **AWS Compliance Resources** – This collection of workbooks and guides might apply to your industry and location.
- **Evaluating Resources with Rules** in the *AWS Config Developer Guide* – The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- **AWS Security Hub** – This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.
- **AWS Audit Manager** – This AWS service helps you continuously audit your AWS usage to simplify how you manage risk and compliance with regulations and industry standards.

Resilience in GameLift

If you're using GameLift FleetIQ as a standalone feature with Amazon EC2, also refer to Security in Amazon EC2 in the *Amazon EC2 User Guide for Linux Instances*.

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability
Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

In addition to the AWS global infrastructure, GameLift offers the following features to help support your data resiliency needs:

- **Multi-region queues** – GameLift game session queues are used to place new game sessions with available hosting resources. Queues that span multiple Regions are able to redirect game session placements in the event of a regional outage. For more information and best practices on creating game session queues, see Design a game session queue (p. 139).

- **Automatic capacity scaling** – Maintain the health and availability of your hosting resources by using GameLift scaling tools. These tools provide a range of options that let you adjust fleet capacity to fit the needs of your game and players. For more information on scaling, see Scaling GameLift hosting capacity (p. 129).

- **Distribution across instances** – GameLift distributes incoming traffic across multiple instances, depending on fleet size. As a best practice, games in production should have multiple instances to maintain availability in case an instance becomes unhealthy or unresponsive.

- **Amazon S3 storage** – Game server builds and scripts that are uploaded to GameLift are stored in Amazon S3 using the Standard storage class, which uses multiple data center replications to increase resilience. Game session logs are also stored in Amazon S3 using the Standard storage class.

## Infrastructure security in GameLift

If you’re using GameLift FleetIQ as a standalone feature with Amazon EC2, also refer to Security in Amazon EC2 in the Amazon EC2 User Guide for Linux Instances.

As a managed service, GameLift is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access GameLift through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

The GameLift service places all fleets into Amazon virtual private clouds (VPCs) so that each fleet exists in a logically isolated area in the AWS Cloud. You can use GameLift policies to control access from specific VPC endpoints or specific VPCs. Effectively, this isolates network access to a given GameLift resource from only the specific VPC within the AWS network. When you create a fleet, you specify a range of port numbers and IP addresses. These ranges limit how inbound traffic can access hosted game servers on a fleet VPC. Use standard security best practices when choosing fleet access settings.

## Configuration and vulnerability analysis in GameLift

If you're using GameLift FleetIQ as a standalone feature with Amazon EC2, also refer to Security in Amazon EC2 in the Amazon EC2 User Guide for Linux Instances.
Configuration and IT controls are a shared responsibility between AWS and you, our customer. For more information, see the AWS shared responsibility model. AWS handles basic security tasks like guest operating system (OS) and database patching, firewall configuration, and disaster recovery. These procedures have been reviewed and certified by the appropriate third parties. For more details, see the following resource: Amazon Web Services: Overview of Security Processes (whitepaper).

The following security best practices also address configuration and vulnerability analysis in GameLift:

- Customers are responsible for the management of software that is deployed to GameLift instances for game hosting. Specifically:
  - Customer-provided game server application software should be maintained, including updates and security patches. To update game server software, upload a new build to GameLift, create a new fleet for it, and redirect traffic to the new fleet.
  - The base Amazon Machine Image (AMI), which includes the operating system, is updated only when a new fleet is created. To patch, update, and secure the operating system and other applications that are part of the AMI, recycle fleets on a regular basis, regardless of game server updates.
  - Customers should consider regularly updating their games with the latest SDK versions, including the AWS SDK, the GameLift Server SDK, and the GameLift Client SDK for Realtime Servers.

## Security Best Practices for Amazon GameLift

If you’re using GameLift FleetIQ as a standalone feature with Amazon EC2, also refer to Security in Amazon EC2 in the Amazon EC2 User Guide for Linux Instances.

GameLift provides a number of security features to consider as you develop and implement your own security policies. The following best practices are general guidelines and don’t represent a complete security solution. Because these best practices might not be appropriate or sufficient for your environment, treat them as helpful considerations rather than prescriptions.

- Amazon Web Services: Overview of Security Processes (whitepaper)
- AWS Security Best Practices (whitepaper)
Amazon GameLift Reference Guides

This section contains reference documentation for using Amazon GameLift.

Topics
- GameLift service API reference (AWS SDK) (p. 204)
- Amazon GameLift Realtime Servers Reference (p. 208)
- Amazon GameLift Server SDK Reference (p. 224)
- Game session placement events (p. 264)

GameLift service API reference (AWS SDK)

This topic provides a task-based list of API operations for use with GameLift managed hosting solutions, including hosting for custom game servers and Realtime Servers. These operations are packaged into the AWS SDK in the aws.gamelift namespace. Download the AWS SDK or view the Amazon GameLift API reference documentation.

The API includes two sets of operations for managed game hosting:

- Set up and manage GameLift hosting resources (p. 204)
- Start game sessions and join players (p. 207)

The GameLift Service API also contains operations for use with other GameLift tools and solutions. For a list of FleetIQ APIs, see FleetIQ API actions. For a list of FlexMatch APIs for matchmaking, see FlexMatch API actions.

Set up and manage GameLift hosting resources

Call these operations to configure hosting resources for your game servers, scale capacity to meet player demand, access performance and utilization metrics, and more. These API operations are used with game servers that are hosted on GameLift, including Realtime Servers. You can use the GameLift Console for most resource management tasks, or you can make calls to the service using the AWS Command Line Interface (AWS CLI) tool or the AWS SDK.

Prepare game servers for deployment

Upload and configure your game's game server code in preparation for deployment and launching on hosting resources.

Manage custom game server builds

- upload-build – Upload build files from a local path and create a new GameLift build resource. This operation, available only as an AWS CLI command, is the most common method for uploading game server builds.
- CreateBuild – Create a new build using files stored in an Amazon S3 bucket.
- ListBuilds – Get a list of all builds uploaded to a GameLift region.
- DescribeBuild – Retrieve information associated with a build.
- UpdateBuild – Change build metadata, including build name and version.
Set up and manage GameLift hosting resources

- **DeleteBuild** – Remove a build from GameLift.

Manage Realtime Servers configuration scripts

- **CreateScript** – Upload JavaScript files and create a new GameLift script resource.
- **ListScripts** – Get a list of all Realtime scripts uploaded to a GameLift region.
- **DescribeScript** – Retrieve information associated with a Realtime script.
- **UpdateScript** – Change script metadata and upload revised script content.
- **DeleteScript** – Remove a Realtime script from GameLift.

Set up computing resources for hosting

Configure hosting resources and deploy them with your game server build or Realtime configuration script.

Create and manage fleets

- **CreateFleet** – Configure and deploy a new GameLift fleet of computing resources to run your game servers. Once deployed, game servers are automatically launched as configured and ready to host game sessions.
- **ListFleets** – Get a list of all fleets in a GameLift region.
- **DeleteFleet** – Terminate a fleet that is no longer running game servers or hosting players.
- **View / update fleet locations.**
  - **CreateFleetLocations** – Add remote locations to an existing fleet that supports multiple locations
  - **DescribeFleetLocationAttributes** – Get a list of all remote locations for a fleet and view the current status of each location.
  - **DeleteFleetLocations** – Remove remote locations from a fleet that supports multiple locations.
- **View / update fleet configurations.**
  - **DescribeFleetAttributes / UpdateFleetAttributes** – View or change a fleet's metadata and settings for game session protection and resource creation limits.
  - **DescribeFleetPortSettings / UpdateFleetPortSettings** – View or change the inbound permissions (IP address and port setting ranges) allowed for a fleet.
  - **DescribeRuntimeConfiguration / UpdateRuntimeConfiguration** – View or change what server processes (and how many) to run on each instance in a fleet.

Manage fleet capacity

- **DescribeEC2InstanceLimits** – Retrieve maximum number of instances allowed for the current AWS account and the current usage level.
- **DescribeFleetCapacity** – Retrieve the current capacity settings for a fleet's home Region.
- **DescribeFleetLocationCapacity** – Retrieve the current capacity settings for each location a multi-location fleet.
- **UpdateFleetCapacity** – Manually adjust capacity settings for a fleet.
- **Set up auto-scaling:**
  - **PutScalingPolicy** – Turn on target-based auto-scaling or create a custom auto-scaling policy, or update an existing policy.
  - **DescribeScalingPolicies** – Retrieve an existing auto-scaling policy.
  - **DeleteScalingPolicy** – Delete an auto-scaling policy and stop it from affecting a fleet's capacity.
  - **StartFleetActions** – Restart a fleet's auto-scaling policies.
  - **StopFleetActions** – Suspend a fleet's auto-scaling policies.
Monitor fleet activity.

- **DescribeFleetUtilization** – Retrieve statistics on the number of server processes, game sessions, and players that are currently active on a fleet.
- **DescribeFleetLocationUtilization** – Retrieve utilization statistics for each location in a multi-location fleet.
- **DescribeFleetEvents** – View logged events for a fleet during a specified time span.
- **DescribeGameSessions** – Retrieve game session metadata, including a game's running time and current player count.

Set up queues for optimal game session placement

Set up multi-fleet, multi-region queues to place game sessions with the best available hosting resources for cost, latency, and resiliency.

- **CreateGameSessionQueue** – Create a queue for use when processing requests for game session placements.
- **DescribeGameSessionQueues** – Retrieve game session queues defined in a GameLift region.
- **UpdateGameSessionQueue** – Change the configuration of a game session queue.
- **DeleteGameSessionQueue** – Remove a game session queue from the region.

Manage aliases

Use aliases to represent your fleets or create a terminal alternative destination. Aliases are useful when transitioning game activity from one fleet to another, such as during game server build updates.

- **CreateAlias** – Define a new alias and optionally assign it to a fleet.
- **ListAliases** – Get all fleet aliases defined in a GameLift region.
- **DescribeAlias** – Retrieve information on an existing alias.
- **UpdateAlias** – Change settings for an alias, such as redirecting it from one fleet to another.
- **DeleteAlias** – Remove an alias from the region.
- **ResolveAlias** – Get the fleet ID that a specified alias points to.

Access hosting instances

View information on individual instances in a fleet, or request remote access to a specified fleet instance for troubleshooting.

- **DescribeInstances** – Get information on each instance in a fleet, including instance ID, IP address, location, and status.
- **GetInstanceAccess** – Request access credentials needed to remotely connect to a specified instance in a fleet.

Set up VPC peering

Create and manage VPC peering connections between your GameLift hosting resources and other AWS resources.

- **CreateVpcPeeringAuthorization** – Authorize a peering connection to one of your VPCs.
- **DescribeVpcPeeringAuthorizations** – Retrieve valid peering connection authorizations.
• **DeleteVpcPeeringAuthorization** – Delete a peering connection authorization.
• **CreateVpcPeeringConnection** – Establish a peering connection between the VPC for a GameLift fleet and one of your VPCs.
• **DescribeVpcPeeringConnections** – Retrieve information on active or pending VPC peering connections with a GameLift fleet.
• **DeleteVpcPeeringConnection** – Delete a VPC peering connection with a GameLift fleet.

### Start game sessions and join players

Call these operations from your game client service to start new game sessions, get information on existing game sessions, and join players to game sessions. These operations are for use with custom game servers that are hosted on GameLift. If you're using Realtime Servers, manage game sessions using the [Realtime Servers Client API (C#) Reference](p. 208).

- **Start new game sessions for one or more players.**
  - **StartGameSessionPlacement** – Ask GameLift to find the best available hosting resources and start a new game session. This is the preferred method for creating new game sessions. It relies on game session queues to track hosting availability across multiple regions, and uses FleetIQ algorithms to prioritize placements based on player latency, hosting cost, location, etc.
  - **DescribeGameSessionPlacement** – Get details and status on a placement request.
  - **StopGameSessionPlacement** – Cancel a placement request.
  - **CreateGameSession** – Start a new, empty game session on a specific fleet location. This operation gives you greater control over where to start the game session, instead of using FleetIQ to evaluate placement options. You must add players to the new game session in a separate step. Available in GameLift Local.

- **Get players into existing game sessions.** Find running game sessions with available player slots and reserve them for new players.
  - **CreatePlayerSession** – Reserve an open slot for a player to join a game session. Available in GameLift Local.
  - **CreatePlayerSessions** – Reserve open slots for multiple players to join a game session. Available in GameLift Local.

- **Work with game session and player session data.** Manage information on game sessions and player sessions.
  - **SearchGameSessions** – Request a list of active game sessions based on a set of search criteria.
  - **DescribeGameSessions** – Retrieve metadata for specific game sessions, including length of time active and current player count. Available in GameLift Local.
  - **DescribeGameSessionDetails** – Retrieve metadata, including the game session protection setting, for one or more game sessions.
  - **DescribePlayerSessions** – Get details on player activity, including status, playing time, and player data. Available in GameLift Local.
  - **UpdateGameSession** – Change game session settings, such as maximum player count and join policy.
  - **GetGameSessionLogUrl** – Get the location of saved logs for a game session.

### Available programming languages

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

- **C++** ([SDK docs](Amazon GameLift))
- **Java** ([SDK docs](Amazon GameLift))
Amazon GameLift Realtime Servers Reference

This section contains reference documentation for the Amazon GameLift Realtime Servers SDK. It includes the Realtime Client API as well as guidance for configuring your Realtime Servers script.

**Topics**
- Realtime Servers Client API (C#) Reference (p. 208)
- Amazon GameLift Realtime Servers Script Reference (p. 218)

**Realtime Servers Client API (C#) Reference**

Use the Realtime Client API to prepare your multiplayer game clients for use with Amazon GameLift Realtime Servers. For more on the integration process, see Get Started with Realtime Servers (p. 41). The Client API contains a set of synchronous API calls and asynchronous callbacks that enable a game client to connect to a Realtime server and exchange messages and data with other game clients via the server.

This API is defined in the following libraries:

Client.cs
- Synchronous Actions (p. 208)
- Asynchronous Callbacks (p. 212)
- Data Types (p. 214)

**To set up the Realtime Client API**

1. Download the Amazon GameLift Realtime Client SDK.
2. Build the C# SDK libraries. Locate the solution file GameScaleLightweightClientSdkNet45.sln. See the README.md file for the C# Server SDK for minimum requirements and additional build options. In an IDE, load the solution file. To generate the SDK libraries, restore the NuGet packages and build the solution.
3. Add the Realtime Client libraries to your game client project.

**Realtime Servers Client API (C#) Reference: Actions**

This C# Realtime Client API reference can help you prepare your multiplayer game for use with Realtime Servers deployed on Amazon GameLift fleets. For details on the integration process, see Get Started with Realtime Servers (p. 41).

- Synchronous Actions
- Asynchronous Callbacks (p. 212)
- Data Types (p. 214)
Client()

Initializes a new client to communicate with the Realtime server and identifies the type of connection to use.

Syntax

```csharp
public Client(ClientConfiguration configuration)
```

Parameters

clientConfiguration

Configuration details specifying the client/server connection type. You can opt to call Client() without this parameter; however, this approach results in an unsecured connection by default.

Type: ClientConfiguration (p. 215)

Required: No

Return value

Returns an instance of the Realtime client for use with communicating with the Realtime server.

Connect()

Requests a connection to a server process that is hosting a game session.

Syntax

```csharp
public ConnectionStatus Connect(string endpoint, int remoteTcpPort, int listenPort,
ConnectionToken token)
```

Parameters

directory

DNS name or IP address of the game session to connect to. The endpoint is specified in a GameSession object, which is returned in response to a client call to the AWS SDK Amazon GameLift API actions StartGameSessionPlacement, CreateGameSession, or DescribeGameSessions.

Type: String

Required: Yes

remoteTcpPort

Port number for the TCP connection assigned to the game session. This information is specified in a GameSession object, which is returned in response to a StartGameSessionPlacement CreateGameSession, or DescribeGameSession request.

Type: Integer

Valid Values: 1900 to 2000.
**Required: Yes**

**listenPort**

Port number that the game client is listening on for messages sent using the UDP channel.

Type: Integer

Valid Values: 33400 to 33500.

Required: Yes

**token**

Optional information that identifies the requesting game client to the server process.

Type: ConnectionToken (p. 215)

Required: Yes

**Return value**

Returns a ConnectionStatus (p. 217) enum value indicating the client's connection status.

**Disconnect()**

When connected to a game session, disconnects the game client from the game session.

**Syntax**

```
public void Disconnect()
```

**Parameters**

This action has no parameters.

**Return value**

This method does not return anything.

**NewMessage()**

Creates a new message object with a specified operation code. Once a message object is returned, complete the message content by specifying a target, updating the delivery method, and adding a data payload as needed. Once completed, send the message using SendMessage().

**Syntax**

```
public RTMessage NewMessage(int opCode)
```

**Parameters**

**opCode**

Developer-defined operation code that identifies a game event or action, such as a player move or a server notification.

Type: Integer

Required: Yes
Return value

Returns an RTMessage (p. 215) object containing the specified operation code and default delivery method. The delivery intent parameter is set to FAST by default.

SendMessage()

Sends a message to a player or group using the delivery method specified.

Syntax

```
public void SendMessage(RTMessage message)
```

Parameters

**message**

Message object that specifies the target recipient, delivery method, and message content.

Type: RTMessage (p. 215)

Required: Yes

Return value

This method does not return anything.

JoinGroup()

Adds the player to the membership of a specified group. Groups can contain any of the players that are connected to the game. Once joined, the player receives all future messages sent to the group and can send messages to the entire group.

Syntax

```
public void JoinGroup(int targetGroup)
```

Parameters

**targetGroup**

Unique ID that identifies the group to add the player to. Group IDs are developer-defined.

Type: Integer

Required: Yes

Return value

This method does not return anything. Because this request is sent using the reliable (TCP) delivery method, a failed request triggers the callback OnError() (p. 213).

LeaveGroup()

Removes the player from the membership of a specified group. Once no longer in the group, the player does not receive messages sent to the group and cannot send messages to the entire group.
**Syntax**

```csharp
public void LeaveGroup(int targetGroup)
```

**Parameters**

**targetGroup**

Unique ID identifying the group to remove the player from. Group IDs are developer-defined.

Type: Integer  
Required: Yes

**Return value**

This method does not return anything. Because this request is sent using the reliable (TCP) delivery method, a failed request triggers the callback `OnError()` (p. 213).

**RequestGroupMembership()**

Requests that a list of players in the specified group be sent to the game client. Any player can request this information, regardless of whether they are a member of the group or not. In response to this request, the membership list is sent to the client via an `OnGroupMembershipUpdated()` (p. 214) callback.

**Syntax**

```csharp
public void RequestGroupMembership(int targetGroup)
```

**Parameters**

**targetGroup**

Unique ID identifying the group to get membership information for. Group IDs are developer-defined.

Type: Integer  
Required: Yes

**Return value**

This method does not return anything.

---

**Realtime Servers Client API (C#) Reference: Asynchronous Callbacks**

Use this C# Realtime Client API reference to help you prepare your multiplayer game for use with Realtime Servers deployed on Amazon GameLift fleets. For details on the integration process, see [Get Started with Realtime Servers](#) (p. 41).

- Synchronous Actions (p. 208)
- Asynchronous Callbacks
- Data Types (p. 214)
A game client needs to implement these callback methods to respond to events. The Realtime server invokes these callbacks to send game-related information to the game client. Callbacks for the same events can also be implemented with custom game logic in the Realtime server script. See Script Callbacks for Realtime Servers (p. 218).

Callback methods are defined in ClientEvents.cs.

**OnOpen()**

Invoked when the server process accepts the game client's connection request and opens a connection.

**Syntax**

```csharp
public void OnOpen()
```

**Parameters**

This method takes no parameters.

**Return value**

This method does not return anything.

**OnClose()**

Invoked when the server process terminates the connection with the game client, such as after a game session ends.

**Syntax**

```csharp
public void OnClose()
```

**Parameters**

This method takes no parameters.

**Return value**

This method does not return anything.

**OnError()**

Invoked when a failure occurs for a Realtime Client API request. This callback can be customized to handle a variety of connection errors.

**Syntax**

```csharp
private void OnError(byte[] args)
```

**Parameters**

This method takes no parameters.

**Return value**

This method does not return anything.
OnDataReceived()

Invoked when the game client receives a message from the Realtime server. This is the primary method by which messages and notifications are received by a game client.

Syntax

```csharp
public void OnDataReceived(DataReceivedEventArgs dataReceivedEventArgs)
```

Parameters

dataReceivedEventArgs

Information related to message activity.

Type: `DataReceivedEventArgs (p. 216)`

Required: Yes

Return value

This method does not return anything.

OnGroupMembershipUpdated()

Invoked when the membership for a group that the player belongs to has been updated. This callback is also invoked when a client calls `RequestGroupMembership`.

Syntax

```csharp
public void OnGroupMembershipUpdated(GroupMembershipEventArgs groupMembershipEventArgs)
```

Parameters

groupMembershipEventArgs

Information related to group membership activity.

Type: `GroupMembershipEventArgs (p. 217)`

Required: Yes

Return value

This method does not return anything.

Realtime Servers Client API (C#) Reference: Data Types

This C# Realtime Client API reference can help you prepare your multiplayer game for use with Realtime Servers deployed on Amazon GameLift fleets. For details on the integration process, see `Get Started with Realtime Servers (p. 41)`.

- Synchronous Actions (p. 208)
- Asynchronous Callbacks (p. 212)
- Data Types
ClientConfiguration

Information about how the game client connects to a Realtime server.

Contents

ConnectionType

Type of client/server connection to use, either secured or unsecured. If you don't specify a connection type, the default is unsecured.

Note

When connecting to a Realtime server on a secured fleet with a TLS certificate, you must use the value RT_OVER_WSS_DTLS_TLS12.

Type: A ConnectionType enum (p. 217) value.

Required: No

ConnectionToken

Information about the game client and/or player that is requesting a connection with a Realtime server.

Contents

playerSessionId

Unique ID issued by GameLift when a new player session is created. A player session ID is specified in a PlayerSession object, which is returned in response to a client call to the GameLift API actions StartGameSessionPlacement, CreateGameSession, DescribeGameSessionPlacement, or DescribePlayerSessions.

Type: String

Required: No

payload

Developer-defined information to be communicated to the Realtime server on connection. This includes any arbitrary data that might be used for a custom sign-in mechanism. For examples, a payload may provide authentication information to be processed by the Realtime server script before allowing a client to connect.

Type: byte array

Required: No

RTMessage

Content and delivery information for a message. A message must specify either a target player or a target group.

Contents

opCode

Developer-defined operation code that identifies a game event or action, such as a player move or a server notification. A message's Op code provides context for the data payload that is being provided. Messages that are created using NewMessage() already have the operation code set, but it can be changed at any time.
**targetPlayer**

Unique ID identifying the player who is the intended recipient of the message being sent. The target may be the server itself (using the server ID) or another player (using a player ID).

Type: Integer

Required: Yes

**targetGroup**

Unique ID identifying the group that is the intended recipient of the message being sent. Group IDs are developer defined.

Type: Integer

Required: No

**deliveryIntent**

Indicates whether to send the message using the reliable TCP connection or using the fast UDP channel. Messages created using `NewMessage()` (p. 210).

Type: DeliveryIntent enum

Valid values: FAST | RELIABLE

Required: Yes

**payload**

Message content. This information is structured as needed to be processed by the game client based on the accompanying operation code. It may contain game state data or other information that needs to be communicated between game clients or between a game client and the Realtime server.

Type: Byte array

Required: No

---

**DataReceivedEventArgs**

Data provided with an `OnDataReceived()` (p. 214) callback.

**Contents**

**sender**

Unique ID identifying the entity (player ID or server ID) who originated the message.

Type: Integer

Required: Yes

**opCode**

Developer-defined operation code that identifies a game event or action, such as a player move or a server notification. A message's Op code provides context for the data payload that is being provided.
data

Message content. This information is structured as needed to be processed by the game client based on the accompanying operation code. It may contain game state data or other information that needs to be communicated between game clients or between a game client and the Realtime server.

Type: Byte array
Required: No

GroupMembershipEventArgs

Data provided with an OnGroupMembershipUpdated() (p. 214) callback.

Contents

sender

Unique ID identifying the player who requested a group membership update.

Type: Integer
Required: Yes

opCode

Developer-defined operation code that identifies a game event or action.

Type: Integer
Required: Yes

groupId

Unique ID identifying the group that is the intended recipient of the message being sent. Group IDs are developer defined.

Type: Integer
Required: Yes

playerId

List of player IDs who are current members of the specified group.

Type: Integer array
Required: Yes

Enums

 Enums defined for the Realtime Client SDK are defined as follows:

ConnectionStatus

- CONNECTED – Game client is connected to the Realtime server with a TCP connection only. All messages regardless of delivery intent are sent via TCP.
• CONNECTED_SEND_FAST – Game client is connected to the Realtime server with a TCP and a UDP connection. However, the ability to receive messages via UDP is not yet verified; as a result, all messages sent to the game client use TCP.
• CONNECTED_SEND_AND_RECEIVE_FAST – Game client is connected to the Realtime server with a TCP and a UDP connection. The game client can send and receive messages using either TCP or UDP.
• CONNECTING – Game client has sent a connection request and the Realtime server is processing it.
• DISCONNECTED_CLIENT_CALL – Game client was disconnected from the Realtime server in response to a Disconnect() (p. 210) request from the game client.
• DISCONNECTED – Game client was disconnected from the Realtime server for a reason other than a client disconnect call.

**ConnectionType**

• RT_OVER_WSS_DTLS_TLS12 – Secure connection type.
  
  For use with Realtime servers that are running on a GameLift fleet with a TLS certificate generated. When using a secure connection, TCP traffic is encrypted using TLS 1.2, and UDP traffic is encrypted using DTLS 1.2.
• RT_OVER_WS_UDP_UNSECURED – Non-secure connection type.
• RT_OVER_WEBSOCKET – Non-secure connection type. This value is no longer preferred.

**DeliveryIntent**

• FAST – Delivered using a UDP channel.
• RELIABLE – Delivered using a TCP connection.

---

### Amazon GameLift Realtime Servers Script Reference

Use these resources to build out custom logic in your Realtime scripts.

**Topics**

- Script Callbacks for Realtime Servers (p. 218)
- Realtime Servers Interface (p. 220)

### Script Callbacks for Realtime Servers

You can provide custom logic to respond to events by implementing these callbacks in your Realtime script.

**init**

Initializes the Realtime server and receives a Realtime server interface.

**Syntax**

```plaintext
init(rtsession)
```

**onMessage**

Invoked when a received message is sent to the server.

**Syntax**

```plaintext
onMessage(gameMessage)
```
onHealthCheck
Invoked to set the status of the game session health. By default, health status is healthy (or true). This callback can be implemented to perform custom health checks and return a status.

Syntax

```
onHealthCheck()
```

onStartGameSession
Invoked when a new game session starts, with a game session object passed in.

Syntax

```
onStartGameSession(session)
```

onProcessTerminate
Invoked when the server process is being terminated by the Amazon GameLift service. This can act as a trigger to exit cleanly from the game session. There is no need to call processEnding().

Syntax

```
onProcessTerminate()
```

onPlayerConnect
Invoked when a player requests a connection and has passed initial validation.

Syntax

```
onPlayerConnect(connectMessage)
```

onPlayerAccepted
Invoked when a player connection is accepted.

Syntax

```
onPlayerAccepted(player)
```

onPlayerDisconnect
Invoked when a player disconnects from the game session, either by sending a disconnect request or by other means.

Syntax

```
onPlayerDisconnect(peerId)
```

onProcessStarted
Invoked when a server process is started. This callback allows the script to perform any custom tasks needed to prepare to host a game session.
Syntax

```python
onProcessStarted(args)
```

**onSendToPlayer**

Invoked when a message is received on the server from one player to be delivered to another player. This process runs before the message is delivered.

Syntax

```python
onSendToPlayer(gameMessage)
```

**onSendToGroup**

Invoked when a message is received on the server from one player to be delivered to a group. This process runs before the message is delivered.

Syntax

```python
onSendToGroup(gameMessage)
```

**onPlayerJoinGroup**

Invoked when a player sends a request to join a group.

Syntax

```python
onPlayerJoinGroup(groupId, peerId)
```

**onPlayerLeaveGroup**

Invoked when a player sends a request to leave a group.

Syntax

```python
onPlayerLeaveGroup(groupId, peerId)
```

### Realtime Servers Interface

When a Realtime script initializes, an interface to the Realtime server is returned. This topic describes the properties and methods available through the interface. Learn more about writing Realtime scripts and view a detailed script example in Creating a Realtime Script (p. 74).

The Realtime interface provides access to the following objects:

- `session`
- `player`
- `gameMessage`
- `configuration`

**Realtime Session object**

Use these methods to access server-related information and perform server-related actions.
getPlayers()

Retrieves a list of peer IDs for players that are currently connected to the game session. Returns an array of player objects.

Syntax

rtSession.getPlayers()

broadcastGroupMembershipUpdate()

Triggers delivery of an updated group membership list to player group. Specify which membership to broadcast (groupIdToBroadcast) and the group to receive the update (targetGroupId). Group IDs must be a positive integer or "-1" to indicate all groups. See Realtime Servers Script Example (p. 75) for an example of user-defined group IDs.

Syntax

rtSession.broadcastGroupMembershipUpdate(groupIdToBroadcast, targetGroupId)

g.getServerId()

Retrieves the server's unique peer ID identifier, which is used to route messages to the server.

Syntax

rtSession.getServerId()

getAllPlayersGroupId()

Retrieves the group ID for the default group that contains all players currently connected to the game session.

Syntax

rtSession.getAllPlayersGroupId()

processEnding()

Triggers the Realtime server to terminate the game server. This function must be called from the Realtime script to exit cleanly from a game session.

Syntax

rtSession.processEnding()

gGetGameSessionId()

Retrieves the unique ID of the game session currently running.

Syntax

rtSession.getGameSessionId()
getLogger()

Retrieves the interface for logging. Use this to log statements that will be captured in your game session logs. The logger supports use of "info", "warn", and "error" statements. For example:

```javascript
logger.info("<string>");
```

Syntax

```javascript
rtSession.getLogger()
```

sendMessage()

Sends a message, created using newTextGameMessage or newBinaryGameMessage, from the Realtime server to a player recipient using the UDP channel. Identify the recipient using the player's peer ID.

Syntax

```javascript
rtSession.sendMessage(gameMessage, targetPlayer)
```

sendGroupMessage()

Sends a message, created using newTextGameMessage or newBinaryGameMessage, from the Realtime server to all players in a player group using the UDP channel. Group IDs must be a positive integer or "-1" to indicate all groups. See Realtime Servers Script Example (p. 75) for an example of user-defined group IDs.

Syntax

```javascript
rtSession.sendGroupMessage(gameMessage, targetGroup)
```

sendReliableMessage()

Sends a message, created using newTextGameMessage or newBinaryGameMessage, from the Realtime server to a player recipient using the TCP channel. Identify the recipient using the player's peer ID.

Syntax

```javascript
rtSession.sendReliableMessage(gameMessage, targetPlayer)
```

sendReliableGroupMessage()

Sends a message, created using newTextGameMessage or newBinaryGameMessage, from the Realtime server to all players in a player group using the TCP channel. Group IDs which must be a positive integer or "-1" to indicate all groups. See Realtime Servers Script Example (p. 75) for an example of user-defined group IDs.

Syntax

```javascript
rtSession.sendReliableGroupMessage(gameMessage, targetGroup)
```

newTextGameMessage()

Creates a new message containing text, to be sent from the server to player recipients using the SendMessage functions. Message format is similar to the format used in the Realtime Client SDK (see RTMessage (p. 215)). Returns a gameMessage object.
Syntax

```javascript
tSession.newTextGameMessage(opcode, sender, payload)
```

**newBinaryGameMessage()**

Creates a new message containing binary data, to be sent from the server to player recipients using the `SendMessage` functions. Message format is similar to the format used in the Realtime Client SDK (see `RTMessage` (p. 215)). Returns a `gameMessage` object.

Syntax

```javascript
tSession.newBinaryGameMessage(opcode, sender, binaryPayload)
```

**Player object**

Access player-related information.

- `player.peerId`
  Unique ID that is assigned to a game client when it connects to the Realtime server and joined the game session.

- `player.playerSessionId`
  Player session ID that was referenced by the game client when it connected to the Realtime server and joined the game session.

**Game message object**

Use these methods to access messages that are received by the Realtime server. Messages received from game clients have the `RTMessage` (p. 215) structure.

- `getPayloadAsText()`
  Gets the game message payload as text.

Syntax

```javascript
gameMessage.getPayloadAsText()
```

- `gameMessage.opcode`
  Operation code contained in a message.

- `gameMessage.payload`
  Payload contained in a message. May be text or binary.

- `gameMessage.sender`
  Peer ID of the game client that sent a message.

- `gameMessage.reliable`
  Boolean indicating whether the message was sent via TCP (true) or UDP (false).
Configuration object
The configuration object can be used to override default configurations.

`configuration.maxPlayers`
The maximum number of client / server connections that can be accepted by RealTimeServers.
The default is 32.

`configuration.pingIntervalTime`
Time interval in milliseconds that server will attempt to send a ping to all connected clients to verify connections are healthy.
The default is 3000ms.

Amazon GameLift Server SDK Reference
This section contains reference documentation for the Amazon GameLift Server SDK. Use the Server SDK to integrate your custom game server with the Amazon GameLift service to start and manage game servers as needed.

Topics
- GameLift Server API reference for C++ (p. 224)
- GameLift Server API reference for C# (p. 239)
- GameLift Server API reference for Unreal Engine (p. 253)

GameLift Server API reference for C++
This GameLift C++ Server API reference can help you prepare your multiplayer game for use with GameLift. For details on the integration process, see Add GameLift to your game server (p. 51).

This API is defined in `GameLiftServerAPI.h`, `LogParameters.h`, and `ProcessParameters.h`.

- Actions (p. 224)
- Data types (p. 235)

GameLift Server API reference for C++: Actions
This GameLift C++ Server API reference can help you prepare your multiplayer game for use with GameLift. For details on the integration process, see Add GameLift to your game server (p. 51).

This API is defined in `GameLiftServerAPI.h`, `LogParameters.h`, and `ProcessParameters.h`.

- Actions
- Data types (p. 235)

AcceptPlayerSession()
Notifies the GameLift service that a player with the specified player session ID has connected to the server process and needs validation. 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, 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: No

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 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

```cpp
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.

```cpp
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

```cpp
DescribePlayerSessionsOutcome DescribePlayerSessions (
    const Aws::GameLift::Server::Model::DescribePlayerSessionsRequest
    &describePlayerSessionsRequest);
```

Parameters

describePlayerSessionsRequest

A `DescribePlayerSessionsRequest` (p. 235) 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 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, GameLift returns the first 10 player sessions records matching the request.

```cpp
// Set request parameters
Aws::GameLift::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::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

```cpp
Aws::StringOutcome GetGameSessionId();
```

### Parameters

This action has no parameters.

### Return value

If successful, returns the game session ID as an `Aws::StringOutcome` object. If not successful, returns an error message.

### Example

```cpp
Aws::GameLift::AwsStringOutcome sessionIdOutcome = Aws::GameLift::Server::GetGameSessionId();
```

### GetInstanceCertificate()

Retrieves the file location of a pem-encoded TLS certificate that is associated with the fleet and its instances. This certificate is generated when a new fleet is created with the certificate configuration set to `GENERATED`. Use this certificate to establish a secure connection with a game client and to encrypt client/server communication.

### Syntax

```cpp
GetInstanceCertificateOutcome GetInstanceCertificate();
```

### Parameters

This action has no parameters.

### Return value

If successful, returns a `GetInstanceCertificateOutcome` object containing the location of the fleet's TLS certificate file, which is stored on the instance. If not successful, returns an error message.

### Example

```cpp
Aws::GameLift::GetInstanceCertificateOutcome certificateOutcome = Aws::GameLift::Server::GetInstanceCertificate();
```

### GetSdkVersion()

Returns the current version number of the SDK in use.

### Syntax

```cpp
Aws::StringOutcome 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 =
  Aws::GameLift::Server::GetSdkVersion();
```

GetTerminationTime()

Returns the time that a server process is scheduled to be shut down, if a termination time is available. A server process takes this action after receiving an `onProcessTerminate()` callback from the GameLift service. GameLift may call `onProcessTerminate()` for the following reasons: (1) when the server process has reported poor health or has not responded to GameLift, (2) when terminating the instance during a scale-down event, or (3) when an instance is being terminated due to a Spot interruption (p. 110).

If the process has received an `onProcessTerminate()` callback, the value returned is the estimated termination time. If the process has not received an `onProcessTerminate()` callback, an error message is returned. Learn more about shutting down a server process (p. 54).

Syntax

```cpp
AwsLongOutcome GetTerminationTime();
```

Parameters

This action has no parameters.

Return value

If successful, returns the termination time as an `AwsLongOutcome` object. The value is the termination time, expressed in elapsed ticks since 0001 00:00:00. For example, the date time value 2020-09-13 12:26:40 -000Z is equal to 637355968000000000 ticks. If no termination time is available, returns an error message.

Example

```cpp
Aws::GameLift::AwsLongOutcome TermTimeOutcome =
  Aws::GameLift::Server::GetTerminationTime();
```

InitSDK()

Initializes the GameLift SDK. This method should be called on launch, before any other 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. 229).

Example

```cpp
Aws::GameLift::Server::InitSDKOutcome initOutcome =
    Aws::GameLift::Server::InitSDK();
```

ProcessEnding()

Notifies the GameLift service that the server process is shutting down. This method should be called after all other cleanup tasks, including shutting down all active game sessions. This method should exit with an exit code of 0; a non-zero exit code results in an event message that the process did not exit cleanly.

Once the method exits with a code of 0, you can terminate the process with a successful exit code. You can also exit the process with an error code. If you exit with an error code, the fleet event will indicated the process terminated abnormally (SERVER_PROCESS_TERMINATED_UNHEALTHY).

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();
if (outcome.Success)
    exit(0);  // exit with success
else  // otherwise, exit with error code
    exit(errorCode);
```

ProcessReady()

Notifies the GameLift service that the server process is ready to host game sessions. Call this method after successfully invoking InitSDK() (p. 228) and completing setup tasks that are required before the server process can host a game session. This method should be called only once per process.

This call is synchronous. To make an asynchronous call, use ProcessReadyAsync() (p. 231). See Initialize the server process (p. 52) for more details.

Syntax

```cpp
GenericOutcome ProcessReady(
    const Aws::GameLift::Server::ProcessParameters &processParameters);
```
Parameters

**processParameters**

A ProcessParameters (p. 237) object communicating the following information about the server process:

- Names of callback methods, implemented in the game server code, that the 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 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. 229) 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),
        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 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 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. 229). See Initialize the server process (p. 52) for more details.

**Syntax**

```cpp
GenericOutcomeCallable ProcessReadyAsync(
    const Aws::GameLift::Server::ProcessParameters &processParameters);
```

**Parameters**

**processParameters**

A `ProcessParameters` (p. 237) object communicating the following information about the server process:

- Names of callback methods, implemented in the game server code, that the 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 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)
```
```cpp
{
    // 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 GameLift service that a player with the specified player session ID has disconnected from the server process. In response, 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: No

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
Aws::GameLift::GenericOutcome disconnectOutcome =
    Aws::GameLift::Server::RemovePlayerSession(playerSessionId);
```

StartMatchBackfill()

Sends a request to find new players for open slots in a game session created with FlexMatch. See also the AWS SDK action `StartMatchBackfill()`. With this action, match backfill requests can be initiated by a game server process that is hosting the game session. Learn more about the FlexMatch backfill feature.

This action is asynchronous. If new players are successfully matched, the GameLift service delivers updated matchmaker data by invoking the callback function `OnUpdateGameSession()`.
A server process can have only one active match backfill request at a time. To send a new request, first call `StopMatchBackfill()` (p. 233) to cancel the original request.

**Syntax**

```cpp
StartMatchBackfillOutcome StartMatchBackfill(
    const Aws::GameLift::Server::Model::StartMatchBackfillRequest &startBackfillRequest);
```

**Parameters**

**StartMatchBackfillRequest**

A `StartMatchBackfillRequest` (p. 238) object that communicates the following information:

- A ticket ID to assign to the backfill request. This information is optional; if no ID is provided, 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::StartMatchBackfillOutcome backfillOutcome = 
    Aws::GameLift::Server::StartMatchBackfill(startBackfillRequest);

// Implement callback function for backfill
void Server::OnUpdateGameSession(Aws::GameLift::Server::Model::GameSession gameSession,
    Aws::GameLift::Server::Model::UpdateReason updateReason, std::string backfillTicketId) {
    // handle status messages
    // perform game-specific tasks to prep for newly matched players
}
```

**StopMatchBackfill()**

 Cancels an active match backfill request that was created with `StartMatchBackfill()` (p. 232). See also the AWS SDK action `StopMatchmaking()`. Learn more about the FlexMatch backfill feature.
Syntax

```cpp
GenericOutcome StopMatchBackfill(
    const Aws::GameLift::Server::Model::StopMatchBackfillRequest &stopBackfillRequest);
```

Parameters

**StopMatchBackfillRequest**

A `StopMatchBackfillRequest` object identifying the matchmaking ticket to cancel:

- ticket ID assigned to the backfill request being canceled
- matchmaker the backfill request was sent to
- game session associated with the backfill request

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```cpp
// 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 matchmaking configuration ARN");  //
    // from the game session matchmaking data

Aws::GameLift::GenericOutcome stopBackfillOutcome =
    Aws::GameLift::Server::StopMatchBackfillRequest(stopBackfillRequest);
```

**TerminateGameSession()**

*This method is deprecated with version 4.0.1. Instead, the server process should call ProcessEnding() (p. 229) after a game session has ended.*

Notifies the GameLift service that the server process has ended the current game session. This action is called when the server process will remain active and ready to host a new game session. It should be called only after your game session termination procedure is complete, because it signals to GameLift that the server process is immediately available to host a new game session.

This action is not called if the server process will be shut down after the game session stops. Instead, call `ProcessEnding()` to signal that both the game session and the server process are ending.

Syntax

```cpp
GenericOutcome TerminateGameSession();
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.
UpdatePlayerSessionCreationPolicy()

Updates the current game session’s ability to accept new player sessions. A game session can be set to either accept or deny all new player sessions. See also the AWS SDK action UpdateGameSession().

Syntax

```cpp
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);
```

GameLift Server API reference for C++: Data types

This GameLift C++ Server API reference can help you prepare your multiplayer game for use with GameLift. For details on the integration process, see Add GameLift to your game server (p. 51).

This API is defined in GameLiftServerAPI.h, LogParameters.h, and ProcessParameters.h.

• Actions (p. 224)
• 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: arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>. The value of <ID string> is either a custom ID string or (if one was specified when the game session was created) a generated string.

Type: String
Required: No

Limit

Maximum number of results to return. Use this parameter with NextToken to get results as a set of sequential pages. If a player session ID is specified, this parameter is ignored.

Type: Integer
Required: No

NextToken

Token indicating the start of the next sequential page of results. Use the token that is returned with 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. 61).

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
LogParameters

This data type is used to identify which files generated during a game session that you want GameLift to upload and store once the game session ends. This information is communicated to the GameLift service in a `ProcessReady()` call.

Contents

logPaths

Directory paths to game server log files that you want 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 GameLift service in a `ProcessReady()` call.

Contents

port

Port number the server process listens on for new player connections. The value must fall into the port range configured for any fleet deploying this game server build. This port number is included in game session and player session objects, which game sessions use when connecting to a server process.

Type: Integer

Required: Yes

logParameters

Object with a list of directory paths to game session log files.

Type: `Aws::GameLift::Server::LogParameters` (p. 237)

Required: No

onStartGameSession

Name of callback function that the GameLift service invokes to activate a new game session. GameLift calls this function in response to the client request `CreateGameSession`. The callback function passes a `GameSession` object (defined in the `GameLift Service API Reference`).

Type: `const std::function<void(Aws::GameLift::Model::GameSession)>& onStartGameSession`

Required: Yes
onProcessTerminate

Name of callback function that the GameLift service invokes to force the server process to shut down. After calling this function, GameLift waits five minutes for the server process to shut down and respond with a ProcessEnding() (p. 229) 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 GameLift service invokes to request a health status report from the server process. GameLift calls this function every 60 seconds. After calling this function 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 GameLift service invokes to pass an updated game session object to the server process. GameLift calls this function when a match backfill request has been processed in order to provide updated matchmaker data. It passes a GameSession object, a status update (updateReason), and the match backfill ticket ID.

Type: std::function<void(Aws::GameLift::Server::Model::UpdateGameSession)> onUpdateGameSession

Required: No

StartMatchBackfillRequest

This data type is used to send a matchmaking backfill request. The information is communicated to the GameLift service in a StartMatchBackfill() (p. 232) call.

Contents

GameSessionArn

Unique game session identifier. The API action GetGameSessionId() (p. 226) returns the identifier in ARN format.

Type: String

Required: Yes

MatchmakingConfigurationArn

Unique identifier, in the form of an ARN, for the matchmaker to use for this request. To find the matchmaker that was used to create the original game session, look in the game session object, in the matchmaker data property. Learn more about matchmaker data in Word with matchmaker data.

Type: String

Required: Yes

Players

A set of data representing all players who are currently in the game session. The matchmaker uses this information to search for new players who are good matches for the current players. See the
Amazon GameLift 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 GameLift service in a StopMatchBackfill() (p. 233) call.

Contents

GameSessionArn

Unique game session identifier associated with the request being canceled.

Type: String
Required: Yes

MatchmakingConfigurationArn

Unique identifier of the matchmaker this request was sent to.

Type: String
Required: Yes

TicketId

Unique identifier of the backfill request ticket to be canceled.

Type: String
Required: Yes

GameLift Server API reference for C#

This GameLift C# Server API reference can help you prepare your multiplayer game for use with GameLift. For details on the integration process, see Add GameLift to your game server (p. 51).

This API is defined in GameLiftServerAPI.cs, LogParameters.cs, and ProcessParameters.cs.

- Actions (p. 240)
GameLift Server API (C#) reference: Actions

This GameLift C# Server API reference can help you prepare your multiplayer game for use with GameLift. For details on the integration process, see Add GameLift to your game server (p. 51).

This API is defined in GameLiftServerAPI.cs, LogParameters.cs, and ProcessParameters.cs.

- **Data Types (p. 249)**

## AcceptPlayerSession()

Notifies the GameLift service that a player with the specified player session ID has connected to the server process and needs validation. 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, GameLift changes the status of the player slot from RESERVED to ACTIVE.

### Syntax

```csharp
GenericOutcome AcceptPlayerSession(String playerSessionId)
```

### Parameters

- **playerSessionId**

  Unique ID issued by GameLift when a new player session is created. A player session ID is specified in a `PlayerSession` object, which is returned in response to a client call to the GameLift API actions `StartGameSessionPlacement`, `CreateGameSession`, `DescribeGameSessionPlacement`, or `DescribePlayerSessions`.

  Type: String  
  Required: No

### 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.

```csharp
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 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

```
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

```
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 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, GameLift will return the first 10 player sessions records matching the request.

```csharp
// Set request parameters
var describePlayerSessionsRequest = new
    Amazon.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 =
    GameLiftServerAPI.DescribePlayerSessions(describePlayerSessionRequest);
```

GetGameSessionId()

Retrieves the ID of the game session currently being hosted by the server process, if the server process is active.

Syntax

```csharp
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

```csharp
var getGameSessionIdOutcome = GameLiftServerAPI.GetGameSessionId();
```

GetInstanceCertificate()

Retrieves the file location of a pem-encoded TLS certificate that is associated with the fleet and its instances. This certificate is generated when a new fleet is created with the certificate configuration set to `GENERATED`. Use this certificate to establish a secure connection with a game client and to encrypt client/server communication.

Syntax

```csharp
GetInstanceCertificateOutcome GetInstanceCertificate();
```

Parameters

This action has no parameters.
**Return value**

If successful, returns a `GetInstanceCertificateOutcome` object containing the location of the fleet's TLS certificate file, which is stored on the instance. If not successful, returns an error message.

**Example**

```csharp
var getInstanceCertificateOutcome = GameLiftServerAPI.GetInstanceCertificate();
```

**GetSdkVersion()**

Returns the current version number of the SDK built into the server process.

**Syntax**

```csharp
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**

```csharp
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 GameLift service. GameLift may call `onProcessTerminate()` for the following reasons: (1) for poor health (the server process has reported port health or has not responded to GameLift), (2) when terminating the instance during a scale-down event, or (3) when an instance is being terminated due to a spot-instance interruption (p. 110).

If the process has received an `onProcessTerminate()` callback, the value returned is the estimated termination time. If the process has not received an `onProcessTerminate()` callback, an error message is returned. Learn more about shutting down a server process (p. 54).

**Syntax**

```csharp
AwsDateTimeOutcome GetTerminationTime()
```

**Parameters**

This action has no parameters.

**Return value**

If successful, returns the termination time as an `AwsDateTimeOutcome` object. The value is the termination time, expressed in elapsed ticks since 0001 00:00:00. For example, the date time value
2020-09-13 12:26:40 -000Z is equal to 637355968000000000 ticks. If no termination time is available, returns an error message.

Example

```csharp
var getTerminationTimeOutcome = GameLiftServerAPI.GetTerminationTime();
```

**InitSDK()**

Initializes the GameLift SDK. This method should be called on launch, before any other GameLift-related initialization occurs.

**Syntax**

```csharp
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. 245).

**Example**

```csharp
var initSDKOutcome = GameLiftServerAPI.InitSDK();
```

**ProcessEnding()**

Notifies the GameLift service that the server process is shutting down. This method should be called after all other cleanup tasks, including shutting down all active game sessions. This method should exit with an exit code of 0; a non-zero exit code results in an event message that the process did not exit cleanly.

Once the method exits with a code of 0, you can terminate the process with a successful exit code. You can also exit the process with an error code. If you exit with an error code, the fleet event will indicated the process terminated abnormally (SERVER_PROCESS_TERMINATED_UNHEALTHY).

**Syntax**

```csharp
GenericOutcome ProcessEnding()
```

**Parameters**

This action has no parameters.

**Return value**

Returns a generic outcome consisting of success or failure with an error message.

**Example**

```csharp
var processEndingOutcome = GameLiftServerAPI.ProcessEnding();
```
if (processReadyOutcome.Success)
    Environment.Exit(0);
// otherwise, exit with error code
Environment.Exit(errorCode);

ProcessReady()

Notifies the GameLift service that the server process is ready to host game sessions. Call this method after successfully invoking InitSDK() (p. 244) and completing setup tasks that are required before the server process can host a game session. This method should be called only once per process.

Syntax

GenericOutcome ProcessReady(ProcessParameters processParameters)

Parameters

processParameters

A ProcessParameters (p. 251) object communicating the following information about the server process:

- Names of callback methods, implemented in the game server code, that the 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 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. 245) call and delegate function implementations.

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

var processReadyOutcome = GameLiftServerAPI.ProcessReady(processParams);

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

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

bool OnHealthCheck()
{
    bool isHealthy;
    // complete health evaluation within 60 seconds and set health
    return isHealthy;
}

RemovePlayerSession()

Notifies the GameLift service that a player with the specified player session ID has disconnected from the server process. In response, 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 GameLift when a new player session is created. A player session ID is specified in a PlayerSession object, which is returned in response to a client call to the GameLift API actions StartGameSessionPlacement, CreateGameSession, DescribeGameSessionPlacement, or DescribePlayerSessions.

Type: String

Required: No

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```csharp
Aws::GameLift::GenericOutcome disconnectOutcome =
    Aws::GameLift::Server::RemovePlayerSession(playerSessionId);
```

StartMatchBackfill()

Sends a request to find new players for open slots in a game session created with FlexMatch. See also the AWS SDK action StartMatchBackfill(). With this action, match backfill requests can be initiated by a game server process that is hosting the game session. Learn more about the FlexMatch backfill feature.

This action is asynchronous. If new players are successfully matched, the GameLift service delivers updated matchmaker data using the callback function OnUpdateGameSession().

Version
246
A server process can have only one active match backfill request at a time. To send a new request, first call `StopMatchBackfill()` (p. 247) to cancel the original request.

**Syntax**

```
StartMatchBackfillOutcome StartMatchBackfill (StartMatchBackfillRequest startBackfillRequest);
```

**Parameters**

**StartMatchBackfillRequest**

A `StartMatchBackfillRequest` (p. 252) object that communicates the following information:

- A ticket ID to assign to the backfill request. This information is optional; if no ID is provided, 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**

```csharp
// 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 matcher data for current players
    // get matchmakerData.Players
    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. 246). See also the AWS SDK action `StopMatching()`. Learn more about the FlexMatch backfill feature.
Syntax

```csharp
GenericOutcome StopMatchBackfill (StopMatchBackfillRequest stopBackfillRequest);
```

Parameters

StopMatchBackfillRequest

A `StopMatchBackfillRequest` object identifying the matchmaking ticket to cancel:
- ticket ID assigned to the backfill request being canceled
- matchmaker the backfill request was sent to
- game session associated with the backfill request

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```csharp
// 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 current game session
};
var stopBackfillOutcome = GameLiftServerAPI.StopMatchBackfillRequest(stopBackfillRequest);
```

TerminateGameSession()

This method is deprecated with version 4.0.1. Instead, the server process should call `ProcessEnding()` (p. 244) after a game session has ended.

Notifies the GameLift service that the server process has ended the current game session. This action is called when the server process will remain active and ready to host a new game session. It should be called only after your game session termination procedure is complete, because it signals to GameLift that the server process is immediately available to host a new game session.

This action is not called if the server process will be shut down after the game session stops. Instead, call `ProcessEnding()` (p. 244) to signal that both the game session and the server process are ending.

Syntax

```csharp
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.

```csharp
// game-specific tasks required to gracefully shut down a game session,
// such as notifying players, preserving game state data, and other cleanup
var terminateGameSessionOutcome = GameLiftServerAPI.TerminateGameSession();
var processReadyOutcome = GameLiftServerAPI.ProcessReady(processParams);
```

**UpdatePlayerSessionCreationPolicy()**

Updates the current game session's ability to accept new player sessions. A game session can be set to either accept or deny all new player sessions. (See also the `UpdateGameSession()` action in the *GameLift Service API Reference*).

**Syntax**

```csharp
GenericOutcome UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy newPlayerSessionPolicy)
```

**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.

```csharp
var updatePlayerSessionCreationPolicyOutcome = 
    GameLiftServerAPI.UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy.ACCEPT_ALL);
```

**GameLift Server API reference for C#: Data types**

This GameLift C# Server API reference can help you prepare your multiplayer game for use with GameLift. For details on the integration process, see *Add GameLift to your game server (p. 51)*.

This API is defined in `GameLiftServerAPI.cs`, `LogParameters.cs`, and `ProcessParameters.cs`.

- Actions (p. 240)
- Data types

Version

249
LogParameters

This data type is used to identify which files generated during a game session that you want GameLift to upload and store once the game session ends. This information is communicated to the GameLift service in a `ProcessReady()` (p. 245) call.

Contents

logPaths

List of directory paths to game server log files you want GameLift to store for future access. These files are generated by a server process during each game session; file paths and names are defined in your game server and stored in the root game build directory. 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:gamelist:<region>::gamesession/fleet-<fleet ID>/<ID string>`. The value of `<ID string>` is either a custom ID string (if one was specified when the game session was created) a generated string.

Type: String

Required: No

Limit

Maximum number of results to return. Use this parameter with `NextToken` to get results as a set of sequential pages. If a player session ID is specified, this parameter is ignored.

Type: Integer

Required: No

NextToken

Token indicating the start of the next sequential page of results. Use the token that is returned with a previous call to this action. To specify the start of the result set, do not specify a value. If a player session ID is specified, this parameter is ignored.

Type: String

Required: No
**PlayerId**

Unique identifier for a player. Player IDs are defined by the developer. See [Generate Player IDs](#) (p. 61).

Type: String

Required: No

**PlayerSessionId**

Unique identifier for a player session.

Type: String

Required: No

**PlayerSessionStatusFilter**

Player session status to filter results on. Possible player session statuses include the following:

- RESERVED – The player session request has been received, but the player has not yet connected to the server process and/or been validated.
- ACTIVE – The player has been validated by the server process and is currently connected.
- COMPLETED – The player connection has been dropped.
- TIMEDOUT – A player session request was received, but the player did not connect and/or was not validated within the time-out limit (60 seconds).

Type: String

Required: No

**ProcessParameters**

This data type contains the set of parameters sent to the GameLift service in a [ProcessReady()](#) call.

**Contents**

**port**

Port number the server process will listen on for new player connections. The value must fall into the port range configured for any fleet deploying this game server build. This port number is included in game session and player session objects, which game sessions use when connecting to a server process.

Type: Integer

Required: Yes

**logParameters**

Object with a list of directory paths to game session log files.

Type: [Aws::GameLift::Server::LogParameters](#) (p. 250)

Required: Yes

**onStartGameSession**

Name of callback function that the GameLift service invokes to activate a new game session. GameLift calls this function in response to the client request [CreateGameSession](#). The callback function takes a [GameSession](#) object (defined in the [GameLift Service API Reference](#)).
Type: `void OnStartGameSessionDelegate(GameSession gameSession)`

Required: Yes

**onProcessTerminate**

Name of callback function that the GameLift service invokes to force the server process to shut down. After calling this function, GameLift waits five minutes for the server process to shut down and respond with a `ProcessEnding()` (p. 244) call before it shuts down the server process.

Type: `void OnProcessTerminateDelegate()`

Required: Yes

**onHealthCheck**

Name of callback function that the GameLift service invokes to request a health status report from the server process. GameLift calls this function every 60 seconds. After calling this function GameLift waits 60 seconds for a response, and if none is received, records the server process as unhealthy.

Type: `bool OnHealthCheckDelegate()`

Required: Yes

**onUpdateGameSession**

Name of callback function that the GameLift service invokes to pass an updated game session object to the server process. GameLift calls this function when a match backfill request has been processed in order to provide updated matchmaker data. It passes a `GameSession` object, a status update (updateReason), and the match backfill ticket ID.

Type: `void OnUpdateGameSessionDelegate ( UpdateGameSession updateGameSession )`

Required: No

**StartMatchBackfillRequest**

This data type is used to send a matchmaking backfill request. The information is communicated to the GameLift service in a `StartMatchBackfill()` (p. 246) call.

**Contents**

**GameSessionArn**

Unique game session identifier. The API action `GetGameSessionId()` (p. 242) returns the identifier in ARN format.

Type: `String`

Required: Yes

**MatchmakingConfigurationArn**

Unique identifier, in the form of an ARN, for the matchmaker to use for this request. To find the matchmaker that was used to create the original game session, look in the game session object, in the matchmaker data property. Learn more about matchmaker data in [Work with matchmaker data](#).

Type: `String`

Required: Yes
Players
A set of data representing all players who are currently in the game session. The matchmaker uses this information to search for new players who are good matches for the current players. See the Amazon GameLift 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 GameLift service in a StopMatchBackfill() (p. 247) call.

Contents
GameSessionArn
Unique game session identifier associated with the request being canceled.

Type: String
Required: Yes

MatchmakingConfigurationArn
Unique identifier of the matchmaker this request was sent to.

Type: String
Required: Yes

TicketId
Unique identifier of the backfill request ticket to be canceled.

Type: String
Required: Yes

GameLift Server API reference for Unreal Engine
This GameLift Server API reference can help you prepare your Unreal Engine game projects for use with GameLift. For details on the integration process, see Add GameLift to your game server (p. 51).

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. 45).

- Actions (p. 254)
- Data types (p. 260)

**GameLift Server API reference for Unreal Engine: Actions**

This GameLift Server API reference can help you prepare your Unreal Engine game projects for use with GameLift. For details on the integration process, see Add GameLift to your game server (p. 51).

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. 45).

- Actions
- Data types (p. 260)

**AcceptPlayerSession()**

Notifies the GameLift service that a player with the specified player session ID has connected to the server process and needs validation. 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, 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: No

**Return value**

Returns a generic outcome consisting of success or failure with an error message.

**ActivateGameSession()**

Notifies the 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.

DescribePlayerSessions()

Retrieves player session data, including settings, session metadata, and player data. Use this action to get information for a single player session, for all player sessions in a game session, or for all player sessions associated with a single player ID.

Syntax

FGameLiftDescribePlayerSessionsOutcome DescribePlayerSessions(const FGameLiftDescribePlayerSessionsRequest &describePlayerSessionsRequest)

Parameters

describePlayerSessionsRequest

A FDescribePlayerSessionsRequest (p. 260) object describing which player sessions to retrieve.

Required: Yes

Return value

If successful, returns a FDescribePlayerSessionsRequest (p. 260) object containing a set of player session objects that fit the request parameters. Player session objects have a structure identical to the AWS SDK GameLift API PlayerSession data type.

GetGameSessionId()

Retrieves the ID of the game session currently being hosted by the server process, if the server process is active.

Syntax

FGameLiftStringOutcome GetGameSessionId()

Parameters

This action has no parameters.

Return value

If successful, returns the game session ID as an FGameLiftStringOutcome object. If not successful, returns an error message.

GetInstanceCertificate()

Retrieves the file location of a pem-encoded TLS certificate that is associated with the fleet and its instances. This certificate is generated when a new fleet is created with the certificate configuration set to GENERATED. Use this certificate to establish a secure connection with a game client and to encrypt client/server communication.
Syntax

FGameLiftGetInstanceCertificateOutcome GetInstanceCertificate()

Parameters

This action has no parameters.

Return value

If successful, returns a FGameLiftGetInstanceCertificateOutcome object containing the location of the fleet's TLS certificate file, which is stored on the instance. 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 GameLift SDK. This method should be called on launch, before any other 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 GameLift service that the server process is shutting down. This method should be called after all other cleanup tasks, including shutting down all active game sessions. This method should exit
with an exit code of 0; a non-zero exit code results in an event message that the process did not exit cleanly.

Syntax

```cpp
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 GameLift service that the server process is ready to host game sessions. Call this method after successfully invoking `InitSDK()` (p. 256) and completing setup tasks that are required before the server process can host a game session. This method should be called only once per process.

Syntax

```cpp
FGameLiftGenericOutcome ProcessReady(FProcessParameters &processParameters)
```

**Parameters**

**FProcessParameters**

A `FProcessParameters` (p. 261) object communicating the following information about the server process:

- Names of callback methods, implemented in the game server code, that the 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 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. 47).

### RemovePlayerSession()

Notifies the GameLift service that a player with the specified player session ID has disconnected from the server process. In response, GameLift changes the player slot to available, which allows it to be assigned to a new player.

Syntax

```cpp
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: No

Return value

Returns a generic outcome consisting of success or failure with an error message.

StartMatchBackfill()

Sends a request to find new players for open slots in a game session created with FlexMatch. See also the AWS SDK action StartMatchBackfill(). With this action, match backfill requests can be initiated by a game server process that is hosting the game session. Learn more about the FlexMatch backfill feature.

This action is asynchronous. If new players are successfully matched, the 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. 258) to cancel the original request.

Syntax

FGameLiftStringOutcome StartMatchBackfill (FStartMatchBackfillRequest &startBackfillRequest);

Parameters

FStartMatchBackfillRequest

A FStartMatchBackfillRequest (p. 262) object that communicates the following information:
- A ticket ID to assign to the backfill request. This information is optional; if no ID is provided, 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

If successful, returns the match backfill ticket as a FGameLiftStringOutcome object. If not successful, returns an error message. Ticket status can be tracked using the AWS SDK action DescribeMatchmaking().

StopMatchBackfill()

 Cancels an active match backfill request that was created with StartMatchBackfill() (p. 258). See also the AWS SDK action StopMatchmaking). Learn more about the FlexMatch backfill feature.
Syntax

FGameLiftGenericOutcome StopMatchBackfill (FStopMatchBackfillRequest &stopBackfillRequest);

Parameters

StopMatchBackfillRequest

A FStopMatchBackfillRequest (p. 263) object identifying the matchmaking ticket to cancel:

- ticket ID assigned to the backfill request being canceled
- matchmaker the backfill request was sent to
- game session associated with the backfill request

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

TerminateGameSession()

This method is deprecated with version 4.0.1. Instead, the server process should call ProcessEnding() (p. 256) after a game session has ended.

Notifies the GameLift service that the server process has ended the current game session. This action is called when the server process will remain active and ready to host a new game session. It should be called only after your game session termination procedure is complete, because it signals to GameLift that the server process is immediately available to host a new game session.

This action is not called if the server process will be shut down after the game session stops. Instead, call ProcessEnding() (p. 256) to signal that both the game session and the server process are ending.

Syntax

FGameLiftGenericOutcome TerminateGameSession()

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

UpdatePlayerSessionCreationPolicy()

Updates the current game session's ability to accept new player sessions. A game session can be set to either accept or deny all new player sessions. (See also the UpdateGameSession() action in the GameLift Service API Reference).

Syntax

FGameLiftGenericOutcome UpdatePlayerSessionCreationPolicy(EPlayerSessionCreationPolicy policy)
Parameters

Policy

Value indicating whether the game session accepts new players.

Type: EPlayerSessionCreationPolicy enum. Valid values include:

• ACCEPT_ALL – Accept all new player sessions.
• DENY_ALL – Deny all new player sessions.

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

GameLift Server API reference for Unreal Engine: Data types

This GameLift Server API reference can help you prepare your Unreal Engine game projects for use with GameLift. For details on the integration process, see Add GameLift to your game server (p. 51).

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. 45).

• Actions (p. 254)
• Data types

FDescribePlayerSessionsRequest

This data type is used to specify which player session(s) to retrieve. You can use it as follows:

• Provide a PlayerSessionId to request a specific player session.
• Provide a GameSessionId to request all player sessions in the specified game session.
• Provide a PlayerId to request all player sessions for the specified player.

For large collections of player sessions, use the pagination parameters to retrieve results in sequential blocks.

Contents

GameSessionId

Unique game session identifier. Use this parameter to request all player sessions for the specified game session. Game session ID format is as follows:

arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>. The value of <ID string> is either a custom ID string or (if one was specified when the game session was created) a generated string.

Type: String

Required: No

Limit

Maximum number of results to return. Use this parameter with NextToken to get results as a set of sequential pages. If a player session ID is specified, this parameter is ignored.
**Type:** Integer  
**Required:** No

**NextToken**

Token indicating the start of the next sequential page of results. Use the token that is returned with a previous call to this action. To specify the start of the result set, do not specify a value. If a player session ID is specified, this parameter is ignored.

**Type:** String  
**Required:** No

**PlayerId**

Unique identifier for a player. Player IDs are defined by the developer. See [Generate Player IDs](p. 61).

**Type:** String  
**Required:** No

**PlayerSessionId**

Unique identifier for a player session.

**Type:** String  
**Required:** No

**PlayerSessionStatusFilter**

Player session status to filter results on. Possible player session statuses include the following:
- **RESERVED** – The player session request has been received, but the player has not yet connected to the server process and/or been validated.
- **ACTIVE** – The player has been validated by the server process and is currently connected.
- **COMPLETED** – The player connection has been dropped.
- **TIMEDOUT** – A player session request was received, but the player did not connect and/or was not validated within the time-out limit (60 seconds).

**Type:** String  
**Required:** No

**FProcessParameters**

This data type contains the set of parameters sent to the GameLift service in a [ProcessReady]() call.

**Contents**

**port**

Port number the server process will listen on for new player connections. The value must fall into the port range configured for any fleet deploying this game server build. This port number is included in game session and player session objects, which game sessions use when connecting to a server process.

**Type:** Integer
logParameters

Object with a list of directory paths to game session log files.

Type: TArray<FString>

onStartGameSession

Name of callback function that the GameLift service invokes to activate a new game session. GameLift calls this function in response to the client request CreateGameSession. The callback function takes a GameSession object (defined in the GameLift Service API Reference).

Type: FOnStartGameSession

onProcessTerminate

Name of callback function that the GameLift service invokes to force the server process to shut down. After calling this function, GameLift waits five minutes for the server process to shut down and respond with a ProcessEnding() (p. 256) call before it shuts down the server process.

Type: FSimpleDelegate

onHealthCheck

Name of callback function that the GameLift service invokes to request a health status report from the server process. GameLift calls this function every 60 seconds. After calling this function GameLift waits 60 seconds for a response, and if none is received, records the server process as unhealthy.

Type: FOnHealthCheck

onUpdateGameSession

Name of callback function that the GameLift service invokes to pass an updated game session object to the server process. GameLift calls this function when a match backfill request has been processed in order to provide updated matchmaker data. It passes a GameSession object, a status update (updateReason), and the match backfill ticket ID.

Type: FOnUpdateGameSession

FStartMatchBackfillRequest

This data type is used to send a matchmaking backfill request. The information is communicated to the GameLift service in a StartMatchBackfill() (p. 258) call.

Contents

GameSessionArn

Unique game session identifier. The API action GetGameSessionId() (p. 255) returns the identifier in ARN format.
Type: FString
Required: Yes

**MatchmakingConfigurationArn**

Unique identifier, in the form of an ARN, for the matchmaker to use for this request. To find the matchmaker that was used to create the original game session, look in the game session object, in the matchmaker data property. Learn more about matchmaker data in [Work with matchmaker data](#).

Type: FString
Required: Yes

**Players**

A set of data representing all players who are currently in the game session. The matchmaker uses this information to search for new players who are good matches for the current players. See the [Amazon GameLift API Reference Guide](#) for a description of the Player object format. To find player attributes, IDs, and team assignments, look in the game session object, in the matchmaker data property. If latency is used by the matchmaker, gather updated latency for the current region and include it in each player's data.

Type: TArray<FPlayer>
Required: Yes

**TicketId**

Unique identifier for a matchmaking or match backfill request ticket. If no value is provided here, Amazon GameLift will generate one in the form of a UUID. Use this identifier to track the match backfill ticket status or cancel the request if needed.

Type: FString
Required: No

### FStopMatchBackfillRequest

This data type is used to cancel a matchmaking backfill request. The information is communicated to the GameLift service in a StopMatchBackfill() (p. 258) call.

**Contents**

**GameSessionArn**

Unique game session identifier associated with the request being canceled.

Type: FString
Required: Yes

**MatchmakingConfigurationArn**

Unique identifier of the matchmaker this request was sent to.

Type: FString
Required: Yes

**TicketId**

Unique identifier of the backfill request ticket to be canceled.
Game session placement events

GameLift emits events for each game session placement request as it is processed. You can publish these events to an Amazon SNS topic, as described in Set up event notification for game session placement (p. 151). These events are also emitted to Amazon CloudWatch Events in near real time and on a best-effort basis.

This topic describes the structure of game session placement events and provides an example for each event type. For more information on the status of game session placement requests, see GameSessionPlacement in the Amazon GameLift API Reference.

Placement event syntax

Events are represented as JSON objects. Event structure conforms to the CloudWatch Events pattern, with similar top-level fields and service-specific details.

Top-level fields include the following (see event pattern for more detail):

- **version**
  - This field is always set to 0 (zero).
- **id**
  - Unique tracking identifier for the event.
- **detail-type**
  - Value is always GameLift Queue Placement Event.
- **source**
  - Value is always aws.gamelift.
- **account**
  - The AWS account that is being used to manage GameLift.
- **time**
  - Event timestamp.
- **region**
  - The AWS Region where the placement request is being processed. This is the Region where the game session queue in use resides.
- **resources**
  - ARN value of the game session queue that is processing the placement request.

**PlacementFulfilled**

The placement request has been successfully fulfilled. A new game session has been started and new player sessions have been created for each player listed in the game session placement request. Player connection information is available.
**Details syntax:**

- **placementId**
  - A unique identifier assigned to the game session placement request.
- **port**
  - The port number for the new game session.
- **gameSessionArn**
  - The ARN identifier for the new game session.
- **ipAddress**
  - The IP address of the game session.
- **dnsName**
  - The DNS identifier assigned to the instance that is running the new game session. The value format is different depending on whether the instance running the game session is TLS-enabled. When connecting to a game session on a TLS-enabled fleet, players must use the DNS name, not the IP address.
  - **TLS-enabled fleets:** `<unique identifier>.<region identifier>.amazongamelift.com`
  - **Non-TLS-enabled fleets:** `ec2-<unique identifier>.compute.amazonaws.com`
- **startTime**
  - Time stamp indicating when this request was placed in the queue.
- **endTime**
  - Time stamp indicating when this request was fulfilled.
- **gameSessionRegion**
  - AWS Region where the game session is being hosted. This information is also in the gameSessionArn value.
- **placedPlayerSessions**
  - The collection of player sessions that have been created for each player in the game session placement request.

**Example**

```json
{
  "version": "0",
  "id": "1111aaaa-bb22-cc33-dd44-5555eeeee66ff",
  "detail-type": "GameLift Queue Placement Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2021-03-01T15:50:52Z",
  "region": "us-east-1",
  "resources": [
  ],
  "detail": {
    "type": "PlacementFulfilled",
    "placementId": "9999ffff-88ee-77dd-66cc-5555bb44aa",
    "port": "6262",
    "gameSessionArn": "arn:aws:gamelift:us-west-2::gamesession/fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa/4444dddd-55ee-66ff-77aa-8888bbbb99cc",
    "version": "265"
  }
}``
PlacementCancelled

The placement request was canceled with a call to the GameLift service StopGameSessionPlacement.

Detail:

placementId

A unique identifier assigned to the game session placement request.

startTime

Time stamp indicating when this request was placed in the queue.

endTime

Time stamp indicating when this request was cancelled.

Example

```
{
  "version": "0",
  "id": "1111aaaa-bb22-cc33-dd44-5555eeee66ff",
  "detail-type": "GameLift Queue Placement Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2021-03-01T15:50:52Z",
  "region": "us-east-1",
  "resources": [
  ],
  "detail": {
    "type": "PlacementCancelled",
    "placementId": "9999ffff-88ee-77dd-66cc-5555bb44aa",
    "startTime": "2021-03-01T15:50:49.741Z",
    "endTime": "2021-03-01T15:50:52.084Z"
  }
}
```

PlacementTimedOut

Game session placement did not successfully complete before the queue's time limit expired. The placement request can be resubmitted as needed.

Detail:
placementId

A unique identifier assigned to the game session placement request.

startTime

Time stamp indicating when this request was placed in the queue.

endTime

Time stamp indicating when this request was cancelled.

Example

```
{
    "version": "0",
    "id": "111aaaaa-bb22-cc33-dd44-5555eeee66ff",
    "detail-type": "GameLift Queue Placement Event",
    "source": "aws.gamelift",
    "account": "123456789012",
    "time": "2021-03-01T15:50:52Z",
    "region": "us-east-1",
    "resources": [
    ],
    "detail": {
        "type": "PlacementTimedOut",
        "placementId": "9999ffff-88ee-77dd-66cc-5555bb44aa",
        "startTime": "2021-03-01T15:50:49.741Z",
        "endTime": "2021-03-01T15:50:52.084Z"
    }
}
```

PlacementFailed

GameLift was not able to fulfill the game session request. This is generally caused by an unexpected internal error. The placement request can be resubmitted as needed.

Detail:

placementId

A unique identifier assigned to the game session placement request.

startTime

Time stamp indicating when this request was placed in the queue.

endTime

Time stamp indicating when this request failed.

Example

```
{
    "version": "0",
    "id": "39c978f3-ba46-3f7c-e787-55bfcca1bd31",
    "detail-type": "GameLift Queue Placement Event",
    "source": "aws.gamelift",
```

Version

267
"account": "252386620677",
"time": "2021-03-01T15:50:52Z",
"region": "us-east-1",
"resources": [
],
"detail": {
   "type": "PlacementFailed",
   "placementId": "e4a1119a-39af-45cf-a990-ef150fe0d453",
   "startTime": "2021-03-01T15:50:49.741Z",
   "endTime": "2021-03-01T15:50:52.084Z"
}
}
GameLift release notes

The GameLift release notes provide details about new features, updates, and fixes related to the service. These may also involve changes to the following components:

- AWS SDK GameLift API and CLI commands
- GameLift Server SDK for use with managed GameLift and custom game servers
- GameLift Client SDK for use with Realtime Servers
- GameLift Console

SDK versions

The following tabs list all GameLift releases with SDK versions. There is no requirement to use comparable SDKs for your game server and client integrations, however older versions of one SDK may not fully support the latest features in another. For more details on GameLift SDKs, see GameLift SDKs (p. 22).

**Note**
If you are currently uploading a game server build or creating a build resource with the AWS SDK using Signature Version 2 for request authentication, you should move to using Signature Version 4. Support is ending for Signature Version 2, as described in AWS Signature Version 2 Turned Off for Amazon S3.

Current version

<table>
<thead>
<tr>
<th>Release:</th>
<th>AWS SDK version:</th>
<th>Server SDK version:</th>
<th>Realtime Client SDK version:</th>
<th>GameLift Local</th>
</tr>
</thead>
<tbody>
<tr>
<td>2021-06-03 (p. 272) 1.8.168 or later</td>
<td>4.0.2</td>
<td>3.4.0</td>
<td>1.2.0</td>
<td>1.0.5</td>
</tr>
</tbody>
</table>

Previous versions

<table>
<thead>
<tr>
<th>Release:</th>
<th>AWS SDK version:</th>
<th>Server SDK version:</th>
<th>Realtime Client SDK version:</th>
<th>GameLift Local</th>
</tr>
</thead>
<tbody>
<tr>
<td>2021-03-23 (p. 272) 1.8.168 or later</td>
<td>4.0.2</td>
<td>3.4.1</td>
<td>1.1.0</td>
<td>1.0.5</td>
</tr>
<tr>
<td>2021-03-16 (p. 273) 1.8.163 or later</td>
<td>4.0.2</td>
<td>3.4.1</td>
<td>1.1.0</td>
<td>1.0.5</td>
</tr>
<tr>
<td>2021-02-09 (p. 274) 1.8.139 or later</td>
<td>4.0.2</td>
<td>3.4.1</td>
<td>1.1.0</td>
<td>1.0.5</td>
</tr>
<tr>
<td>2020-12-22 (p. 275) 1.8.95 or later</td>
<td>4.0.2</td>
<td>3.4.1</td>
<td>1.1.0</td>
<td>1.0.5</td>
</tr>
<tr>
<td>2020-11-24 (p. 275) 1.8.95 or later</td>
<td>4.0.2</td>
<td>3.4.1</td>
<td>1.1.0</td>
<td>1.0.5</td>
</tr>
<tr>
<td>2020-11-11 (p. 277) 1.8.36 or later</td>
<td>4.0.2</td>
<td>3.4.1</td>
<td>1.1.0</td>
<td>1.0.5</td>
</tr>
<tr>
<td>2020-09-17 (p. 278) 1.8.36 or later</td>
<td>4.0.1</td>
<td>3.4.1</td>
<td>1.1.0</td>
<td>1.0.5</td>
</tr>
</tbody>
</table>
## Release notes

The following release notes are in chronological order, with the latest updates listed first. GameLift was first released in 2016. For release notes dated earlier than those listed here, see the release date links in SDK versions (p. 269).

<table>
<thead>
<tr>
<th>Release:</th>
<th>AWS SDK version:</th>
<th>Server SDK version:</th>
<th>Realtime Client SDK version:</th>
<th>GameLift Local</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020-08-27 (p. 279)</td>
<td>1.7.310 or later</td>
<td>4.0.0</td>
<td>3.3.1</td>
<td>1.1.0</td>
</tr>
<tr>
<td>2020-04-16 (p. 279)</td>
<td>1.7.310 or later</td>
<td>4.0.0</td>
<td>3.3.1</td>
<td>1.1.0</td>
</tr>
<tr>
<td>2020-04-02 (p. 279)</td>
<td>1.7.310 or later</td>
<td>3.4.0</td>
<td>3.3.1</td>
<td>1.1.0</td>
</tr>
<tr>
<td>2019-12-19 (p. 279)</td>
<td>1.7.249 or later</td>
<td>3.4.0</td>
<td>3.3.1</td>
<td>1.1.0</td>
</tr>
<tr>
<td>2019-11-14 (p. 280)</td>
<td>1.7.210 or later</td>
<td>3.4.0</td>
<td>3.3.1</td>
<td>1.1.0</td>
</tr>
<tr>
<td>2019-10-24</td>
<td>1.7.210 or later</td>
<td>3.4.0</td>
<td>3.3.1</td>
<td>1.1.0</td>
</tr>
<tr>
<td>2019-09-03</td>
<td>1.7.175 or later</td>
<td>3.4.0</td>
<td>3.3.1</td>
<td>1.1.0</td>
</tr>
<tr>
<td>2019-07-09</td>
<td>1.7.140 or later</td>
<td>3.3.0</td>
<td>3.3.0</td>
<td>1.0.0</td>
</tr>
<tr>
<td>2019-04-25</td>
<td>1.7.91 or later</td>
<td>3.3.0</td>
<td>3.3.0</td>
<td>1.0.0</td>
</tr>
<tr>
<td>2019-03-07</td>
<td>1.7.65 or later</td>
<td>3.3.0</td>
<td>3.3.0</td>
<td>1.0.0</td>
</tr>
<tr>
<td>2019-02-07</td>
<td>1.7.45 or later</td>
<td>3.3.0</td>
<td>3.3.0</td>
<td>1.0.0</td>
</tr>
<tr>
<td>2018-12-14</td>
<td>1.6.20 or later</td>
<td>3.3.0</td>
<td>3.3.0</td>
<td>1.0.0</td>
</tr>
<tr>
<td>2018-09-27</td>
<td>1.6.20 or later</td>
<td>3.2.1</td>
<td>3.2.1</td>
<td>1.0.0</td>
</tr>
<tr>
<td>2018-06-14</td>
<td>1.4.47 or later</td>
<td>3.2.1</td>
<td>3.2.1</td>
<td>1.0.0</td>
</tr>
<tr>
<td>2018-05-10</td>
<td>1.4.47 or later</td>
<td>3.2.1</td>
<td>3.2.1</td>
<td>1.0.0</td>
</tr>
<tr>
<td>2018-02-15</td>
<td>1.3.58 or later</td>
<td>3.2.1</td>
<td>3.2.1</td>
<td>1.0.0</td>
</tr>
<tr>
<td>2018-02-08</td>
<td>1.3.52 or later</td>
<td>3.2.0</td>
<td>3.2.0</td>
<td>1.0.0</td>
</tr>
<tr>
<td>2017-09-01</td>
<td>1.1.43 or later</td>
<td>3.1.7</td>
<td>3.1.7</td>
<td>1.0.0</td>
</tr>
<tr>
<td>2017-08-16</td>
<td>1.1.31 or later</td>
<td>3.1.7</td>
<td>3.1.7</td>
<td>1.0.0</td>
</tr>
<tr>
<td>2017-05-16</td>
<td>1.0.122 or later</td>
<td>3.1.5</td>
<td>3.1.5</td>
<td>1.0.0</td>
</tr>
<tr>
<td>2017-04-11</td>
<td>1.0.103 or later</td>
<td>3.1.5</td>
<td>3.1.5</td>
<td>1.0.0</td>
</tr>
<tr>
<td>2017-02-21</td>
<td>1.0.72 or later</td>
<td>3.1.5</td>
<td>3.1.5</td>
<td></td>
</tr>
<tr>
<td>2016-11-18</td>
<td>1.0.31 or later</td>
<td></td>
<td></td>
<td>3.1.0</td>
</tr>
<tr>
<td>2016-10-13</td>
<td>1.0.17 or later</td>
<td></td>
<td></td>
<td>3.1.0</td>
</tr>
<tr>
<td>2016-09-01</td>
<td>0.14.9 or later</td>
<td></td>
<td></td>
<td>3.1.0</td>
</tr>
<tr>
<td>2016-08-04</td>
<td>0.12.16 or later</td>
<td></td>
<td></td>
<td>3.0.7</td>
</tr>
</tbody>
</table>
September 20 2021: GameLift adds support for the Plug-in for Unity

Amazon GameLift now provides a Amazon GameLift Plug-in for Unity you can use

Release date: September 20, 2021

The Amazon GameLift Unity Plug-in version 1.0.0 contains libraries and native UI that makes it easier to access GameLift resources and integrate GameLift into your Unity game. You can use the GameLift Unity Plug-in to access GameLift APIs and deploy AWS CloudFormation templates for common gaming scenarios. The plug-in also includes a sample game that works with the sample scenarios. You can use GameLift Local to see messages passed between the game client and the game server to learn how a typical game interacts with GameLift.

The Plug-in for Unity supports Unity 2019.4 LTS and 2020.3 LTS.

Highlights:

- Learn the basics of integrating a game with GameLift with the sample game provided with the Plug-in for Unity. Build, run, and modify the sample with different scenarios — or create your own.
- Deploy sample AWS CloudFormation scenarios for typical game scenarios including auth only, single-region fleet, multi-region fleets with queue and custom matchmaker, SPOT fleets with queue and custom matchmaker, and FlexMatch.

Learn more:

- Integrating Games with the Amazon GameLift Plug-in for Unity

June 30 2021: FlexMatch adds batchDistance rule

Amazon GameLift now supports the new batchDistance rule type, making it easier to balance players by different attributes like skill or maps, create simpler syntax for common rules, and create expansions based on batch distance over time.

Release date: June 30, 2021

Added a new rule type, called a batchDistance rule, which allows for a string or numeric attribute to be specified, bringing a host of benefits to each segment.

When playing multiplayer games, players expect to enter sessions quickly and exit sessions with a sense of accomplishment. To achieve this thrill for players, a good matchmaker must group together players based on their attributes, including skill level, playstyle, and latency. Often times, players get grouped in either large matches (>40 players) or small matches (up to 40). The new batchDistance rule type can help.

Highlights:

- For large matches (>40 players), instead of evenly balancing players by only skill, you can now now get that same balance based on skill, modes, and maps. Ensure everyone in the match is in a skill band, band multiple numeric attributes like league or playstyle, and group according to string attributes like map or game mode. You can also create expansions over time. For example, you can create an expansion to allow a greater skill level range to enter the match the longer the player is waiting.

For matches under 40 players, you can use a new simplified rules expression.

Learn more:
June 3 2021: GameLift Realtime Client SDK and Server SDK Updates

Amazon GameLift now supports IL2CPP in the RTS Client SDK, building the Native libraries as Frameworks, supports DescribePlayerSessions and GetInstanceCertificate in the Unreal Plugin, and added Server SDK support for Unreal version 4.26.

Release date: June 3, 2021

Updated SDK versions: AWS SDK 1.8.168

With this latest SDK update, you now have the ability to integrate IL2CPP into your mobile applications that use the RTS Client SDK and follow best practices with Frameworks. You can also now build the Amazon GameLift Server SDK for Unreal Version 4.26. This update contains components that integrate with your Windows or Linux game server, including C++ and C# versions of the GameLift Server SDK, Amazon GameLift Local, and an Unreal Engine plugin.

Highlights:

- Added support in the RTS Client SDK for IL2CPP and building the Native libraries as Frameworks, allowing RTS clients to be built for the latest mobile devices.
- You can use DescribePlayerSessions() (p. 255) to get information for a single player session, for all player sessions in a game session, or for all player sessions associated with a single player ID.
- You can use GetInstanceCertificate() (p. 255) to retrieve the file location of a pem-encoded TLS certificate that is associated with the fleet and its instances.
- The existing C# SDK, version 4.0.2, has been verified compatible with Unity 2020.3. No SDK updates were required.

Learn more:

- GameLift Developer Guide:
  - DescribePlayerSessions() (p. 255)
  - GetInstanceCertificate() (p. 255)

March 23 2021: GameLift adds notifications to game session placement

Amazon GameLift now provides event notification for tracking game session placement requests.

Release date: March 23, 2021

Updated SDK versions: AWS SDK 1.8.168

You can now use events to monitor game session placement activity for a game session queue. Create an Amazon Simple Notification Service (Amazon SNS) topic to publish event notifications, or set up event tracking using CloudWatch Events.
Highlights:

- For each queue, you can set a custom text string to be included in all event messaging.
- When using an Amazon SNS topic, you can set additional access conditions that limit publishing to specific queues.

Learn more:

- GameLift Developer Guide:
  - Set up event notification for game session placement (p. 151) (new)
  - Game session placement events (p. 264) (new)
- API Reference (AWS SDK)
  - New game session queue parameters NotificationTarget and CustomEventData: GameSessionQueue, CreateGameSessionQueue, UpdateGameSessionQueue
- GameLift forum

March 16 2021: GameLift adds multi-region fleets, six new Regions

Amazon GameLift broadens global coverage for game hosting, adding six new AWS Regions and introducing multi-region fleets, which can manage game server deployment across multiple locations.

Release date: March 16, 2021

Updated SDK versions: AWS SDK 1.8.163

GameLift managed hosting is now available in 21 AWS Regions. The new Regions are Cape Town (af-south-1), Bahrain (me-south-1), Hong Kong (ap-east-1), Milan (eu-south-1), Paris (eu-west-3), and Stockholm (eu-north-1).

With the new GameLift multi-location fleets feature, you can now set up a single fleet to host your game servers in any or all of 20 GameLift-supported Regions (Beijing Region excepted). This feature aims to significantly reduce the work required to set up and maintain GameLift hosting resources globally. Multi-location fleets can be created in the following AWS Regions: us-east-1 (N. Virginia), us-west-2 (Oregon), eu-central-1 (Frankfurt), eu-west-1 (Ireland), ap-southeast-2 (Sydney), ap-northeast-1 (Tokyo), and ap-northeast-2 (Seoul). In all other Regions, you can continue to set up single-location fleets as needed. All fleets that were created prior to this release are single-location fleets. Using multi-location fleets does not affect your hosting costs; as previously, GameLift pricing is based on the type, location, and volume of instances you use (see GameLift Pricing). Cloud Formation support for multi-location fleets will be available soon.

Note

Multi-location fleets are not available in the China Region. GameLift resources that reside in a China Region cannot interact with or be used by resources in other GameLift Regions.

Highlights:

- With a multi-location fleet, explicitly add a list of remote locations. GameLift deploys instances of the same type and configuration, including the build and runtime configuration, to the fleet's home Region and all added locations.
- Adjust capacity settings and scaling for each location independently. Auto-scaling policies apply to an entire fleet, but you can turn them on or off by location.
- Start new game sessions at specific fleet locations. When using game session queues or matchmaking to place game sessions, you can now prioritize where new game sessions are started by location, hosting cost, and player latency.
• Get hosting metrics in the GameLift console, aggregated for all locations in a fleet or broken out by each fleet location.

Learn more:

• Amazon Game Tech Blog
• API reference (AWS SDK)
  • New fleet location operations: CreateFleetLocations, DescribeFleetLocationAttributes, DescribeFleetLocationCapacity, DescribeFleetLocationUtilization, DeleteFleetLocations
  • Updated fleet operations, with new multi-location support: CreateFleet, UpdateFleetCapacity, DescribeEC2InstanceLimits, DescribeInstances, StopFleetActions, StartFleetActions
  • Updated game session placement operations, with new priority & filtering capability: CreateGameSessionQueue, DescribeGameSessionQueues, UpdateGameSessionQueue
  • Updated game session creation operations, with new location support: CreateGameSession, DescribeGameSessions, DescribeGameSessionDetails, SearchGameSessions

• GameLift Developer Guide:
  • Using Amazon GameLift in AWS Regions (p. 25) (updated)
  • GameLift fleet design guide (p. 106) (new)
    • Scaling GameLift hosting capacity (p. 129) (updated)
    • Design a game session queue (p. 139) (new)
    • View fleet details (p. 163) (updated)
  • GameLift forum

February 9 2021: GameLift extends support for AMD instances, Standalone FlexMatch

Customers in China (Beijing) and all GameLift FleetIQ customers can now use AMD instances. GameLift FlexMatch as a standalone matchmaking solution is now available in China (Beijing), and FlexMatch matchmaking notification now supports SNS FIFO topics.

Release date: February 9, 2021

Updated SDK versions: AWS SDK 1.8.139

This release includes the following updates:

• GameLift FleetIQ game server groups can now be configured to manage the AMD instance families C5a, M5a, and R5a. Allowed EC2 instance types, as listed for the GameServerGroup InstanceDefinition, now include the following:
  • c5a.large, c5a.xlarge, c5a.2xlarge, c5a.4xlarge, c5a.8xlarge, c5a.12xlarge, c5a.16xlarge, c5a.24xlarge
  • m5a.large, m5a.xlarge, m5a.2xlarge, m5a.4xlarge, m5a.8xlarge, m5a.12xlarge, m5a.16xlarge, m5a.24xlarge
  • r5a.large, r5a.xlarge, r5a.2xlarge, r5a.4xlarge, r5a.8xlarge, r5a.12xlarge, r5a.16xlarge, r5a.24xlarge

Note: AMD instances for FleetIQ are currently not available for use in the China (Beijing) Region. See Feature Availability and Implementation Differences in China.

• GameLift managed game hosting now supports AMD instances in the China (Beijing) Region, operated by Sinnet. The new AMD instance families include M5a and R5a. Allowed EC2 instance types, as listed for Fleet InstanceType, now include the following:
  • m5a.large, m5a.xlarge, m5a.2xlarge, m5a.4xlarge, m5a.8xlarge, m5a.12xlarge, m5a.16xlarge, m5a.24xlarge
• r5a.large, r5a.xlarge, r5a.2xlarge, r5a.4xlarge, r5a.8xlarge, r5a.12xlarge, r5a.16xlarge, r5a.24xlarge

GameLift FlexMatch can now be used as a standalone matchmaking solution in the China (Beijing) Region, operated by Sinnet. Customers can create a FlexMatch matchmaker in the Beijing region and configure the FlexMatchMode parameter to STANDALONE. For more information about FlexMatch, either with GameLift managed hosting or with a non-GameLift hosting solution, in the GameLift FlexMatch Developer Guide.

• When setting up event notification for GameLift FlexMatch, you can now designate an SNS FIFO topic as the notification target. For more information, see:
  • MatchmakingConfiguration NotificationTarget, Amazon GameLift API Reference
  • Set up FlexMatch event notification, GameLift FlexMatch Developer Guide
  • Introducing Amazon SNS FIFO – First-In-First-Out Pub/Sub Messaging, AWS News Blog

December 22 2020: GameLift Server SDK supports Unreal Engine 4.25 and Unity 2020

Amazon GameLift releases an updated Server SDK to support game server development with Unreal Engine 4.25 and verifies support for Unity 2020.

Release date: December 22, 2020

Updated SDK versions: GameLift Server SDK 4.0.2, Unreal plugin version 3.3.3

The latest version of the GameLift Server SDK contains the following components:

• The updated Unreal plugin has been updated for compatibility with Unreal Engine 4.25. The API was not changed.
• The existing C# SDK, version 4.0.2, has been verified compatible with Unity 2020. No SDK updates were required.

Download the latest version of the GameLift Server SDK at Amazon GameLift Getting Started.

November 24 2020: GameLift FlexMatch now available for games hosted anywhere

Amazon GameLift extends availability of the FlexMatch matchmaking service to multiplayer games that use any hosting solution, including peer-to-peer, on-premises, or cloud services.

Release date: November 24, 2020

Updated SDK versions: AWS SDK 1.8.95

GameLift FlexMatch is a customizable matchmaking service for multiplayer games. Initially designed for users of GameLift managed hosting, FlexMatch can now be integrated into games that use other hosting systems, including peer-to-peer, proprietary on-premises computing, and cloud compute primitives. Games that use GameLift FleetIQ for game hosting on Amazon EC2 can now implement matchmaking with FlexMatch.

FlexMatch provides a robust matchmaking algorithm and rules language that gives you wide latitude to customize the matchmaking process so that players are matched together based on key player characteristics and reported latency. In addition, FlexMatch offers a matchmaking request workflow that supports features such as player parties, player acceptance, and match backfill. When FlexMatch is used with GameLift managed hosting or Realtime Servers, the matchmaker automatically triggers GameLift to find hosting resources and start a new game sessions for newly formed matches. When using FlexMatch
as a standalone service, the matchmaker delivers match results back to your game, which can then start a new game session using your hosting solution.

API operations for FlexMatch are part of the GameLift service API, which is included in the AWS SDK and the AWS CLI. This release includes these updates to support standalone matchmaking:

- The API resource `MatchmakingConfiguration` has the following changes:
  - New property, `FlexMatchMode` indicates whether the matchmaker is being used with GameLift managed hosting or as standalone matchmaking.
  - Property `GameSessionQueueArns` is not required when `FlexMatchMode` is set to standalone.
  - These properties are not used with standalone matchmaking: `AdditionalPlayerCount`, `BackfillMode`, `GameProperties`, `GameSessionData`.
  - The automatic backfill feature is not available with standalone matchmaking.

Learn more:

- Amazon Game Tech Blog
- GameLift FlexMatch Developer Guide:
  - How GameLift FlexMatch works
  - Getting started with FlexMatch
  - FlexMatch API reference (AWS SDK)
- GameLift forum

Note
The GameLift documentation has been expanded. Please see the GameLift documentation home page to find all GameLift content and resources. GameLift developer documentation now includes:

- GameLift Developer Guide for use with GameLift managed hosting and Realtime Servers.
- GameLift FleetIQ Developer Guide to optimize use of Amazon EC2 Spot Instances for game hosting
- GameLift FlexMatch Developer Guide for matchmaking.

November 24 2020: AMD instances now available on GameLift

*Amazon GameLift now offers AMD instance families for additional cost savings with multiplayer game hosting*

**Release date:** November 24, 2020

**Updated SDK versions:** AWS SDK 1.8.95

The list of Amazon EC2 instance types supported by GameLift now includes three new instance families: C5a, M5a, and R5a. These families consist of AMD compute-optimized instances that are powered by AMD EPYC processors running at frequencies up to 3.3 GHz. The AMD instances are x86 compatible; games that are currently running on GameLift can be deployed to AMD instance types without alteration. The new instances are available in the following AWS Regions: US East (N. Virginia and Ohio), US West (Oregon and N. California), Central Canada (Montreal), South America (Sao Paulo), EU Central (Frankfurt), EU West (London and Ireland), Asia Pacific South (Mumbai), Asia Pacific Northeast (Seoul and Tokyo), and Asia Pacific Southeast (Singapore and Sydney).

The new AMD instances include:

- c5a.large, c5a.xlarge, c5a.2xlarge, c5a.4xlarge, c5a.8xlarge, c5a.12xlarge, c5a.16xlarge, c5a.24xlarge
• m5a.large, m5a.xlarge, m5a.2xlarge, m5a.4xlarge, m5a.8xlarge, m5a.12xlarge, m5a.16xlarge, m5a.24xlarge
• r5a.large, r5a.xlarge, r5a.2xlarge, r5a.4xlarge, r5a.8xlarge, r5a.12xlarge, r5a.16xlarge, r5a.24xlarge

Learn more:
• Amazon Game Tech Blog
• Amazon GameLift Instance Pricing
• Amazon EC2 instances featuring AMD EPYC processors
• GameLift forum

November 11 2020: Version update to GameLift Server SDK

Amazon GameLift releases updated Server SDK to fix issue with StartMatchBackfill()

Release date: November 11, 2020

Updated SDK versions: GameLift Server SDK 4.0.2

The new Server SDK version 4.0.2 fixes a known issue with the method StartMatchBackfill(). This method now returns a correct response to a match backfill request.

The issue did not affect the match backfill process, and there is no change to how this feature works. The issue may have impacted log messaging and error handling for match backfill requests.

Download the latest version of the GameLift Server SDK at Amazon GameLift Getting Started.

November 5 2020: New FlexMatch algorithm customizations

Amazon GameLift releases customization options for FlexMatch matchmaking process.

Release date: November 5, 2020

FlexMatch users now have the ability to adjust the following default behaviors for the matchmaking process. These customizations are set in a matchmaking rule set. There are no changes to the GameLift SDKs.

• Prioritize backfill tickets: You can choose to raise or lower how match backfill tickets are prioritized when searching for acceptable matches. Prioritizing backfill tickets is particularly useful when the auto-backfill feature is enabled. Use the algorithm property backfillPriority.

• Pre-sort to optimize match consistency and efficiency: Configure your matchmaker to pre-sort the ticket pool prior to batching tickets for evaluation. By pre-sorting tickets based on key player attributes, your resulting matches tend to have players who are more similar in those attributes. You can also boost efficiency in the evaluation process by pre-sorting on the same attributes that are used in match rules. Use the algorithm property sortByAttributes with the strategy property set to "sorted".

• Adjust how expansion wait times are triggered: Choose between triggering expansions based on the age of the newest (default) or oldest ticket in an incomplete match. Triggering on the oldest ticket tends to complete matches faster, while triggering on the newest ticket leads to higher match quality. Use the algorithm property expansionAgeSelection.

Learn more:
GameLift Developer Guide
• Design a FlexMatch rule set: Customize the match algorithm
• Rule set schema – This topic now has detailed reference information for each rule set component.

September 17 2020: GameLift updates Server SDK

Amazon GameLift releases minor updates to the Server SDK

Release date: September 17, 2020

Updated SDK versions: GameLift Server SDK 4.0.1

The new Server SDK contains the following updates:

• C# API version 4.0.1
  • The method TerminateGameSession() (p. 248) is deprecated. Replace with a call to ProcessEnding() (p. 244) to end both a game session and the server process.
  • A known issue with the method GetInstanceCertificate() (p. 242) is fixed.
  • The method GetTerminationTime() (p. 243) now returns a value of data type AwsDateTimeOutcome.

• C++ API version 3.4.1
  • The method TerminateGameSession() (p. 234) is deprecated. Replace with a call to ProcessEnding() (p. 229) to end both a game session and the server process.

• Unreal Engine plugin version 3.3.2
  • The method TerminateGameSession() (p. 259) is deprecated. Replace with a call to ProcessEnding() (p. 256) to end both a game session and the server process.
  • The callback method OnUpdateGameSession is added to FProcessParameters (p. 261) to support match backfill.

Download the latest version of the GameLift Server SDK at Amazon GameLift Getting Started.

August 27 2020: GameLift FleetIQ for game hosting with Amazon EC2 (general availability)

Amazon GameLift extends its FleetIQ for Spot Instance optimization.

Release date: August 27, 2020

Updated SDK versions: AWS SDK 1.8.36

The GameLift FleetIQ solution for low-cost, cloud-based game hosting on Amazon EC2 is now generally available. GameLift FleetIQ gives developers the ability to host game servers directly on Amazon EC2 Spot Instances by optimizing their viability for game hosting. Game developers can use GameLift FleetIQ with new games or to supplement capacity for existing games. This solution supports the use of containers or other AWS services such as AWS Shield and Amazon Elastic Container Service (Amazon ECS).

This general availability release includes the following updates to the GameLift FleetIQ solution:

• New API operation DescribeGameServerInstances returns information, including status, on all active instances for a GameLift FleetIQ game server group.
• New balancing strategy, ON_DEMAND_ONLY, configures a game server group to use On-Demand Instances only. A game server group’s balancing strategy can be updated at any time, making it possible to switch between using Spot Instances and On-Demand instances as needed.
• The following preview elements have been dropped for general availability:
• Use of custom sort keys for game server resources. Game servers can be sorted based on registration timestamp.
• Tagging for game server resources.

Learn more:
• AWS Training and Certification course: Using Amazon GameLift FleetIQ for Game Servers
• GameLift FleetIQ API Reference
• GameLift FleetIQ Developer Guide
• Announcements on Amazon Game Tech Blog and AWS What's New Feed

April 16 2020: GameLift updates Server SDK for Unity and Unreal Engine

Amazon GameLift releases an updated Server SDK to support game server development with Unity 2019 and Unreal Engine 4.24.

Release date: April 16, 2020

Updated SDK versions: GameLift Server SDK 4.0.0, GameLift Local 1.0.5

The latest version of the GameLift Server SDK contains the following updated components:
• C# SDK version 4.0.0 updated for Unity 2019
• Unreal plugin version 3.3.1 updated for Unreal Engine 4.22, 4.23, and 4.24
• GameLift Local version 1.0.5 updated to test integrations that use the C# server SDK v 4.0.0

Download the latest version of the GameLift Server SDK at Amazon GameLift Getting Started.

April 2 2020: GameLift FleetIQ available for game hosting on EC2 (public preview)

Amazon GameLift extends its FleetIQ for Spot Instance optimization.

Release date: April 02, 2020

Updated SDK versions: AWS SDK 1.7.310

The GameLift FleetIQ feature optimizes the viability of low-cost Spot Instances for use with game hosting. This feature is now extended for customers who want to manage their hosting resources directly rather than through the managed GameLift service. This solution supports the use of containers or other AWS services such as AWS Shield and Amazon Elastic Container Service (Amazon ECS).

Learn more:
• GameTech blog post on GameLift FleetIQ

December 19 2019: Improved AWS resource management for GameLift resources

Amazon GameLift adds new ARN values and tagging support
Release date: December 19, 2019

Updated SDK versions: AWS SDK 1.7.249

You can now take advantage of AWS resource management tools with GameLift resources. In particular, all key GameLift resources -- builds, scripts, fleets, game session queues, matchmaking configurations, and matchmaking rule sets -- are now assigned ARN values. A resource ARN provides a consistent identifier that is unique across all Regions. They can be used to create resource-specific IAM permissions policies. Resources are now assigned an ARN as well as the pre-existing resource identifier, which is not Region-specific.

In addition, GameLift resources now support tagging. You can use tags to organize resources, create IAM permissions policies to manage access to groups of resources, customize AWS cost breakdowns, etc. When managing tags for GameLift resources, use the GameLift API actions TagResource(), UntagResource(), and ListTagsForResource().

Learn more:
- TagResource in the Amazon GameLift API Reference
- Tagging AWS Resources in the AWS General Reference
- Amazon Resource Names in the AWS General Reference

November 14 2019: New CloudFormation templates, updates in China (Beijing) Region

Amazon GameLift expands use of CloudFormation templates to help automate GameLift resource management

Release date: December 19, 2019

Updated SDK versions: AWS SDK 1.7.210

CloudFormation templates for GameLift

GameLift resources can now be created and managed through AWS CloudFormation. The existing AWS CloudFormation templates for Build and Fleet have been updated to align with the current resources, and new templates are now available for Script, Queue, MatchmakingConfiguration, and MatchmakingRuleSet resources. AWS CloudFormation templates greatly simplify the task of managing groups of related AWS resources, particularly when deploying games across multiple regions.

Learn more:
- Amazon GameLift Resource Type Reference in the AWS CloudFormation User Guide
- Create resources using AWS CloudFormation (p. 91) in the Amazon GameLift Developer Guide

Lumberyard compatibility

For Amazon Lumberyard users, the following table lists the GameLift SDK versions that are bundled into or are compatible with the Lumberyard game engine.

<table>
<thead>
<tr>
<th>Amazon Lumberyard version:</th>
<th>Are bundled with GameLift SDK versions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4 to 1.5 (beta)</td>
<td>• Server SDK: 3.0.7</td>
</tr>
<tr>
<td></td>
<td>• AWS SDK: 0.12.16</td>
</tr>
</tbody>
</table>

Version

280
## Amazon Lumberyard version:

<table>
<thead>
<tr>
<th>Version</th>
<th>Are bundled with GameLift SDK versions:</th>
</tr>
</thead>
</table>
| 1.6 to 1.7 (beta)        | • Server SDK: 3.1.0  
                          | • AWS SDK: 0.14.9                                                  |
| 1.8 to 1.14 (beta)       | • Server SDK: 3.1.5  
                          | • AWS SDK: 1.0.72 to 1.1.13                                       |
| 1.15 to 1.25             | • Server SDK: 3.2.1  
                          | • AWS SDK: 1.4.34 or later                                        |
| 1.26 and later           | • Server SDK: 3.4.0  
                          | • AWS SDK: 1.7.310 or later                                       |
Limits and Supported Regions

For AWS GameLift service limits, see Amazon GameLift Limits.

For information about requesting limit increases for AWS resources, see AWS Service Limits.

For a list of the AWS Regions supporting Amazon GameLift, see Amazon GameLift Regions.
# Document history for GameLift

The following table describes important changes to the Amazon GameLift documentation. For details on releases of new and updated features, see the GameLift release notes (p. 269).

<table>
<thead>
<tr>
<th>Date</th>
<th>Change</th>
<th>Documentation updates</th>
<th>API Versions Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 16, 2020</td>
<td>Updates</td>
<td>Developer Guide:</td>
<td>Server SDK 4.0.0</td>
</tr>
<tr>
<td></td>
<td>• <strong>New Server SDK version</strong></td>
<td>• Updated docs on building and using the Server SDK with Unity (p. 49) and Unreal (p. 45) games.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>April 2, 2020</td>
<td><strong>New Feature</strong></td>
<td>Developer Guide:</td>
<td>AWS SDK: 2020-04-02</td>
</tr>
</tbody>
</table>
|                  | • FleetIQ as a standalone solution (public preview) | • New guide:  
|                  |                   |                       |                      |
|                  |                   | • [https://docs.aws.amazon.com/gamelift/latest/fleetiqguide/gsg-intro.html](https://docs.aws.amazon.com/gamelift/latest/fleetiqguide/gsg-intro.html) |                      |
|                  |                   | Service API Reference (AWS SDK): |                      |
|                  |                   | • New FleetIQ operations:  
|                  |                   | • TagResource  
|                  |                   | • UntagResource  
|                  |                   | • ListTagsForResource |                      |
| December 12, 2019| Updates           | Service API Reference (AWS SDK): | AWS SDK: 2019-12-12  |
|                  | • AWS resource management support – Key GameLift resources now have ARN values and support tagging. | • New tagging operations:  
|                  |                   |                       |                      |
|                  |                   | • TagResource  
|                  |                   | • UntagResource  
<p>|                  |                   | • ListTagsForResource |                      |
|                  |                   | • New tag and ARN parameters for these GameLift resources: builds, scripts, fleets, game session queues, matchmaking configurations, matchmaking rule sets. |                      |
| November 15, 2019| Updates           | Developer Guide:      |                      |
|                  | • Expanded CloudFormation | • New topic:          |                      |</p>
<table>
<thead>
<tr>
<th>Date</th>
<th>Change</th>
<th>Documentation updates</th>
<th>API Versions Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>October 24, 2019</td>
<td><strong>Updates</strong></td>
<td>• Create resources using AWS CloudFormation (p. 91)</td>
<td>AWS SDK: 2019-10-24</td>
</tr>
<tr>
<td></td>
<td><strong>template support for GameLift</strong> – Existing CloudFormation templates for Build and Fleet have been updated and there are new templates Script, Queue, MatchmakingConfiguration, and MatchmakingRuleSet.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Support for Amazon Linux 2 and Series 5 instance types – Host games using the new C5/M5/R5 instance types. Game servers can now run on Amazon Linux 2. All Realtime Servers now run on Amazon Linux 2.</td>
<td>Developer Guide and API reference docs updated to reflect the additional options. Updates were released in the Global Regions on October 24, 2019, and released in the China (Beijing) Region, operated by Sinnet, on November 15, 2019.</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Change</td>
<td>Documentation updates</td>
<td>API Versions Updated</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>September 3, 2019</td>
<td><strong>New features</strong></td>
<td>• <strong>TLS certificate generation</strong> – GameLift can now generate fleet-level TLS certificates to support server authentication and data packet encryption for custom and Realtime servers.</td>
<td><a href="https://aws.amazon.com">AWS SDK: 2019-09-03</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Developer Guide:</td>
<td>Server SDK 3.4.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The following topics are updated with information about TLS certificates:</td>
<td>Realtime Client SDK 1.1.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Hosting game servers (p. 4)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• How Realtime Clients and Servers Interact (p. 10)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Retrieve a TLS certificate (p. 53)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Integrating a Game Client for Realtime Servers (p. 71)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>SDK &amp; API References:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GameLift Service API, new CreateFleet() parameter CertificateConfiguration</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GameLift Server SDK, new operation GetInstanceCertificate() C++ (p. 227) and C# (p. 242)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Realtime Client SDK, new parameter the section called “Data Types” (p. 214)</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Change</td>
<td>Documentation updates</td>
<td>API Versions Updated</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>July 9, 2019</td>
<td><strong>New features</strong></td>
<td>• FlexMatch support for large matches – Matchmaking expanded to support 200 players per match with new, faster algorithm. Match backfill now has an automatic mode.</td>
<td>AWS SDK: 2019-07-09</td>
</tr>
</tbody>
</table>

**Developer Guide:**
- FlexMatch topics updated or expanded to include information on working with large matches:
  - How GameLift FlexMatch works
  - Design a FlexMatch rule set
  - FlexMatch rule set schema
  - FlexMatch rule set examples
  - Backfill existing games with FlexMatch
  - Design a FlexMatch matchmaker

**Service API Reference (AWS SDK):**
- Matchmaking configuration API operations updated to support automatic backfill:
  - CreateMatchmakingConfiguration
  - UpdateMatchmakingConfiguration
  - StopMatchmaking
<table>
<thead>
<tr>
<th>Date</th>
<th>Change</th>
<th>Documentation updates</th>
<th>API Versions Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 25, 2019</td>
<td>New features</td>
<td><strong>Developer Guide:</strong></td>
<td>AWS SDK: 2019-04-25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• New topics on Realtime Servers:</td>
<td>Realtime Client SDK 1.0.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• How Realtime Servers Work (p. 8)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Integrating Games with Amazon GameLift Realtime Servers (p. 70)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Realtime Servers Client API (C#) Reference (p. 208)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Service API Reference (AWS SDK):</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• New API operations to manage Realtime Servers scripts: CreateScript, DescribeScript, UpdateScript, DeleteScript, ListScripts</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Fleet API operations updated to enable Realtime Servers deployment with scripts.</td>
<td></td>
</tr>
<tr>
<td>March 7, 2019</td>
<td>New features</td>
<td><strong>Developer Guide:</strong></td>
<td>AWS SDK: 2019-03-07</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• New topic Communicate with other AWS resources from your fleets (p. 55) on secure access options.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Service API Reference (AWS SDK):</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CreateFleet and FleetAttributes – InstanceRoleArn added.</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Change</td>
<td>Documentation updates</td>
<td>API Versions Updated</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td></td>
<td>• Delete matchmaking rule</td>
<td>• Create matchmaking rule sets updated for matchmaking rule set deletion.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sets</td>
<td>Service API Reference (AWS SDK):</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• DeleteMatchmakingRuleSet (new)</td>
<td></td>
</tr>
<tr>
<td>December 14, 2018</td>
<td>Updates</td>
<td>Developer Guide:</td>
<td>Server SDK 3.3.0</td>
</tr>
<tr>
<td></td>
<td>• New Server SDK version</td>
<td>• Updated docs on building and using the Server SDK with Unity (p. 49) and Unreal (p. 45) games.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>– Updated Server SDK is</td>
<td>• Lumberyard compatibility (p. 280) updated.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>now compatible with</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>latest Unity &amp; Unreal</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>engines and can be used</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>with Visual Studio 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2017.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• FlexMatch Backfill</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>supported for Unreal</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>engine games.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• GameLift in China – The</td>
<td>• New topic Using Amazon GameLift in AWS Regions (p. 25) provides links to information on using AWS in China.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GameLift service is now</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>available in the AWS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>China (Beijing) Region</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>operated by Sinnet.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Queue metrics for FleetIQ</td>
<td>• Five new queue metrics added to GameLift metrics for queues (p. 176).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>– Use FleetIQ metrics to</td>
<td>• New topic Evaluate queue metrics (p. 142) to optimize queue performance.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>track queue performance.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Change</td>
<td>Documentation updates</td>
<td>API Versions Updated</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------</td>
<td>----------------------</td>
</tr>
</tbody>
</table>
| May 10, 2018| New features    | • **Auto-scaling with Target Tracking** – Use this new scaling method to match fleet capacity to player demand.  
• **Enable/disable auto-scaling** – Switch between auto-scaling and manual scaling without having to delete scaling policies. | AWS SDK: 2018-05-10  |
|             | Developer Guide:| • New section *Scaling GameLift hosting capacity* (p. 129) encompasses all scaling-related topics, including auto-scaling.  
• Updated section on *Scaling fleet capacity* (p. 6) in the "How GameLift Works" topic. |                      |
|             | Service API Reference (AWS SDK): | • *PutScalingPolicy* (updated with new examples)  
• *StopFleetActions* and *StartFleetActions* (new) |                      |
<table>
<thead>
<tr>
<th>Date</th>
<th>Change</th>
<th>Documentation updates</th>
<th>API Versions Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>February 15, 2018</td>
<td><strong>New features</strong></td>
<td><strong>Developer Guide:</strong></td>
<td>AWS SDK: 2018-02-15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Spot Instances and FleetIQ – Use the new FleetIQ feature with spot fleets to significantly lower hosting costs.</td>
<td>Server SDK 3.2.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Developer Guide:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Using Spot Instances with GameLift (p. 110) – New step-by-step guide for spot fleet usage.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Design a game session queue (p. 139) – 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>Choosing computing resources (p. 107)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deploy a GameLift fleet with a custom game build (p. 112)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Respond to a server process shutdown notification (p. 54)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Service API Reference (AWS SDK):</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• New FleetType parameter added to enable spot fleets:</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></td>
<td></td>
<td><strong>Server SDK:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GetTerminationTime (C++) (p. 228) (new)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GetTerminationTime (C#) (p. 243) (new)</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Change</td>
<td>Documentation updates</td>
<td>API Versions Updated</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>February 8, 2018</td>
<td><strong>New features</strong></td>
<td><strong>Developer Guide:</strong></td>
<td>AWS SDK: 2018-02-08</td>
</tr>
<tr>
<td></td>
<td>• <strong>FlexMatch Backfill</strong> – Use FlexMatch to inject new players into matched game sessions that are in progress.</td>
<td>• How GameLift FlexMatch works – Updated FlexMatch feature overview.</td>
<td>Server SDK 3.2.0</td>
</tr>
<tr>
<td></td>
<td>• <strong>Game session search</strong> – use custom game properties</td>
<td>• Backfill existing games with FlexMatch – New how-to guide.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Get Game Sessions (p. 59) – Updated for custom game properties.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Service API Reference (AWS SDK):</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• StartMatchBackfill (new)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SearchGameSessions (new)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GameSession (new)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Server SDK:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• StartMatchBackfill (C++) (p. 232) (new)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• StartMatchBackfill (C#) (p. 246) (new)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• StopMatchBackfill (C++) (p. 233) (new)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• StopMatchBackfill (C#) (p. 247) (new)</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Change</td>
<td>Documentation updates</td>
<td>API Versions Updated</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
<td>-------------------------</td>
</tr>
</tbody>
</table>
| September 3, 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:**  
- Interacting with other AWS resources (p. 8) – Get more information on VPC peering and how it works with GameLift.  
- VPC peering for GameLift (p. 156) – Learn more about how to set up VPC peering. | AWS SDK: 2017-09-01 |

**Service API Reference:**

- New APIs for VPC peering:
  - CreateVpcPeeringAuthorization
  - DescribeVpcPeeringAuthorizations
  - DeleteVpcPeeringAuthorization
  - CreateVpcPeeringConnection
  - DescribeVpcPeeringConnections
  - DeleteVpcPeeringConnection
- MatchmakingTicket  
  -EndTime and EstimatedWaitTime added.
- The documentation for DescribeMatchmaking was corrected. This operation can be used to retrieve a maximum of 10 matchmaking tickets.
<table>
<thead>
<tr>
<th>Date</th>
<th>Change</th>
<th>Documentation updates</th>
<th>API Versions Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>August 16, 2017</td>
<td><strong>New features</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>GameLift FlexMatch</strong> – Add matchmaking to your games using this customizable matchmaking service. With FlexMatch, you can design a rule set based on the team formats and player characteristics that best fit your game.</td>
<td><strong>Developer Guide:</strong></td>
<td><strong>AWS SDK: 2017-08-16</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• How GameLift FlexMatch works – Get more information on FlexMatch key features and how it works.</td>
<td><strong>Server SDK 3.1.7</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Adding FlexMatch matchmaking – Learn more about how to set up FlexMatch and customize it to your game.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Server SDK:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GameLift Server API – New APIs for managing FlexMatch resources and starting new games with matchmaking.</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Change</td>
<td>Documentation updates</td>
<td>API Versions Updated</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
<td>---------------------------------------</td>
</tr>
</tbody>
</table>
|                 | • GameLiftMetrics are now supported in Amazon CloudWatch. This includes the ability to work with aggregated metrics for a group of fleets. | • Monitoring Amazon GameLift (p. 170)  
- New monitoring section including list of metrics available in the GameLift console and in Cloudwatch. 
- Deploy a GameLift fleet with a custom game build (p. 112) and Manage your GameLift fleets (p. 120) – Updated instructions on creating and updating fleet configurations. 
- Auto-scale fleet capacity with GameLift (p. 132) – Updated instructions on setting manual and auto-scaling for a fleet. |                                       |
<p>|                 | • Limit game session activations on instances in a fleet.                                      |                                                                                           |                                       |
|                 | Updates:                                     | Service API Reference:                                                                     |                                       |
|                 | • Take advantage of additional metrics for automatic scaling.                                  | • New MetricGroups parameter added to enable aggregated metrics:                          |                                       |
|                 | • Use new console UI to set fleet scaling.                                                    |     - CreateFleet                                                                     |                                       |
|                 |                                                                                             |     - UpdateFleetAttributes                                                            |                                       |
|                 |                                                                                             |     - FleetAttributes                                                                |                                       |
|                 |                                                                                             |     - RuntimeConfiguration – game session activation limits added.                     |                                       |</p>
<table>
<thead>
<tr>
<th>Date</th>
<th>Change</th>
<th>Documentation updates</th>
<th>API Versions Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 11, 2017</td>
<td><strong>New Features:</strong></td>
<td>Developer Guide:</td>
<td>AWS SDK: 2017-04-11</td>
</tr>
<tr>
<td></td>
<td>• Amazon GameLift Local – Test your game integration locally.</td>
<td>• Testing Your Integration (p. 65) – New topic on setting up and using Amazon GameLift Local.</td>
<td>GameLiftLocal 1.0.0</td>
</tr>
<tr>
<td></td>
<td>• Game session queues – Create multi-fleet, multi-region queues to place new game sessions with the best possible hosting resources for latency, cost, and resiliency.</td>
<td>• Create a game session queue (p. 148) – 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>Service API Reference:</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>• PlayerLatencyPolicy (new data type)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• PlacedPlayerSession (new data type)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GameSessionPlacement – GameSessionId, IpAddress, Port, PlacedPlayerSessions added.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CreateGameSession – IdempotencyToken replaces GameSessionId.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GameSessionQueue – GameSessionQueueArn added.</td>
<td></td>
</tr>
<tr>
<td>February 21, 2017</td>
<td><strong>New features</strong></td>
<td>Developer Guide:</td>
<td>AWS SDK: 2017-02-21</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 managed GameLift (p. 13) – Architecture diagram.</td>
<td>Server SDK 3.1.5</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. 43) – Help with using 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>• Setting up GameLift queues for game session placement (p. 137) – 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 Updated</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>November 18, 2016</td>
<td><strong>New features</strong></td>
<td>• Remote access to GameLift fleet instances</td>
<td>AWS SDK: 2016-11-18</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Developer Guide:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• New topics:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Remotely access GameLift fleet instances (p. 126)</strong> – How to get access and remotely connect to GameLift instances.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>Debug GameLift fleet issues (p. 124)</strong> – Troubleshooting tips for new fleets that fail to activate.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Service API Reference:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• For remote access:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>GetInstanceAccess</strong> (new)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>InstanceAccess</strong> (new)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• <strong>InstanceCredentials</strong> (new)</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Change</td>
<td>Documentation updates</td>
<td>API Versions Updated</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
</tbody>
</table>
| October 13, 2016 | **New features**  | • Resource creation protection  
• Access to instance data  | **Developer Guide:**  
• Revised topics:  
  • *How GameLift works (p. 3)* – Added description of resource protection, and improved description of capacity handling.  
  • Added Linux-specific help:  
    • *Package your game build files (p. 99)* – Install scripts for Linux.  
    • *Upload a custom server build to GameLift (p. 97)* – New Linux examples.  
    • *Deploy a GameLift fleet with a custom game build (p. 112)*  – New launch path example for Linux.  | **AWS SDK: 2016-10-13** |
|            | **Updates and corrections** | • Additional help for Linux.  | **Service API Reference:**  
• *CreateFleet* and *UpdateFleetAttributes*  – New  
  *ResourceCreationLimitPolicy* parameter.  
• *ResourceCreationLimitPolicy* (new)  
• *CreateGameSession* – *CreatorId* added.  
• *DescribeInstances* (new)  |
<table>
<thead>
<tr>
<th>Date</th>
<th>Change</th>
<th>Documentation updates</th>
<th>API Versions Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>September 1, 2016</td>
<td><strong>New features</strong></td>
<td>• Game servers can now run on Linux</td>
<td>AWS SDK: 2016-09-01</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Developer Guide:</strong></td>
<td>Server SDK for C++: 3.1.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• New topics:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GameLift SDKs (p. 22)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Reference topic describing all GameLift SDKs, including supported languages and operating systems.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Service API Reference:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• New OS parameters were added to the following:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• upload-build (p. 97) (CLI only)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CreateBuild()</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Build</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• FleetAttributes</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Change</td>
<td>Documentation updates</td>
<td>API Versions Updated</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
</tbody>
</table>
| August 4, 2016 | **New features**       | • Game session search  
• Customized health checks  

**Updates:**  
• Expanded support for capacity allocation (multiple processes per fleet instance)  
• 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:  
  • GameLift Server API reference for C++ (p. 224) – Complete reference documentation.  
  • Managing how game servers are launched for hosting (p. 109) – Technical overview of capacity allocation and how to configure a fleet to run multiple processes.  
  • Tools and Resources (p. 24) – Comprehensive list of tools & resources, including SDK version compatibility.  
• Revised topics:  
  • How Players Connect to Games (p. 12) – Expanded topic describes features related to game sessions, including the new search feature.  
  • Add GameLift to your game server (p. 51) – Integration steps have been revised for use with version 3.0.7 Server SDK for C++.  
  • Add Amazon GameLift to Your Game Client (p. 58) – Integration steps have been revised for use with the AWS SDK for C++.  

**Service API Reference:**  
• SearchGameSessions() (new)  

|          |                         | AWS SDK: 2016-08-04  
<p>|          |                         | Server SDK for C++: 3.0.7 |</p>
<table>
<thead>
<tr>
<th>Date</th>
<th>Change</th>
<th>Documentation updates</th>
<th>API Versions Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• Revised topics:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Package your game build files (p. 99)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Description now reflects how GameLift handles an install.bat file in a game build.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Deploy a GameLift fleet with a custom game build (p. 112)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>– 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. 163) and View Data on Game and Player Sessions (p. 167)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Console page descriptions now reflect current metrics and scaling tabs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• GameLift and game client/server interactions (p. 62)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>– Descriptions and diagram (p. 64) 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>Date</td>
<td>Change</td>
<td>Documentation updates</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
<td>-----------------------</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.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 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></td>
<td></td>
<td>API Versions Updated</td>
<td></td>
</tr>
</tbody>
</table>

Version 301
<table>
<thead>
<tr>
<th>Date</th>
<th>Change</th>
<th>Documentation updates</th>
<th>API Versions Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>March 10, 2016</td>
<td>New features</td>
<td>• Auto-scaling&lt;br&gt;• Game session protection&lt;br&gt;• Fleet capacity limits</td>
<td>AWS SDK 2016-03-10</td>
</tr>
<tr>
<td></td>
<td>Developer Guide:</td>
<td>• New topics:&lt;br&gt;• Auto-scale fleet capacity with GameLift (p. 132) – How to set up and manage auto-scaling policies.&lt;br&gt;• Manually set capacity for a GameLift fleet (p. 131) – How to change the number of instances in a fleet and set limits.&lt;br&gt;• How GameLift works (p. 3) – A technical overview of how GameLift manages game deployment across virtual resources.&lt;br&gt;• Revised topics:&lt;br&gt;• Deploy a GameLift fleet with a custom game build (p. 112) – Revised to include settings for game session protection and safe scaling.&lt;br&gt;• Other changes:&lt;br&gt;• Lumberyard-GameLift tutorial was moved to the GameDev Tutorials repository.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Service API Reference:</td>
<td>• For auto-scaling:&lt;br&gt;• PutScalingPolicy&lt;br&gt;• DescribeScalingPolicies&lt;br&gt;• DeleteScalingPolicy&lt;br&gt;• For game session protection:&lt;br&gt;• DescribeGameSessionDetails&lt;br&gt;• CreateFleet (revised)&lt;br&gt;• UpdateFleetAttributes (revised)&lt;br&gt;• DescribeFleetAttributes (revised)</td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>Change</td>
<td>Documentation updates</td>
<td>API Versions Updated</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------</td>
<td>-----------------------------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>February 9, 2016</td>
<td>Service launch</td>
<td>Developer Guide and API Reference for the GameLift service released on AWS.</td>
<td>AWS SDK 2016-02-09</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>UpdateGameSession</strong> (revised)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>For fleet capacity limits:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>UpdateFleetCapacity</strong> (revised)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DescribeFleetCapacity</strong> (revised)</td>
<td></td>
</tr>
</tbody>
</table>
GameLift developer resources

To view all GameLift documentation and developer resources, see the Amazon GameLift Documentation home page.
AWS glossary

For the latest AWS terminology, see the AWS glossary in the AWS General Reference.