aws

Developer Guide

Amazon GameLift

Amazon GameLift: Developer Guide

Table of Contents

What is Amazon GameLift?

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

Uses of Amazon GameLift

Amazon GameLift supports these use cases and more:

- Use your own custom multiplayer game servers, or use ready-to-go Realtime servers to host your games.
- Run low cost hosting resources using Amazon Elastic Compute Cloud (Amazon EC2) Spot Instances.
- Automatically scale the amount of hosting resources that your game needs based on usage.
- Manage your Amazon EC2 compute resources all in one place using Amazon GameLift FleetIQ.
- Match players in multiplayer games with Amazon GameLift FlexMatch.
- Iteratively test your game server and client builds with Amazon GameLift Anywhere.
- Use your own hardware while managing it all in one place with Amazon GameLift Anywhere.

> **Tip**
>
> To try out Amazon GameLift game server hosting, see Getting started with Amazon GameLift.

Get started with Amazon GameLift solutions

Amazon GameLift solutions for game developers

- Amazon GameLift hosting for custom servers
- Amazon GameLift hosting with Realtime Servers
- Amazon GameLift FleetIQ for hosting on Amazon EC2

- Amazon GameLift FlexMatch for matchmaking
- Amazon GameLift Anywhere hardware hosting

Amazon GameLift hosting for custom servers

Amazon GameLift replaces the work required to host your own custom game servers. Auto scaling capabilities help you avoid paying for more resources than you need. Auto scaling also helps make sure that you always have games available for new players to join with minimal waiting.

For more information about Amazon GameLift hosting, see How Amazon GameLift works.

Key features

- Use Amazon GameLift management features, including auto scaling, multi-location queues, and game session placement.
- Deploy game servers to run on Amazon Linux or Windows Server operating systems.
- Manage game sessions and player sessions.
- Set up customized health tracking for server processes to detect problems and to resolve poor-performing processes.
- Manage your game resources using AWS CloudFormation templates for Amazon GameLift.

Amazon 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 game servers that you can configure to fit your game.

For more information about Amazon GameLift hosting with Realtime Servers, see Integrating games with Amazon GameLift Realtime Servers.

Key features

- Use Amazon GameLift management features, including auto scaling, multi-location queues, and game session placement.
- Use Amazon GameLift hosting resources and choose the type of AWS computing hardware for your fleets.
- Take advantage of a full network stack for game client and server interaction.

- Get core game server functionality with customizable server logic.
- Make live updates to Realtime configurations and server logic.

Amazon GameLift FleetIQ for hosting on Amazon EC2

Use Amazon GameLift FleetIQ to work directly with your hosting resources in Amazon EC2 and Amazon EC2 Auto Scaling. This provides the benefit of Amazon GameLift optimizations for inexpensive, resilient game hosting. This solution is for game developers who need more flexibility than what fully managed Amazon GameLift solutions provide.

For information about how Amazon GameLift FleetIQ works with Amazon EC2 and EC2 Auto Scaling for game hosting, see the Amazon GameLift FleetIQ Developer Guide.

Key features

- Get optimized Spot Instance balancing using the FleetIQ algorithm.
- Use player routing features to manage your game server resources efficiently, and provide a better player experience for joining games.
- Automatically scale hosting capacity based on player usage.
- Directly manage Amazon EC2 instances in your own AWS account.
- Use any of the supported game server executable formats, including Windows, Linux, containers, and Kubernetes.

Amazon GameLift FlexMatch for matchmaking

Use FlexMatch to build custom rule sets to define multiplayer matches for your game. FlexMatch uses rule sets to compare compatible players for each match and provide players with the ideal multiplayer experience.

For more information about FlexMatch, see What is Amazon GameLift FlexMatch?

Key features

- Balance match creation speed and match quality.
- Match players or teams based on defined characteristics.
- Define rules to place players into matches based on latency.

Amazon GameLift Anywhere hardware hosting

Use Amazon GameLift Anywhere to integrate hardware anywhere in your environment into your Amazon GameLift game hosting. You can integrate Anywhere fleets and EC2 fleets in matchmaker and game session queues to manage matchmaking and game placement across your hardware.

For more information about testing with Anywhere, see Test your integration using Amazon GameLift Anywhere fleets. For more information about setting up an Anywhere fleet, see Setting up Amazon GameLift fleets.

Key features

- Perform fast, iterative testing of your game server and client builds.
- Use the set Amazon GameLift tools to deploy games to your own hardware.
- Use hardware closest to your players, anywhere.

Accessing Amazon GameLift

Use these tools to work with Amazon GameLift.

Amazon GameLift SDKs

The Amazon GameLift SDKs contain the libraries needed to communicate with Amazon GameLift from your game clients, game servers, and game services. For more information, see Development support with Amazon GameLift.

Amazon GameLift Realtime Client SDK

The Realtime Client SDK enables a game client to connect to the Realtime server, join game sessions, and stay in sync with other players. Download the SDK and learn more about making API calls with the Realtime Servers client API (C#).

Amazon GameLift console

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 Command Line Interface (AWS CLI).

AWS CLI

Use this command line tool to make calls to the AWS SDK, including the Amazon GameLift API. For information about using the AWS CLI, see Getting started with the AWS CLI in the *AWS Command Line Interface User Guide*.

Pricing for Amazon GameLift

Amazon GameLift charges for instances by duration of use, and for bandwidth by quantity of data transferred. For a complete list of charges and prices for Amazon GameLift, see Amazon GameLift Pricing.

For information about calculating the cost of hosting your games or matchmaking with Amazon GameLift, see Generating Amazon GameLift pricing estimates, which describes how to use the AWS Pricing Calculator.

How Amazon GameLift works

This topic covers the core components for game hosting and describes how Amazon GameLift makes your multiplayer game servers available to players.

Ready to prepare your game for hosting on Amazon GameLift? Check out Amazon GameLift managed hosting roadmap.

Key components

Setting up Amazon GameLift to host your game involves working with the following components. The diagram in Game architecture with managed Amazon GameLift visualizes the relationships between these components.

- A **game server** is your game's server software running on a **fleet**. You upload your **game server build** or **script** to Amazon GameLift and tell Amazon GameLift. When you use Amazon GameLift Anywhere or Amazon GameLift FleetIQ, you upload your game server build directly to the compute resource.
- A **game session** is an in progress game with players. You define the basic characteristics of a game session, such as its life span and number of players. Players then connect to the game server to join a game session.

- A **game client** is your game's software running on a player's device. A game client connects to a game server through backend services to join a game session, based on connection information that it receives from Amazon GameLift.
- **Backend services** are additional, custom services that handle tasks related to Amazon GameLift. As a best practice, your backend services should handle all game client communication with Amazon GameLift.

Hosting game servers

With Amazon GameLift, you can host your game servers in three different ways: Managed Amazon GameLift, Amazon GameLift FleetIQ, and Amazon GameLift Anywhere. For more information about Amazon GameLift FleetIQ, see What is Amazon GameLift FleetIQ?

You can design a fleet to fit your game's needs. For more information about designing a fleet, see Amazon GameLift fleet design guide.

Managed Amazon GameLift

With managed Amazon GameLift, you can host your game servers on Amazon GameLift virtual computing resources, called **instances**. Set up your hosting resources by creating a **fleet** of instances and deploying them to run your game servers.

Amazon GameLift Anywhere

With Amazon GameLift Anywhere, you can host your game servers on compute that you manage. Set up your hosting resources by creating an Anywhere fleet that references your compute.

Fleet aliases

An **alias** is a designation that you can transfer between fleets, making it a convenient way to have a generic fleet location. You can use an alias to switch game clients from using one fleet to another without changing your game client. You can also create a terminal alias that you point to content.

Running game sessions

After you deploy your game server build to a fleet and Amazon GameLift launches game server processes on each instance, the fleet can host game sessions. Amazon GameLift starts new game sessions when your game client service sends a placement request to the backend service or to Amazon GameLift.

Game session placement and the FleetIQ algorithm

Queues use the FleetIQ algorithm to select an available game server to host a new game session. The key component for game session placement is the Amazon GameLift game session **queue**. You assign a game session queue a list of fleets, which determines where the queue can place game sessions. For more information about game session queues and how to design them for your game, see Design a game session queue.

Player connections to games

As part of the game session placement process, the queue or game session prompts the selected game server to start a new game session. The game server responds to the prompt and reports back to Amazon GameLift when it's ready to accept player connections. Amazon GameLift then delivers connection information to the backend service or game client service. Your game clients use this information to connect directly to the game session and begin gameplay.

Scaling fleet capacity

When a fleet is active and ready to host game sessions, you can adjust your fleet capacity to meet player demand. We recommend that you find a balance between all incoming players finding a game quickly and overspending on resources that sit idle.

Amazon GameLift provides a highly effective auto scaling tool, or you can manually set fleet capacity. For more information, see Scaling Amazon GameLift hosting capacity.

Auto scaling

Amazon GameLift provides two methods of auto scaling:

- Target-based auto scaling
- Auto scale with rule-based policies

Additional scaling features

- **Game session protection** – Prevent Amazon GameLift from ending game sessions that are hosting active players during a scale-down event.
- **Scaling limits** – Control overall instance usage by setting minimum and maximum limits on the number of instances in a fleet.
- **Suspending auto scaling** – Suspend auto scaling at the fleet location level without changing or deleting your auto scaling policies.

- **Scaling metrics** – Track a fleet's history of capacity and scaling events.

Monitoring Amazon GameLift

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

- Fleet, location, game session, and player session details
- Usage metrics
- Server process health
- Game session logs

For more information about monitoring in Amazon GameLift, see Monitoring Amazon GameLift.

Using other AWS resources

Your game servers and applications can communicate with other AWS resources. For example, you might use a set of web services for player authentication or social networking. For your game servers to access AWS resources that your AWS account manages, explicitly allow Amazon GameLift to access your AWS resources.

Amazon GameLift provides a couple of options for managing this type of access. For more information, see Communicate with other AWS resources from your fleets.

How players connect to games

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

For detailed information about creating and managing game sessions and player sessions with custom game servers, see Add Amazon GameLift to your game client. For information about connecting players to Realtime Servers, see Integrating a game client for Realtime Servers.

Amazon GameLift provides several features related to game and player sessions.

Host game sessions on best available resources across multiple locations

Choose from multiple options when configuring how Amazon GameLift selects resources to host new game sessions. If you're running fleets in multiple locations, then you can design **game session queues** that place new game sessions on any fleet regardless of location.

Control player access to game sessions

Configure game sessions to allow or deny join requests from new players, regardless of the number of players connected.

Use custom game and player data

Add custom data to game session objects and player session objects. Amazon GameLift passes game session data to a game server when starting a new game session. Amazon GameLift passes player session data to the game server when a player connects to the game session.

Filter and sort available game sessions

Use session search and sort to find the best possible match for a prospective player, or let player choose from a list of available game sessions. Use session search and sort to find game sessions based on session characteristics . You can also search and sort based on your own custom game data.

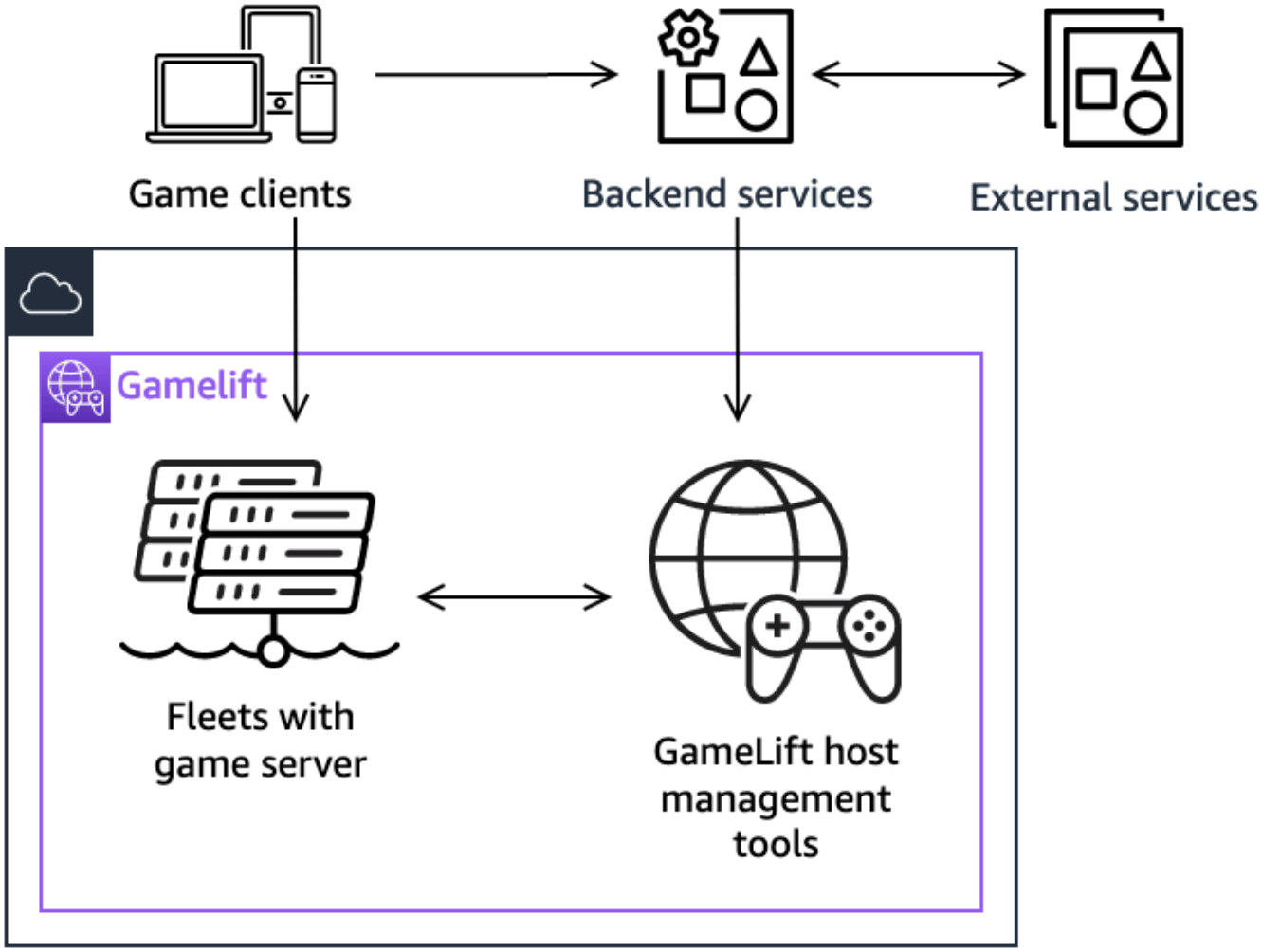
Track game and player usage data

Automatically have logs stored for completed game sessions. You can set up log storage when integrating Amazon GameLift into your game servers. For more information, see Logging server messages in Amazon GameLift.

Use the Amazon GameLift console to view detailed information about game sessions, including session metadata, settings, and player session data. For more information, see View data on game and player sessions and Metrics.

Game architecture with managed Amazon GameLift

The following diagram illustrates the key components of a game architecture that's hosted using the managed Amazon GameLift solution.



The key components of this architecture include the following:

Game clients

To join a game hosted on Amazon 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 Amazon GameLift through a backend service. The backend service makes requests to Amazon GameLift, and in response, the service receives game session information, which it relays back to the game client. The game client then connects to the game server. For more information, see Preparing games for Amazon GameLift.

Backend services

A backend service handles communication between game clients and Amazon GameLift by calling the Amazon GameLift service API operations in the AWS SDK. You can also use backend services for other game-specific tasks such as player authentication and authorization, inventory, or currency control. For more information, see Design your game client service.

External services

Your game can rely on an external service, such as for validating a subscription membership. An external service can pass information to your game servers through a backend service and Amazon GameLift.

Game servers

You upload your game server software to Amazon GameLift, and Amazon GameLift deploys it onto hosting machines to host game sessions and accept player connections. Game servers communicate with Amazon GameLift to start game sessions, validate newly connected players, and report the status of game sessions, player connections, and available resources.

Custom game servers communicate with Amazon GameLift by using the Amazon GameLift Server SDK. Game clients connect directly to a game server after receiving connection details from Amazon GameLift through a backend service. For more information, see Integrate games with custom game servers.

Realtime servers are game servers that run your custom script. When joining a game, a game client connects directly to a Realtime server using the Realtime Client SDK. For more information, see Integrating games with Amazon GameLift Realtime Servers.

Host management tools

When setting up and managing hosting resources, game owners use hosting management tools to manage game server builds or scripts, fleets, matchmaking, and queues. The Amazon GameLift tool set in the AWS SDK and the console provides multiple ways for you to manage your hosting resources. You can remotely access any individual game server for troubleshooting.

Setting up

Get help with setting up your AWS account to use Amazon GameLift to host your multiplayer games.

> **Tip**
>
> To try out Amazon GameLift game server hosting, see Getting started with Amazon GameLift.

Topics

Set up an AWS account

To start using Amazon GameLift, create and set up your AWS account. There's no charge to create an AWS account. This section walks you through creating your account, setting up your users, and configuring permissions.

Topics

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to an administrative user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create an administrative user

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.
2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create an administrative user

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to an administrative user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the administrative user

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Manage user permissions for Amazon GameLift

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

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

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

- Users and groups in AWS IAM Identity Center:

  Create a permission set. Follow the instructions in Create a permission set in the *AWS IAM Identity Center User Guide*.

- Users managed in IAM through an identity provider:

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

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

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

Set up programmatic access for users

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

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

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

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

If you use access keys, see Best practices for managing AWS access keys.

Set up programmatic access for your game

Most games use backend services to communicate with Amazon GameLift using the AWS SDKs. For example, you use a backend service (acting on behalf of game clients) to request game sessions,

place players into games, and other tasks. These services need programmatic access and security credentials to authenticate calls to Amazon GameLift service APIs.

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

- Create an IAM role with player user permissions and allow the player user to assume the role when needed. The backend service must include code to assume this role before making requests to Amazon GameLift. In accordance with security best practices, roles provide limited, temporary access. You can use roles for workloads running on AWS resources (IAM roles) or outside of AWS (IAM Roles Anywhere).
- Create an IAM user group with player user permissions and add your player user to the group. This option gives your player user long-term credentials, which the backend service must store and use when communicating with Amazon GameLift.

For permissions policy syntax, see Player user permission examples.

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

IAM permission examples for Amazon GameLift

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

If you're using Amazon GameLift FleetIQ as a standalone solution, see Set up your AWS account for Amazon GameLift FleetIQ.

Administrator permission examples

These examples give a user full access to manage Amazon GameLift game hosting resources.

Example Syntax for Amazon GameLift resource permissions

The following example extends access to all Amazon GameLift resources.

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

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

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

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

Example Syntax for Amazon GameLift resource and `PassRole` permissions

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

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

```
    {
      "Effect": "Allow",
      "Action": "gamelift:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "gamelift.amazonaws.com"
        }
      }
    }
  ]
}
```

Player user permission examples

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

Example Syntax for game session placement permissions

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

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

```
  }
}
```

Example Syntax for matchmaking permissions

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

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

Example Syntax for manual game session placement permissions

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

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

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

Set up an IAM service role for Amazon GameLift

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

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

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

Create a custom IAM role

Step 1: Create a permissions policy.

To use the JSON policy editor to create a policy

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

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

3. At the top of the page, choose **Create policy**.
4. In the **Policy editor** section, choose the **JSON** option.
5. Enter or paste a JSON policy document. For details about the IAM policy language, see IAM JSON policy reference.
6. Resolve any security warnings, errors, or general warnings generated during policy validation, and then choose **Next**.

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

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

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

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

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

```
{
```

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

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

Permission policy syntax

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

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

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

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

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

Development support with Amazon GameLift

Amazon GameLift provides a set of SDKs that you can use with your managed game hosting solutions. Use Amazon GameLift SDKs to add the necessary functionality to multiplayer game servers, game clients, and game services that need to interact with the Amazon GameLift hosting service.

For the latest information about Amazon GameLift SDK versions and SDK compatibility, see Amazon GameLift release notes.

For custom game servers

Create and deploy 64-bit custom game servers with the Amazon GameLift server SDK. Game servers that are integrated with the server SDK and deployed for hosting can communicate with the Amazon GameLift service to start and manage game sessions. For information on integrating the server SDK, see the topics in Preparing games for Amazon GameLift.

Development operating systems

- Windows
- Linux

Supported programming languages

Amazon GameLift provides the server SDK for the following languages. Download each server SDK package at Download Server SDKs. For detailed version-specific information, see the included Readme files in each package.

- C++ server SDK
  - SDK reference
  - SDK integration
- C# server SDK (versions may support .NET 4 and .NET 6)
  - SDK reference
  - SDK integration
- Go
  - SDK reference
  - SDK integration

Supported game engines

Use language-specific SDKs with any engines that support C++, C#, or Go libraries. In addition, Amazon GameLift provides these game engine plugins:

- **Unity**
  - C# server SDK plugin for Unity is a lightweight plugin with pre-built libraries that you can install using the Unity package manager. Use this plugin with the following Unity versions: 2020.3 LTS, 2021.3 LTS and 2022.3 LTS for Windows and Mac OS. It supports Unity's .NET Framework and .NET Standard profiles, with .NET Standard 2.1 and .NET 4.x.
    - Integrate Amazon GameLift into a Unity project
  - Standalone plugin for Unity 2021.3 LTS and 2022.3 LTS is a full-featured plugin with the C# SDK libraries built for Unity and GUI elements for configuring and deploying Amazon GameLift resources for hosting.
    - Amazon GameLift plugin for Unity guide for server SDK 5.x
    - Amazon GameLift server SDK reference for C#
- **Unreal Engine**
  - C++ server SDK plugin for Unreal is a lightweight plugin consisting of C++ Unreal source code that you can build into libraries for use with Unreal Engine versions 4, 5, and 5.1.
    - Integrate Amazon GameLift into an Unreal Engine project

  - Amazon GameLift Unreal Engine server SDK 5.x reference
- Standalone plugin for Unreal Engine 5.0, 5.1, and 5.2 is a full-featured plugin with the C++ for Unreal server SDK libraries and AWS SDK. The plugin is installed in the Unreal editor, with UI elements and supporting materials for configuring and deploying Amazon GameLift resources for hosting.
  - Integrating games with the Amazon GameLift plugin for Unreal Engine
  - Amazon GameLift Unreal Engine server SDK 5.x reference

Game server operating systems

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

- Windows Server 2016
- Amazon Linux 2023
- Amazon Linux 2 (AL2)
- Windows Server 2012 (see Amazon GameLift FAQ for Windows 2012)
- Amazon Linux (AL1) (see Amazon GameLift FAQ for AL1)

For custom client services

Create 64-bit custom client services using the AWS SDK with the Amazon GameLift API. This SDK enables client services to manage game sessions and join players to games that are hosted on Amazon GameLift. To get started, download the AWS SDK. For more information about using the SDK with Amazon GameLift, see the Amazon GameLift API Reference.

For Realtime Servers

Configure and deploy Realtime servers to host your multiplayer games. To allow your game clients to connect to Realtime servers, use the Amazon GameLift Realtime Client SDK. Game clients use this SDK to exchange messages with a Realtime server and with other game clients that connect to the server. To get started, download the Amazon GameLift Realtime Client SDK. For configuration information, see Integrating a game client for Realtime Servers.

SDK support

The Realtime Client SDK contains source for the following languages:

- C# (.NET)

Development environments

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

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

Game server operating systems

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

- Amazon Linux
- Amazon Linux 2

Manage your game hosting costs

Your AWS bill reflects your game hosting costs. You can view estimated charges for the current month, and final charges for previous months on the Billing console at https://console.aws.amazon.com/billing/. For more information about tools and resources to help you manage your AWS costs, see the AWS Billing User Guide. This guide can help you review your resource consumption, establish future usage, and determine your scaling needs.

In particular, consider these tips to help you manage the cost of Amazon GameLift services.

Create billing alerts to monitor usage

Set up an AWS Free Tier usage alert to notify you when your usage is nearing or exceeding the Free Tier limits for Amazon GameLift and other AWS services. You can configure the alerts to take action based on your usage levels. For example, you can automatically set your budget to zero when your reach a Free Tier limit.

You can also set Amazon CloudWatch billing alerts to get notifications when usage hits custom thresholds.

For more information, see these topics in the *AWS Billing User Guide*:

- Tracking your AWS Free Tier usage
- Billing alert preferences

Track costs per Amazon GameLift fleet

Use AWS cost allocation tags to organize and track your game hosting costs based on Amazon GameLift Amazon EC2 fleets and other EC2 resources. By tagging your fleets, either individually or by groups, you can create cost allocation reports that categorize costs based on the assigned tag. You can use this type of report to identify how fleets are contributing to your hosting costs. You can also use tags to filter views in AWS Cost Explorer.

For more information, see these topics:

- Using AWS cost allocation tags, *AWS Billing User Guide*
- Analyzing your costs with AWS Cost Explorer, *AWS Cost Management User Guide*

Set unused fleet capacity to zero

Fleets can continue to incur costs even when they're not in use hosting game sessions. To avoid incurring unnecessary charges, scale your fleet down to zero when not in use. If you use auto scaling, suspend this activity and manually set the fleet capacity.

Amazon GameLift hosting locations

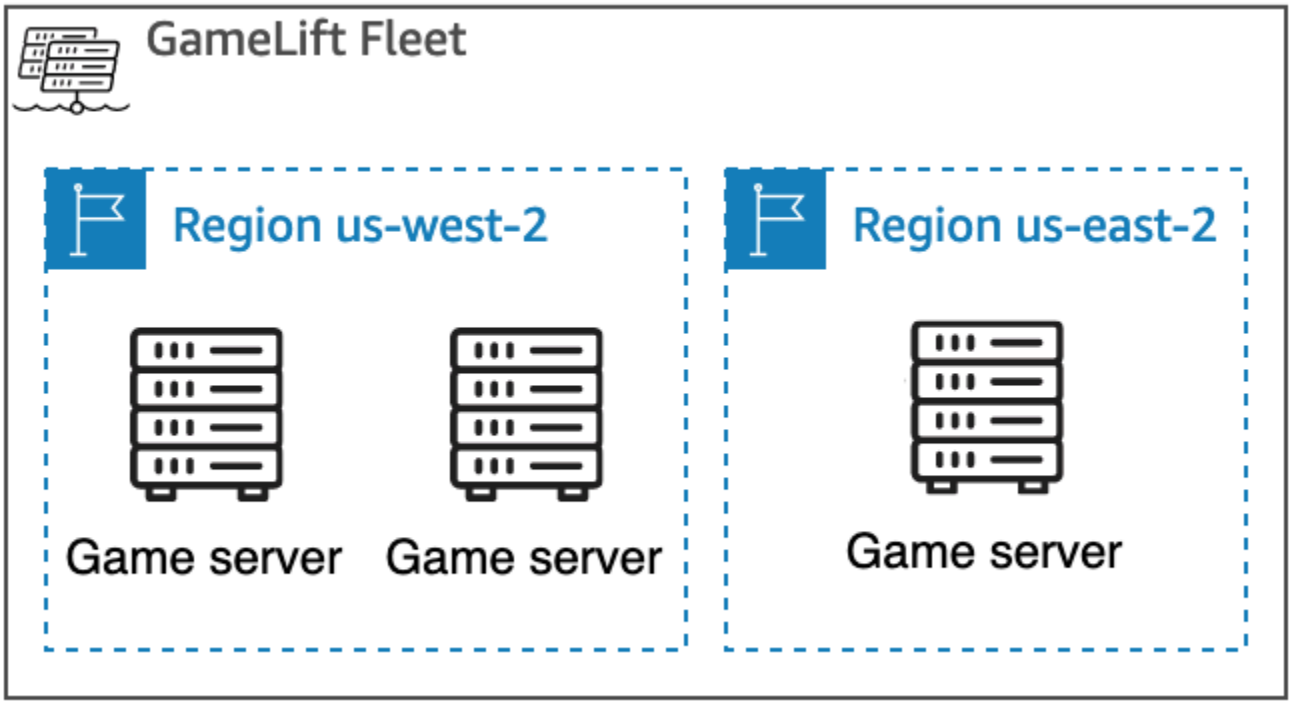
Amazon GameLift is available in multiple AWS Regions and Local Zones. For a complete list of locations, see Amazon GameLift endpoints and quotas in the *AWS General Reference*.

Amazon GameLift hosting

When you create a Amazon GameLift fleet, Amazon GameLift creates the fleet's resources in your current AWS Region. Amazon GameLift calls this Region the fleet's *home Region*. To manage a fleet, access it from its home Region.

Multi-location fleets deploy instances to other locations in addition to the fleet's home Region. 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 Region or Local Zone that Amazon GameLift supports. The following diagram depicts a multi-location fleet

with resources in two Regions. In the diagram, the `us-west-2` Region includes two game servers, and the `us-east-2` Region has one game server.



If you choose to use a multi-location fleet with instances in Regions that aren't enabled by default, you must enable those Regions in your AWS account. Also, your Amazon GameLift administrator policy must allow the `ec2:DescribeRegions` action. For more information about Regions that aren't enabled by default and how to enable them, see Managing AWS Regions in the *AWS General Reference*. For a policy example with Regions that aren't enabled by default, see Administrator permission examples.

Important

To use Regions that aren't enabled by default, enable them in your AWS account.

- Fleets with Regions that aren't enabled that you created before February 28, 2022 are unaffected.
- To create new multi-location fleets or to update existing multi-location fleets, first enable any Regions that you choose to use.

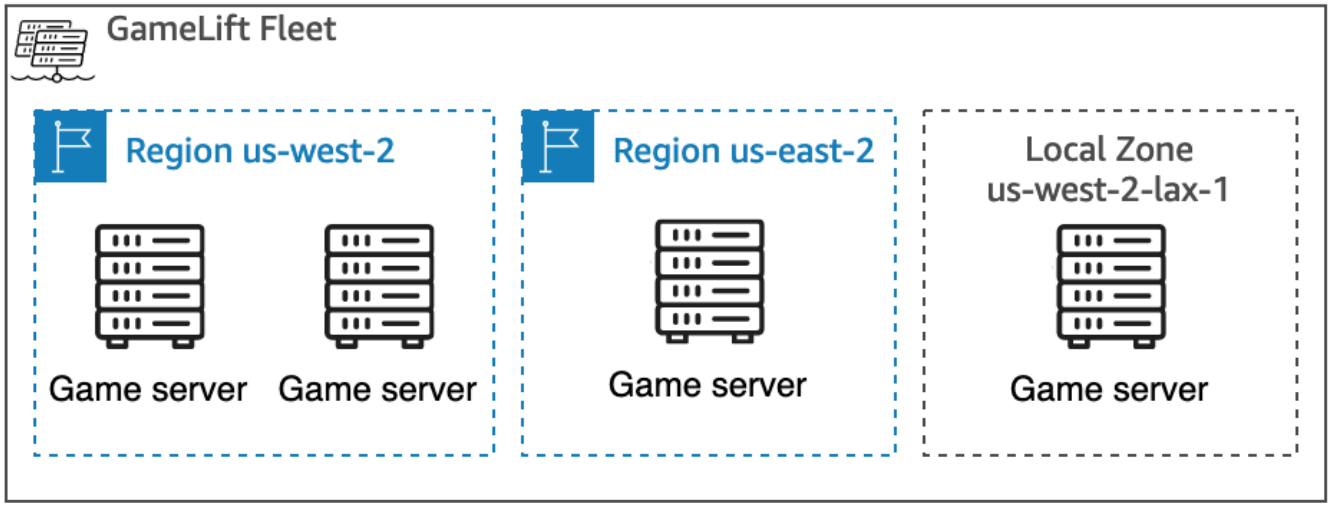
For game session placement, you can create game session queues in any location that Amazon GameLift supports. Amazon GameLift places game sessions from the location where you created the queue.

Local Zones

A *Local Zone* is an extension of an AWS Region in geographic proximity to your users. Local Zones have their own connections to the internet. Local Zones also support AWS Direct Connect so that resources created in a Local Zone can serve local users with low-latency communications. For more information, see AWS Local Zones.

The code for a Local Zone is its Region code, followed by an identifier that indicates its physical location. For example, the `us-west-2-lax-1` Local Zone is in Los Angeles. For a list of available Local Zones, see Available Local Zones.

Amazon GameLift hosts your games in each of the locations that you choose for your fleet. The following diagram depicts a fleet with two game servers in the `us-west-2` Region, one game server in the `us-east-2` Region, and one game server in the `us-west-2-lax-1` Local Zone.



Available Local Zones

The following table lists the available Local Zones and their physical locations.

Local Zone	Location (metro area)	
`us-east-1-atl-1`	Atlanta	

Local Zone	Location (metro area)	
us-east-1-chi-1	Chicago	
us-east-1-dfw-1	Dallas	
us-east-1-iah-1	Houston	
us-east-1-mci-1	Kansas City	
us-west-2-den-1	Denver	
us-west-2-lax-1	Los Angeles	
us-west-2-phx-1	Phoenix	

Amazon GameLift Anywhere

You can use Amazon GameLift Anywhere to create fleets with your own hardware, and manage your game builds, scripts, game servers, and clients using Amazon GameLift. Amazon GameLift Anywhere is available in all Regions that Amazon GameLift supports. For more information about creating an Anywhere fleet and testing your game server integration, see Create a Amazon GameLift Anywhere fleet and Test your integration using Amazon GameLift Anywhere fleets.

With Amazon GameLift Anywhere you create custom locations that represent the physical location of the hardware you are using to host your Amazon GameLift integrated game servers.

Amazon GameLift FlexMatch

For FlexMatch, you can host match-generated game sessions in any location that Amazon GameLift supports. Actual matchmaking activity takes place in the AWS Region where you chose to create your matchmaker resources. Amazon GameLift routes match requests to the matchmaker and processes them in that location. For more information about Amazon GameLift FlexMatch, see What is Amazon GameLift FlexMatch?

AWS Regions that support FlexMatch resources

Amazon GameLift in China

When using Amazon GameLift for resources in the China (Beijing) Region, operated by Sinnet, or the China (Ningxia) Region, operated by NWCD, you must have a separate AWS (China) account. Note that some features are unavailable in the China Regions. For more information about using Amazon GameLift in these Regions, see the following resources:

- Amazon Web Services in China
- Amazon GameLift (Getting Started with Amazon Web Services in China)

Getting started with Amazon GameLift

We recommend that you try the following examples before you use Amazon GameLift for your own game. The custom game server example gives you experience with game hosting in the Amazon GameLift console. The Realtime Servers example shows you how to prepare a game for hosting using Realtime Servers.

To get started with Amazon GameLift for your own game, see Amazon GameLift managed hosting roadmap.

Custom game server example

This example demonstrates a live custom game on Amazon GameLift. The example walks you through the following steps:

- Creating an example game build.
- Creating a fleet to run the game server.
- Connecting to the game server from the example game client.
- Reviewing fleet and game session metrics.

After these steps, you can start up multiple game clients and play the game to generate hosting data. Then, you can explore the Amazon GameLift console to view your hosting resources, track metrics, and experiment with ways to scale hosting capacity.

To get started, sign in to the Amazon GameLift console.

Realtime Servers example game

This example is a complete multiplayer game named Mega Frog Race, with source code included. The example shows how to integrate your game client with Realtime Servers. You can also use this example game as a starting point to experiment with other Amazon GameLift features such as FlexMatch.

For a hands-on tutorial, see Creating Servers for Multiplayer Mobile Games with Just a Few Lines of JavaScript on the AWS for Games Blog.

For the source code of Mega Frog Race, see the GitHub repository.

The source code includes the following parts:

- Game client – A source code for the C++ game client, created in Unity. The game client gets game session connection information, connects to the server, and exchanges updates with other players.
- Backend service – A source code for an AWS Lambda function that manages direct API calls to Amazon GameLift.
- Realtime script – A source script file that configures a fleet of Realtime Servers for the game. This script includes the minimum configuration required for Realtime Servers to communicate with Amazon GameLift and to host games.

Amazon GameLift managed hosting roadmap

This topic helps you choose from the different Amazon GameLift hosting options for your session-based multiplayer game. The rest of the topics in this section walk you through how to use Amazon GameLift for your managed hosting.

Before you start preparing to launch your game to production, fill out the launch questionnaire to begin working with the Amazon GameLift team.

Topics

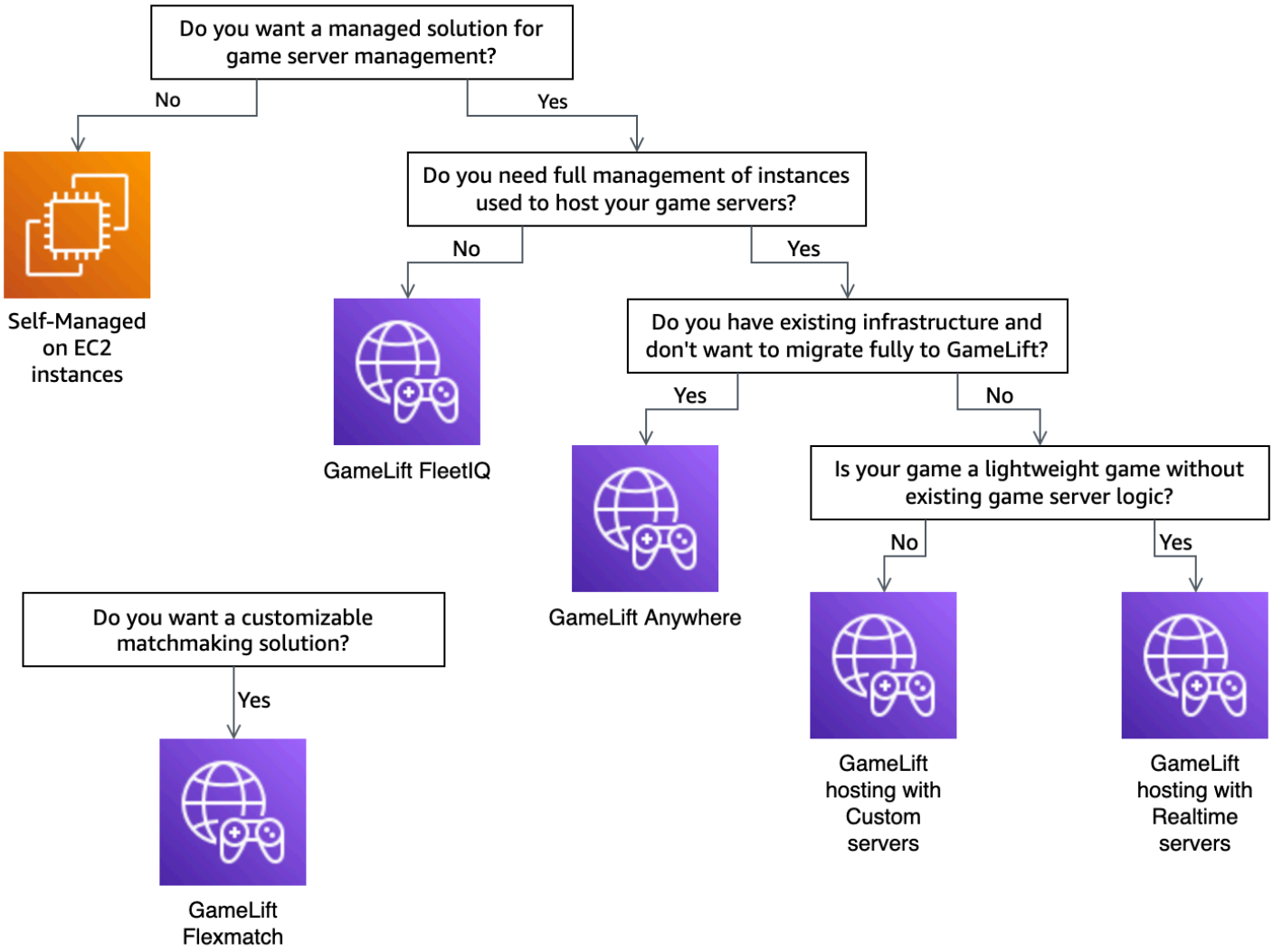
Choose a hosting option

The following flowchart asks questions to lead you to the correct Amazon GameLift solution for your use case.

1. Do you want a managed solution for game server management?
   - **Yes** – Continue to step two.
   - **No** – Consider self-managed game servers on Amazon EC2 instances.
2. Do you need full control of the instances hosting your game servers?
   - **Yes** – Consider Amazon GameLift FleetIQ.
   - **No** – Continue to step 3.
3. Do you have existing infrastructure you want to use with Amazon GameLift?
   - **Yes** – Consider Amazon GameLift Anywhere.
   - **No** – Continue to step four.

4. Is your game lightweight without existing game server logic?

   - **Yes** – Consider Realtime servers.
   - **No** – Consider custom servers.



Here's some more information about some of the Amazon GameLift hosting options mentioned in the flowchart:

Amazon GameLift Anywhere

Use Amazon GameLift Anywhere to host your games on your own hardware with the benefit of Amazon GameLift management tools. You can also use Anywhere fleets to test your game servers iteratively. For more information, see Create a Amazon GameLift Anywhere fleet.

Managed Amazon GameLift

There are two options for managed Amazon GameLift hosting:

Custom servers – Amazon GameLift hosts your custom server that runs your game server binary.

Realtime Servers – Amazon GameLift hosts your lightweight game server.

Amazon GameLift FleetIQ

In the flowchart, a *lift and shift* migration refers to a migration when you can't make changes to the game architecture. Using Amazon GameLift FleetIQ requires fewer changes to your existing deployment and provides Amazon GameLift tools for fleet management. For more information, see the Amazon GameLift FleetIQ Developer Guide.

If you decide to use Amazon GameLift Anywhere or managed Amazon GameLift, continue to Prepare your game for Amazon GameLift.

Prepare your game for Amazon GameLift

This topic describes the steps for preparing your multiplayer game for integration with managed Amazon GameLift hosting. To prepare your game, you must activate communication between it and Amazon GameLift.

Prepare your custom game server

To start and stop game sessions, and to perform other tasks, a game server must be able to notify Amazon GameLift about its status. To activate communication with Amazon GameLift, add code to your game server project. For more information, see Integrate games with custom game servers.

1. **Prepare your custom game server for hosting on Amazon GameLift.**
   - Get the Amazon GameLift Server SDK and build it for your preferred programming language and game engine.
   - Add code to your game server project to activate communication with Amazon GameLift.
2. **Prepare your game client to connect to Amazon GameLift hosted game sessions.**
   - Add the AWS SDK to your backend service and game client project. For more information, see Download Amazon GameLift SDKs for client services.

- Add functionality to retrieve information on game sessions, place new game sessions, and reserve space for players on a game session.
- (Optional) Use FlexMatch for player matchmaking. For more information, see FlexMatch integration with Amazon GameLift hosting.

Prepare your Realtime server

Amazon GameLift Realtime Servers provides a lightweight server solution that you can configure to fit your game. A Realtime server provides the same benefits that Amazon GameLift offers to game servers, but with reduced game server customizability.

Create a Realtime script for hosting on Amazon GameLift.

Realtime scripts contain your server configuration and optional custom game logic. Realtime servers are built to start and stop game sessions, accept player connections, and manage communication with Amazon GameLift and between players in a game. There are also hooks for you to add custom server logic for your game. Realtime servers use Node.js and JavaScript. For more information, see Creating a Realtime script and Test your integration with Amazon GameLift.

Test your integration with Amazon GameLift

Amazon GameLift supports fast iteration when testing your game servers. This topic guides you through the types of testing available.

Custom game servers

Use Amazon GameLift to integrate hardware anywhere in your environment into your Amazon GameLift game hosting architecture. Amazon GameLift Anywhere registers your hardware with Amazon GameLift in an Anywhere fleet, so that you can test using your own local development computer. For more information about testing with Amazon GameLift Anywhere, see Test your integration using Amazon GameLift Anywhere fleets. For more information about using Amazon GameLift Anywhere for hosting your games with on-premises solutions, see Choosing Amazon GameLift compute resources.

Realtime Servers

With Realtime Servers, you can update your scripts at any time. When you update a Realtime script, Amazon GameLift distributes the new version to your hosting resources within minutes. After

Amazon GameLift deploys the new script, all new game sessions use the new script version. After Amazon GameLift deploys the new script, you can begin testing immediately. For more information about Realtime Servers see, Integrating games with Amazon GameLift Realtime Servers

Plan and deploy your Amazon GameLift resources

Use the following tips to help plan your global Amazon GameLift resources deployment. For information about where you can host your games with Amazon GameLift, see Amazon GameLift hosting locations.

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

- **Package and upload your game server to Amazon GameLift or to your hardware.** When uploading your server to Amazon GameLift, you upload it only to the home AWS Region of your fleet. Amazon GameLift automatically distributes the server to other locations in the fleet. For more information, see Uploading builds and scripts to Amazon GameLift.
- **Design and deploy a Amazon GameLift fleet for your game.** Determine the type of computing resources to use, which locations to deploy to, whether to use queues, and other options. For more information, see Amazon GameLift fleet design guide.
- **Create queues to manage new game session placements and Spot Instance strategies.** For more information, see Design a game session queue.
- **Use automatic scaling to manage your fleet's hosting capacity for expected player demand.** For more information, see Scaling Amazon GameLift hosting capacity.
- **Use FlexMatch matchmaking rules for your game.** For more information, see FlexMatch integration with Amazon GameLift hosting.

Automatically deploy your Amazon GameLift resources

To streamline the global deployment of your Amazon GameLift resources, we recommend that you use infrastructure as code (IaC) to define the resources. Because Amazon GameLift supports AWS CloudFormation templates, you can set parameters in the templates for any deployment-specific configurations.

To manage the deployment of your AWS CloudFormation stacks, we also recommend using continuous integration and continuous delivery (CI/CD) tools and services such as AWS CodePipeline. These help you deploy automatically or with approval whenever you build game server binary.

The following are some common steps of Amazon GameLift resources deployment for a new game server version that you can automate using a CI/CD tool or service:

- Building and testing your game server binary.
- Uploading the binary to Amazon GameLift or your hardware.
- Deploying new fleets in the new build.
- After you deploy the new fleets, removing the previous version fleets from your Amazon GameLift queue and replacing them with the new fleets.
- After the previous version fleets successfully end all game sessions, deleting the AWS CloudFormation stacks of those fleets.

You can also use the AWS Cloud Development Kit (AWS CDK) to define your Amazon GameLift resources. For more information about the AWS CDK, see the AWS Cloud Development Kit (AWS CDK) Developer Guide.

Design your game client service

We recommend that you implement a game client service that authenticates your players and communicates with the Amazon GameLift API. By implementing a custom game client service, you can:

- Customize authentication for your players.
- Control how Amazon GameLift matches and starts game sessions.
- Use your player database for player attributes such as skill rating for matchmaking instead of trusting the client.

Using a game client service also reduces security risks introduced by game clients interacting directly with your Amazon GameLift API.

Authenticating your players

You can use Amazon Cognito and player session IDs to authenticate your game clients. To manage the lifecycle and properties of your player identities, use Amazon Cognito user pools.

If you prefer, build a custom identity solution and host it on AWS. You can also use Lambda authorizers for custom authorization logic with API Gateway.

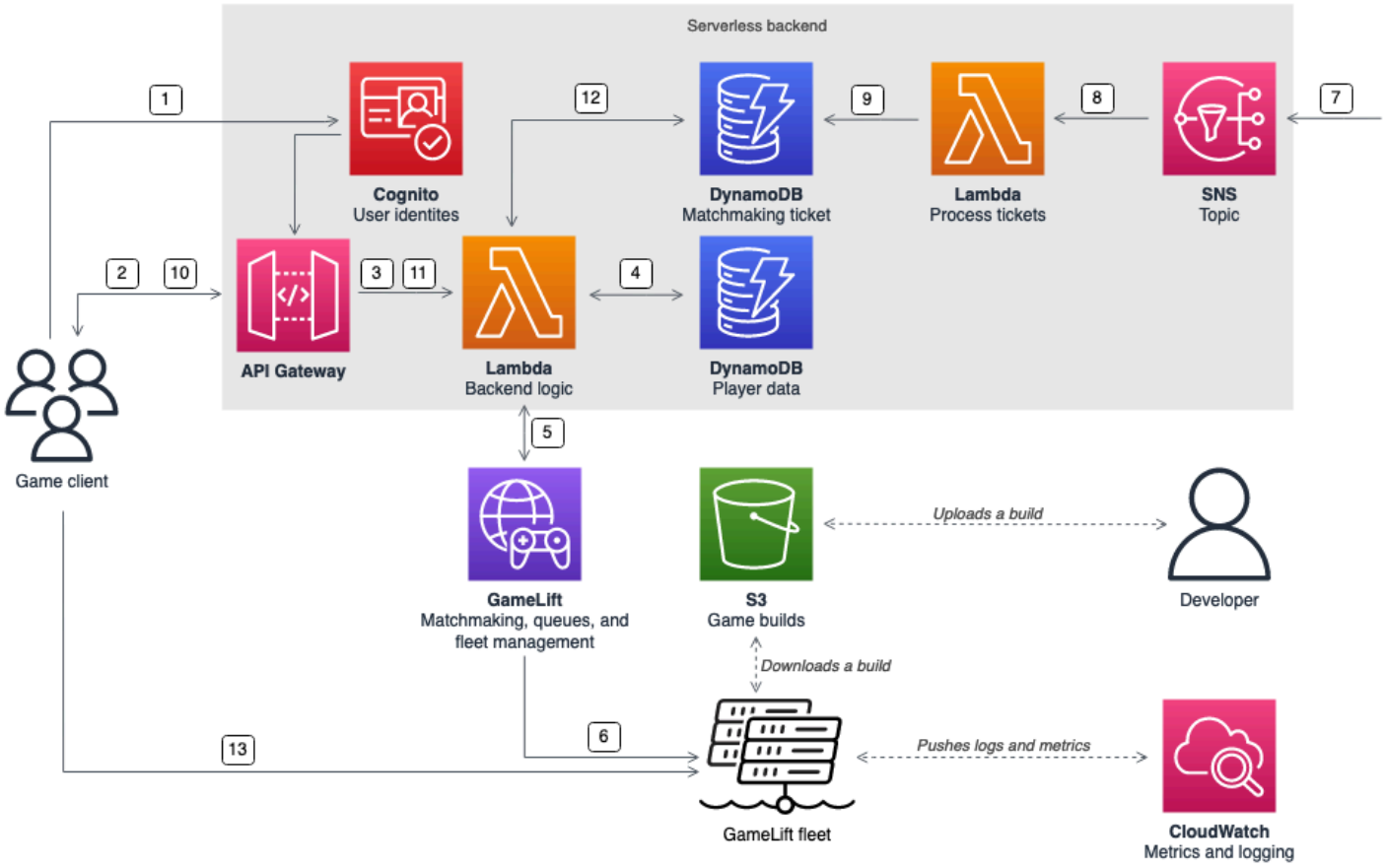
Additional resources:

- Using identity pools (federated identities) (Amazon Cognito Developer Guide)
- Getting started with user pools (Amazon Cognito Developer Guide)
- How to Set Up Player Authentication with Amazon Cognito (AWS for Games Blog)

Standalone game session servers with a serverless backend

Using a serverless client service architecture, the backend can view the status of matchmaking tickets from a highly scalable database instead of by directly accessing the Amazon GameLift API.

The following diagram shows a serverless backend built with AWS services that matches players into games running on Amazon GameLift fleets. The following list provides a description for each numbered callout in the diagram. To try out this example, see Multiplayer Session-based Game Hosting on AWS on GitHub.



1. The game client requests an Amazon Cognito user identity from an Amazon Cognito identity pool.

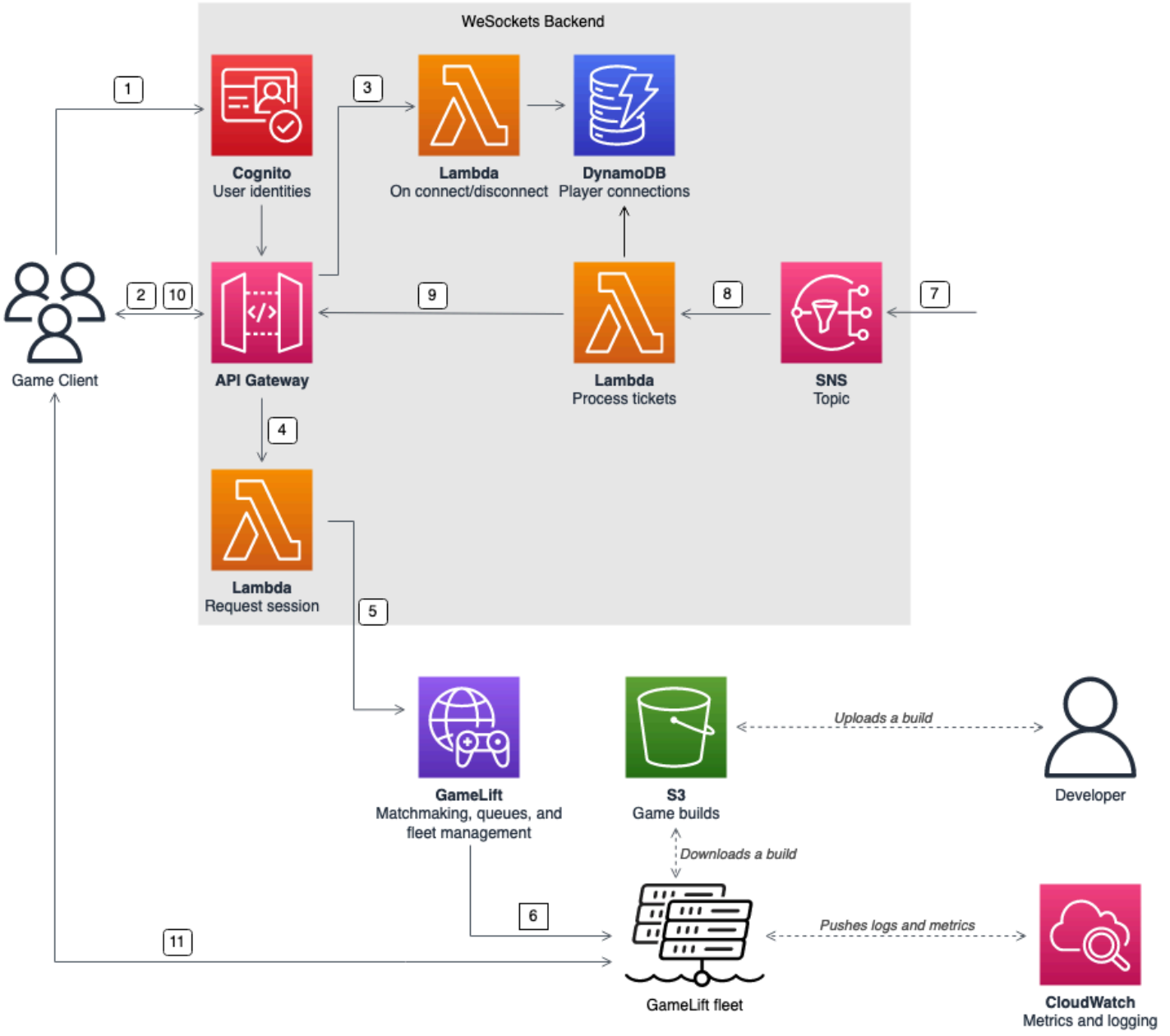
2. The game client receives temporary access credentials and requests a game session through an Amazon API Gateway API.
3. API Gateway invokes an AWS Lambda function.
4. The Lambda function requests player data from an Amazon DynamoDB NoSQL table. The function provides the Amazon Cognito identity in the request context data.
5. The Lambda function requests a match through Amazon GameLift FlexMatch matchmaking.
6. FlexMatch matches a group of players with suitable latency, and then requests a game session placement through a Amazon GameLift queue. The queue has fleets with one or more AWS Region locations in it.
7. After Amazon GameLift places the session on one of the fleet's locations, Amazon GameLift sends an event notification to an Amazon Simple Notification Service (Amazon SNS) topic.
8. A Lambda function receives the Amazon SNS event and processes it.
9. If the matchmaking ticket is a `MatchmakingSucceeded` event, then the Lambda function writes the result, along with the port and IP address of the game server, to a DynamoDB table.
10. The game client makes a signed request to API Gateway to view the status of the matchmaking ticket on a specific interval.
11. API Gateway uses a Lambda function that checks the matchmaking ticket status.
12. The Lambda function checks the DynamoDB table to see if the ticket is successful. If it has succeeded, the function sends the game server's port and IP address, along with the player session ID, back to the client. If the ticket hasn't succeeded, the function sends a response verifying that the match isn't ready yet.
13. The game client connects to the game server using TCP or UDP by using the port and IP address that the backend service provides. The game client then sends the player session ID to the game server, which then validates the ID using the Amazon GameLift Server SDK.

Standalone game session servers with a WebSocket-based backend

Using an Amazon API Gateway WebSocket-based architecture, you can make matchmaking requests with WebSockets and send push notifications for matchmaking completion using server-initiated messages. This architecture improves performance by having two-way communication between the client and the server.

For more information about using API Gateway WebSock APIs, see Working with WebSocket APIs.

The following diagram shows a WebSocket-based backend architecture that uses API Gateway and other AWS services to match players into games running on Amazon GameLift fleets. The following list provides a description for each numbered callout in the diagram.



1. The game client requests an Amazon Cognito user identity from an Amazon Cognito identity pool.
2. The game client signs a WebSocket connection to an API Gateway API with the Amazon Cognito credentials.

3. API Gateway calls an AWS Lambda function on the connection. The function stores the connection information in an Amazon DynamoDB table.
4. The game client sends a message to a Lambda function, through the API Gateway API over the WebSocket connection, to request a session.
5. A Lambda function receives the message and then requests a match through Amazon GameLift FlexMatch matchmaking.
6. After FlexMatch matches a group of players, FlexMatch requests a game session placement through a Amazon GameLift queue.
7. After Amazon GameLift places the session on one of the fleet's locations, Amazon GameLift sends an event notification to an Amazon Simple Notification Service (Amazon SNS) topic.
8. A Lambda function receives the Amazon SNS event and processes it.
9. If the matchmaking ticket is a `MatchmakingSucceeded` event, then the Lambda function requests the correct player connection from DynamoDB. The function then sends a message to the game client through the API Gateway API over the WebSocket connection. In this architecture, the game client doesn't actively poll the status of matchmaking.
10. The game client receives the port and IP address of the game server, along with the player session ID, through the WebSocket connection.
11. The game client connects to the game server using TCP or UDP using the port and IP address that the backend service provides. The game client also sends the player session ID to the game server, which then validates the ID using the Amazon GameLift Server SDK.

Set up metrics and logging for Amazon GameLift

You can use data collected from your Amazon GameLift game servers and resources to help identify anomalies. You can also use metrics to help improve performance.

Key areas to observe for Amazon GameLift include:

- **Amazon GameLift service metrics** – Amazon GameLift provides Amazon CloudWatch metrics on your resources including game servers, fleets, queues, and FlexMatch. You can find these metrics in the Amazon GameLift console and the CloudWatch console. For more information about Amazon GameLift metrics in CloudWatch, see Monitor Amazon GameLift with Amazon CloudWatch.
- **Game server metrics** – Amazon GameLift can't access your game server metrics. However, you can send custom metrics to CloudWatch directly from your game server by using the CloudWatch

agent. You can also use the fleet AWS Identity and Access Management (IAM) role and the AWS SDK to send metrics directly to CloudWatch. For an example of how to configure metrics, see Multiplayer Session-based Game Hosting on AWS on GitHub. This repository includes an example CloudWatch agent configuration and code for a C# StatsD client.

- **Game server logs** – To configure your game server log files on the game server, use the Amazon GameLift Server SDK configuration. You can also use Amazon CloudWatch Logs as a real-time log management solution, and you can configure logs with the CloudWatch agent. For more information, see Logging server messages in Amazon GameLift.

Game launch checklists

You can use these checklists to validate the phases of deployment of your game. In the checklists, items marked **[Critical]** are critical for your production launch.

Topics

Onboarding

Use the following checklist to track items for onboarding your game for Amazon GameLift hosting. Items marked **[Critical]** are critical for your production launch.

- **[Critical]** Fill out the Amazon GameLift onboarding questionnaire in the Amazon GameLift console.
- **[Critical]** Design and implement a backend service for game clients to interact with your game servers.
- **[Critical]** Create AWS Identity and Access Management (IAM) roles that you provide to Amazon GameLift server instances for access to other AWS resources.
- **[Critical]** Design and implement failover to other AWS Regions for FlexMatch and queues.
- Plan the rollout of fleets to your target locations, considering your game's queue and fleet structure.

- Automate your deployment using infrastructure as code (IaC) with AWS CloudFormation and the AWS Cloud Development Kit (AWS CDK).
- Collect logs and analytics using Amazon CloudWatch and Amazon Simple Storage Service (Amazon S3).

Testing

Use the following checklist to track testing items while developing your game with Amazon GameLift hosting. Items marked **[Critical]** are critical for your production launch.

- **[Critical]** Complete the launch questionnaire, and submit the completed questionnaire to the Amazon GameLift launch team. You can find the launch questionnaire in the Amazon GameLift console.
- **[Critical]** Request increases for Amazon GameLift service quotas and other AWS service quotas so that your live environment can scale up to production needs.
- **[Critical]** Verify that the open ports on live fleets match the range of ports that your servers could use.
- **[Critical]** Close RDP port 3389 and SSH port 22.
- Develop a plan for the DevOps management of your game. If you're using Amazon CloudWatch Logs or Amazon CloudWatch custom metrics, define alarms for severe or critical problems on the server fleet. Simulate failures and test the runbooks.
- Verify that the number of servers running on an instance at full usage are within the capabilities of the server instance type.
- Tune your scaling policy to be more conservative at first and provide more idle capacity than you think you need. You can optimize for cost later. Consider the use of target-based scaling policy with 20 percent idle capacity.
- Use FlexMatch latency rules to match players who are geographically near the same AWS Region. Test how this behaves under load with synthetic latency data from your load test client.
- Load test your player authentication and game session infrastructure to see if it scales effectively to meet demand.
- Verify that a server left running for several days can still accept connections.
- Raise your AWS Support plan level to Business or Enterprise so that AWS can respond to you during problems or outages.

Launch

Use the following checklist to track launch items for your game hosted on Amazon GameLift. 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 doesn't stop active game sessions.
- **[Critical]** Set fleet maximum sizes high enough to accommodate peak anticipated demand, at minimum. We recommend that you double your maximum size for unanticipated demand.
- Encourage your whole dev team to participate in the launch event and monitor your game launch in a launch room.
- Monitor player latency and player experience.

Post-launch

Use the following checklist to track post-launch items for your game hosted on Amazon GameLift.

- Tune scaling rules to minimize idle capacity.
- Modify FlexMatch rules or add additional locations based on your latency requirements.
- Optimize the server executable, as its performance efficiency directly affects the fleet costs. To run more game sessions with the same infrastructure, increase the number of server processes per instance.
- Use your analytics data to drive continued development, improve player experience and game longevity, and optimize monetization.

Preparing games for Amazon GameLift

To prepare your multiplayer game for hosting on Amazon GameLift, set up communication between your game and Amazon GameLift. The topics in this section provide detailed help for integrating your game with Amazon GameLift, custom game servers, and Realtime Servers, and for adding matchmaking with FlexMatch.

Topics

Integrate games with custom game servers

Amazon GameLift provides a full tool set for preparing your multiplayer games and custom game servers to run on Amazon GameLift. The Amazon GameLift SDKs contain libraries needed for game clients and servers to communicate with Amazon GameLift. For more information about the SDKs and where to get them, see Development support with Amazon GameLift.

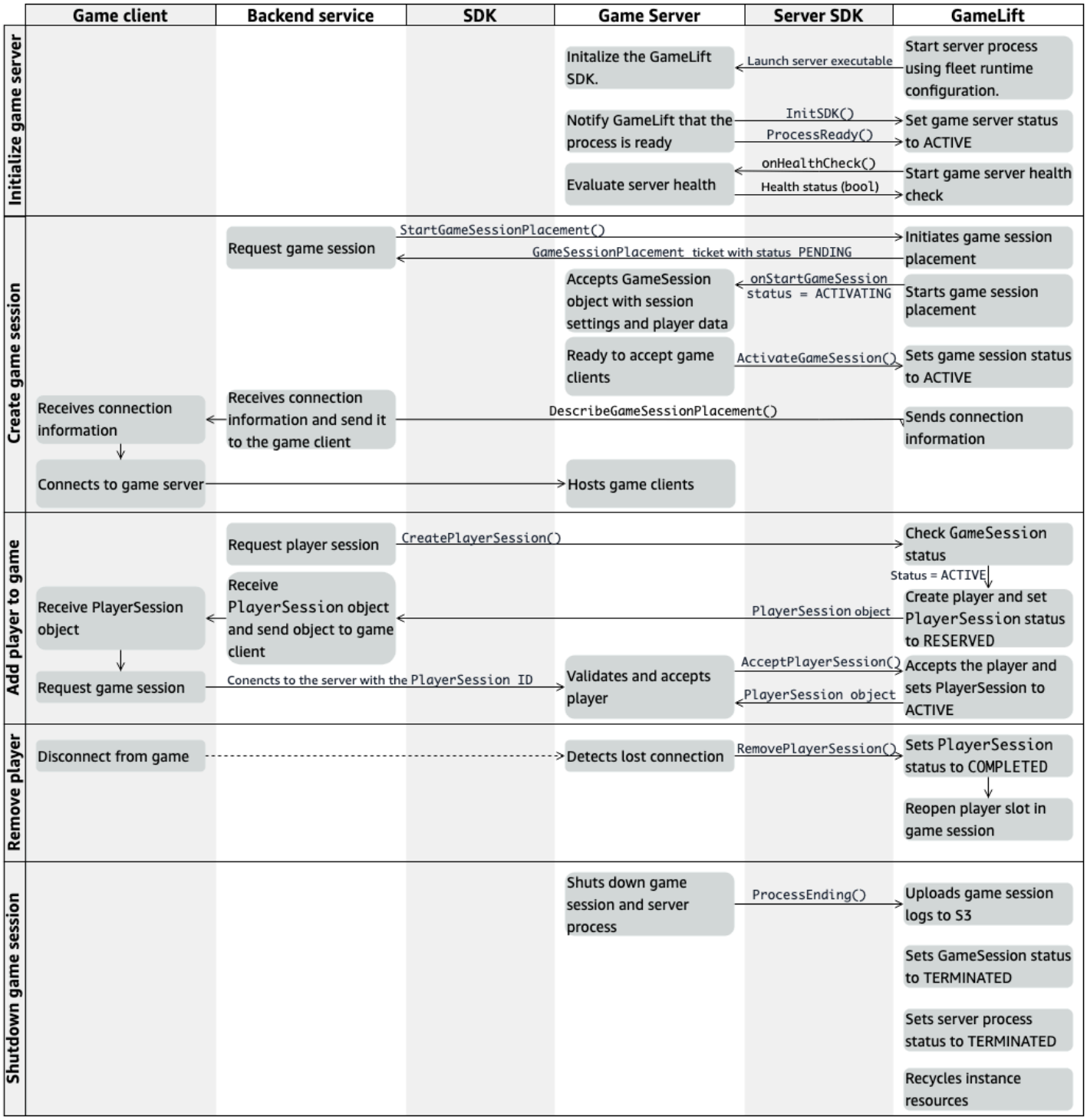
The topics in this section contain detailed instructions about how to add Amazon GameLift functionality to your game client and game server before deploying on Amazon GameLift. For a complete roadmap to getting your game up and running on Amazon GameLift, see Amazon GameLift managed hosting roadmap.

Topics

Amazon GameLift and game client server interactions

This topic describes the interactions between the game client, a backend service, a game server, and Amazon GameLift.

The following diagram illustrates interactions between the game client, backend service, Amazon GameLift SDK, managed EC2 game server, Amazon GameLift server SDK, and Amazon GameLift. For a detailed description of the interactions shown, see the following sections on this page.



Initialize a game server

The following steps describe the interactions that occur when you prepare your game server to host game sessions.

1. Amazon GameLift launches the server executable on an Amazon Elastic Compute Cloud (Amazon EC2) instance.
2. The game server calls:

   a. `InitSDK()` to initialize the server SDK.

   b. `ProcessReady()` to communicate game session readiness, connection information, and location of game session log files.

   The server process then waits for a callback from Amazon GameLift.
3. Amazon GameLift updates the status of the server process to `ACTIVE` to enable game session placement.
4. Amazon GameLift begins calling the on`HealthCheck` callback and continues to call it periodically while the server process is active. The server process can report healthy or not healthy within one minute.

Create a game session

After you've initialized your game server, the following interactions occur when you create game sessions to host your players.

1. The backend service calls the SDK operation `StartGameSessionPlacement()`.
2. Amazon GameLift creates a new `GameSessionPlacement` ticket with status `PENDING` and returns it to the backend service.
3. The backend service obtains a placement ticket status from a queue. For more information, see Set up event notification for game session placement.
4. Amazon GameLift starts game session placement by selecting an appropriate fleet and searching for an active server process in a fleet with `0` game sessions. When Amazon GameLift locates a server process, Amazon GameLift does the following:

   a. Creates a `GameSession` object with the game session settings and player data from the placement request with an `ACTIVATING` status.

   b. Invokes the on`StartGameSession` callback on the server process. Amazon GameLift passes information to the `GameSession` object indicating that the server process may set up the game session.

   c. Changes the server process's number of game sessions to 1.

5. The server process runs the onStartGameSession callback function. When the server process is ready to accept player connections, it calls ActivateGameSession() and waits for player connections.
6. Amazon GameLift updates the GameSession object with connection information for the server process. (This information includes the port setting that was reported with ProcessReady().) Amazon GameLift also changes the status to ACTIVE.
7. The backend service calls DescribeGameSessionPlacement() to detect the updated ticket status. The backend service then uses the connection information to connect the game client to the server process and join the game session.

Add a player to a game

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 backend service calls the client API operation CreatePlayerSession() with a game session ID.
2. Amazon GameLift checks the game session status (must be ACTIVE), and looks for an open player slot in the game session. If a slot is available, then Amazon GameLift does the following:
    a. Creates a new PlayerSession object and sets the status to RESERVED.
    b. Responds to the backend service request with the PlayerSession object.
3. The backend service connects the game client directly to the server process with the player session ID.
4. The server calls the server API operation AcceptPlayerSession() to validate the player session ID. If validated, then Amazon GameLift passes the PlayerSession object to the server process. The server process either accepts or rejects the connection.
5. Amazon GameLift does one of the following:
    a. If the connection is accepted, then Amazon GameLift sets the PlayerSession status to ACTIVE.
    b. If no response is received within 60 seconds of the backend server's original CreatePlayerSession() call, then Amazon GameLift changes the PlayerSession status to TIMEDOUT and reopens the player slot in the game session.

Remove a player

When removing players from a game session to create space for new players to join, the following interactions occur.

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

Shut down the game session

This sequence of interactions occurs when a server process shuts down the current game session.

1. The server shuts down the game session and server.
2. The server calls `ProcessEnding()` to Amazon GameLift.
3. Amazon GameLift does the following:
    a. Uploads game session logs to Amazon Simple Storage Service (Amazon S3).
    b. Changes the `GameSession` status to `TERMINATED`.
    c. Changes the server process status to `TERMINATED`.
    d. Recycles instance resources.

Integrate your game server with Amazon GameLift

After your custom game server is deployed and running on Amazon GameLift instances, it must be able to interact with Amazon GameLift (and potentially other resources). This section describes how to integrate your game server software with Amazon GameLift.

> **Note**
>
> These instructions assume that you've created an AWS account and that you have an existing game server project.

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

- Establish communication between Amazon GameLift and your game servers.
- Generate and use a TLS certificate to establish a secure connection between game client and game server.
- Provide permissions for your game server software to interact with other AWS resources.
- Allow game server processes to get information about the fleet that they're running on.

Topics

Add Amazon GameLift to your game server

Your custom game server must communicate with Amazon GameLift, because each game server process must be able to respond to events that Amazon GameLift starts. Your game server must also keep Amazon GameLift informed about the server process status and player connections. For more information about how your game server, backend service, game client, and Amazon GameLift work together to manage game hosting, see Amazon GameLift and game client server interactions.

To prepare your game server to interact with Amazon GameLift, add the Amazon GameLift Server SDK to your game server project and build in the functionality described in this topic. The Server SDK is available in several languages. For more information about the Amazon GameLift Server SDK, see Development support with Amazon GameLift.

Server SDK API references:

- Amazon GameLift server SDK 5.x reference for C++
- Amazon GameLift server SDK 5.x reference for C# and Unity
- Amazon GameLift Unreal Engine server SDK 5.x reference

Initialize the server process

Add code to establish communication with Amazon GameLift and to report that the server process is ready to host a game session. This code must run before any Amazon GameLift code.

1. Initialize Amazon GameLift API client by calling `InitSdk()`. To initialize a server process on a Amazon GameLift Anywhere compute resource, call `InitSdk()` with the following `ServerParameters`:

   - The URL of the websocket used to connect to your game server.
   - The ID of the process used to host your game server.
   - The ID of the compute hosting your game server processes.
   - The ID of the GameLift fleet containing your Amazon GameLift Anywhere compute.
   - The authorization token generated by the Amazon GameLift operation `GetComputeAuthToken`.

   > **Note**
   >
   > To initialize a game server on a Amazon GameLift managed Amazon EC2 instance, construct your ServerParameters using the default `InitSDK()` (C++) (C#) (Unreal) constructor (without parameters). Amazon GameLift sets up the compute environment and automatically connects to Amazon GameLift for you.

2. Notify Amazon GameLift that a server process is ready to host a game session. Call `ProcessReady()` (C++) (C#) (Unreal) with the following information. (Note that you should call `ProcessReady()` only once per server process).

   - The port number that the server process uses. The backend service provides the port number and an IP address to game clients to connect to the server process and join a game session.
   - The location of files, such as game session logs, that you want Amazon GameLift to retain. The server process generates these files during a game session. They're temporarily stored on the instance where the server process is running, and they're lost when the instance shuts down. Any files that you list are uploaded to Amazon GameLift. You can access these files through the Amazon GameLift console or by calling the Amazon GameLift API operation GetGameSessionLogUrl().
   - The names of callback functions that Amazon GameLift can call to your server process. Your game server must implement these functions. For more information, see (C++) (C#) (Unreal) .

- (Optional) `onHealthCheck` – Amazon GameLift calls this function regularly to request a health status report from the server.
- `onStartGameSession` – Amazon GameLift calls this function in response to the client request CreateGameSession().
- `onProcessTerminate` – Amazon GameLift forces the server process to stop, letting it shut down gracefully.
- (Optional) `onUpdateGameSession` – Amazon GameLift delivers an updated game session object to the game server or provides a status update on a match backfill request. The FlexMatch backfill feature requires this callback.

You can also set up a game server to securely access AWS resources that you own or control. For more information, see Communicate with other AWS resources from your fleets.

(Optional) Report server process health

Add code to your game server to implement the callback function `onHealthCheck()`. Amazon GameLift invokes this callback method periodically to collect health metrics. To implement this callback function, do the following:

- Evaluate the health status of the server process. For example, you might report the server process as unhealthy if any external dependencies have failed.
- Complete the health evaluation and respond to the callback within 60 seconds. If Amazon GameLift doesn't receive a response in that time, it automatically considers the server process to be unhealthy.
- Return a Boolean value: true for healthy, false for unhealthy.

If you don't implement a health check callback, then Amazon GameLift considers the server process to be healthy unless the server doesn't respond.

Amazon GameLift uses server process health to end unhealthy processes and clear up resources. If a server process continues to report as unhealthy or doesn't respond for three consecutive health checks, then Amazon GameLift might shut down the process and start a new one. Amazon GameLift collects metrics on a fleet's server process health.

(Optional) Get a TLS certificate

If the server process is running on a fleet that has TLS certificate generation activated, then you can retrieve the TLS certificate to establish a secure connection with a game client and to encrypt client server communication. A copy of the certificate is stored on the instance. To get the file location, call `GetComputeCertificate()` (C++) (C#) (Unreal) .

Start a game session

Add code to implement the callback function on`StartGameSession`. Amazon GameLift invokes this callback to start a game session on the server.

The on`StartGameSession` function takes a GameSession object as an input parameter. This object includes key game session information, such as maximum players. It can also include game data and player data. The function implementation should do the following tasks:

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

(Optional) Validate a new player

If you're tracking the status of player sessions, then add code to validate a new player when they connect to a game server. Amazon GameLift tracks current players and available game session slots.

For validation, a game client requesting access to the game session must include a player session ID. Amazon GameLift automatically generates this ID when a player asks to join a game using StartGameSessionPlacement() or StartMatchmaking(). The player session then reserves an open slot in a game session.

When the game server process receives a game client connection request, it calls `AcceptPlayerSession()` (C++) (C#) (Unreal) with the player session ID. In response, Amazon GameLift verifies that the player session ID corresponds to an open slot reserved in the game session. After Amazon GameLift validates the player session ID, the server process accepts the

connection. The player can then join the game session. If Amazon GameLift doesn't validate the player session ID, then the server process denies the connection.

(Optional) Report a player session ending

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

To handle dropped connections, in your code, add a call to the server API operation `RemovePlayerSession()` (C++) (C#) (Unreal) with the corresponding player session ID.

End a game session

Add code to the server process shutdown sequence to notify Amazon GameLift when a game session is ending. To recycle and refresh hosting resources, Amazon GameLift shuts down server processes after the game session is complete.

At the start of the server process shutdown code, call the server API operation `ProcessEnding()` (C++) (C#) (Unreal) . This call notifies Amazon GameLift that the server process is shutting down. Amazon GameLift changes the game session status and server process status to `TERMINATED`. After calling `ProcessEnding()`, it's safe for the process to shut down.

Respond to a server process shutdown notification

Add code to shut down the server process in response to a notification from Amazon GameLift. Amazon GameLift sends this notification when the server process consistently reports unhealthy, or if the instance where the server process is running is being terminated. Amazon GameLift can stop an instance as part of a capacity scale-down event, or in response to Spot Instance interruption.

To handle a shutdown notification, make the following changes to your game server code:

- Implement the callback function on`ProcessTerminate()`. This function should call the code that shuts down the server process. When Amazon GameLift invokes this operation, Spot Instance interruptions provide a two-minute notice. This notice gives the server process time to disconnect players gracefully, preserve game state data, and perform other cleanup tasks.
- Call the server API operation `GetTerminationTime()` (C++) (C#) (Unreal) from your game server shutdown code. If Amazon GameLift has issued a call to stop the server process, then `GetTerminationTime()` returns the estimated termination time.

- At the start of your game server shutdown code, call the server API operation `ProcessEnding()` (C++) (C#) (Unreal) . This call notifies Amazon GameLift that the server process is shutting down, and Amazon GameLift then changes the server process status to `TERMINATED`. After calling `ProcessEnding()`, it's safe for the process to shut down.

Communicate with other AWS resources from your fleets

When you're creating a game server build for deployment on Amazon GameLift fleets, you might want the applications in your game build to communicate directly and securely with other AWS resources that you own. Because Amazon GameLift manages your game hosting fleets, you must give Amazon GameLift limited access to these resources and services.

Some example scenarios include:

- Use an Amazon CloudWatch agent to collect metrics, logs, and traces from managed EC2 fleets and Anywhere fleets
- Send instance log data to Amazon CloudWatch Logs.
- Obtain game files stored in an Amazon Simple Storage Service (Amazon S3) bucket.
- Read and write game data (such as game modes or inventory) stored in an Amazon DynamoDB database or other data storage service.
- Send signals directly to an instance using Amazon Simple Queue Service (Amazon SQS).
- Access custom resources that are deployed and running on Amazon Elastic Compute Cloud (Amazon EC2).

Amazon GameLift supports these methods for establishing access:

- Access AWS resources with an IAM role
- Access AWS resources with VPC peering

Access AWS resources with an IAM role

Use an IAM role to specify who can access your resources and set limits on that access. Trusted parties can "assume" a role and get temporary security credentials that authorize them to interact with the resources. When the parties make API requests related to the resource, they must include the credentials.

To set up access controlled by an IAM role, do the following tasks:

1. Create the IAM role
2. Modify applications to acquire credentials
3. Associate a fleet with the IAM role

Create the IAM role

In this step, you create an IAM role, with a set of permissions to control access to your AWS resources and a trust policy that gives Amazon GameLift rights to use the role's permissions.

For instructions on how to set up the IAM role , see Set up an IAM service role for Amazon GameLift. When creating the permissions policy, choose specific services, resources, and actions that your applications need to work with. As a best practice, limit the scope of the permissions as much as possible.

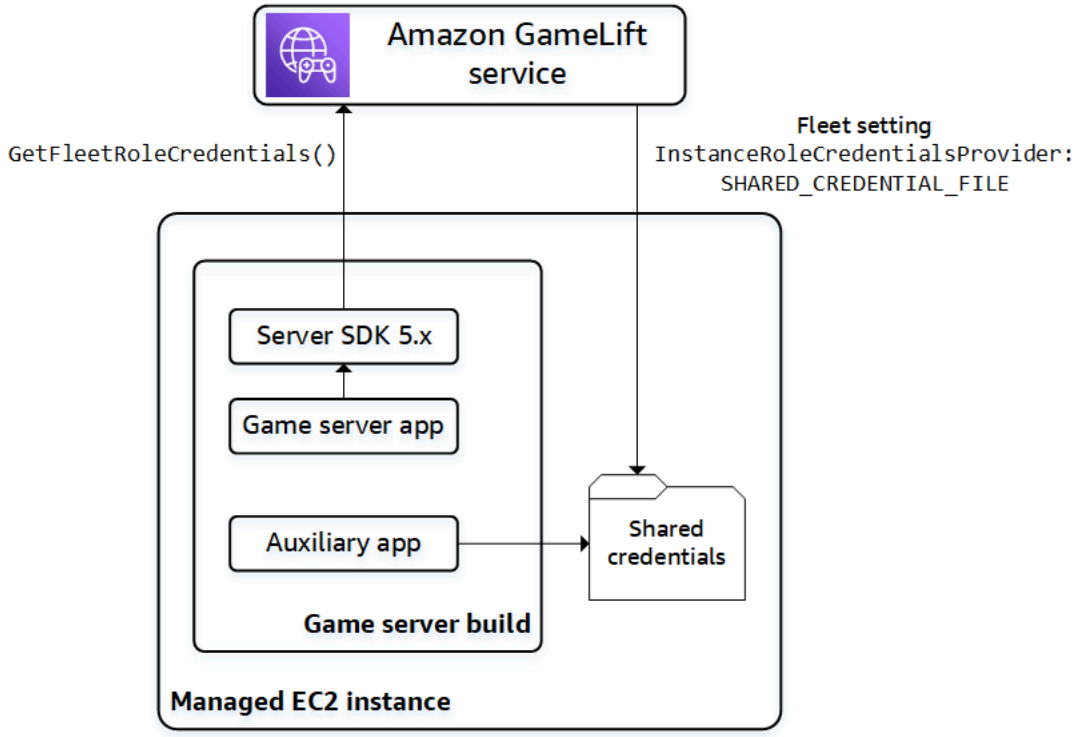
After you create the role, take note of the role's Amazon Resource Name (ARN). You need the role ARN during fleet creation.

Modify applications to acquire credentials

In this step, you configure your applications to acquire security credentials for the IAM role and use them when interacting with your AWS resources . See the following table to determine how to modify your applications based on (1) the type of application, and (2) the server SDK version your game uses to communicate with Amazon GameLift.

	Game server applications	Other applications
Using server SDK version 5.x	Call the server SDK method `GetFleetRoleCredentials()` from your game server code.	Add code to the application to pull credentials from a shared file on the fleet instance.
Using server SDK version 4 or earlier	Call AWS Security Token Service (AWS STS) `AssumeRole` with the role ARN.	Call AWS Security Token Service (AWS STS) `AssumeRole` with the role ARN.

For games integrated with server SDK 5.x, this diagram illustrates how applications in your deployed game build can acquire credentials for the IAM role.



Call `GetFleetRoleCredentials()` (server SDK 5.x)

In your game server code, which should already be integrated with the Amazon GameLift server SDK 5.x, call `GetFleetRoleCredentials` (C++) (C#) (Unreal) to retrieve a set of temporary credentials. When the credentials expire, you can refresh them with another call to `GetFleetRoleCredentials`.

Use shared credentials (server SDK 5.x)

For non-server applications that are deployed with game server builds using server SDK 5.x, add code to get and use credentials stored in a shared file. Amazon GameLift generates a credentials profile for each fleet instance. The credentials are available for use by all applications on the instance. Amazon GameLift continually refreshes the temporary credentials.

You must configure a fleet to generate the shared credentials file on fleet creation.

In each application that needs to use the shared credentials file, specify the file location and profile name, as follows:

Windows:

```
[credentials]
```

```
shared_credential_profile= "FleetRoleCredentials"
shared_credential_file= "C:\\Credentials\\credentials"
```

Linux:

```
[credentials]
shared_credential_profile= "FleetRoleCredentials"
shared_credential_file= "/local/credentials/credentials"
```

Example: Set up a CloudWatch agent to collect metrics for Amazon GameLift fleet instances

If you want to use an Amazon CloudWatch agent to collect metrics, logs, and traces from your Amazon GameLift fleets, use this method to authorize the agent to emit the data to your account. In this scenario, take the following steps:

1. Retrieve or write the CloudWatch agent `config.json` file.
2. Update the `common-config.toml` file for the agent to identify the credentials file name and profile name, as described above.
3. Set up your game server build install script to install and start the CloudWatch agent.

Use `AssumeRole()` (server SDK 4)

Add code to your applications to assume the IAM role and get credentials to interact with your AWS resources. Any application that runs on an Amazon GameLift fleet instance with server SDK 4 or earlier can assume the IAM role.

In the application code, before accessing an AWS resource, the application must call the AWS Security Token Service (AWS STS) `AssumeRole` API operation and specify the role ARN. This operation returns a set of temporary credentials that authorizes the application to access to the AWS resource. For more information, see Using temporary credentials with AWS resources in the *IAM User Guide*.

Associate a fleet with the IAM role

After you've created the IAM role and updated the applications in your game server build to get and use the access credentials, you can deploy a fleet. When you configure the new fleet, set the following parameters:

- InstanceRoleArn – Set this parameter to the ARN of the IAM role.

- InstanceRoleCredentialsProvider – To prompt Amazon GameLift to generate a shared credentials file for each fleet instance, set this parameter to `SHARED_CREDENTIAL_FILE`.

You must set these values when you create the fleet. They can't be updated later.

Access AWS resources with VPC peering

You can use Amazon Virtual Private Cloud (Amazon VPC) peering to communicate between applications running on a Amazon GameLift instance and another AWS resource. A VPC is a virtual private network that you define that includes a set of resources managed through your AWS account. Each Amazon GameLift fleet has its own VPC. With VPC peering, you can establish a direct network connection between the VPC for your fleet and for your other AWS resources.

Amazon GameLift streamlines the process of setting up VPC peering connections for your game servers. It handles peering requests, updates route tables, and configures the connections as required. For instructions about how to set up VPC peering for your game servers, see VPC peering for Amazon GameLift.

Integrate your game client with Amazon GameLift

The topics in this section describe the managed Amazon GameLift functionality that you can add to a backend service. A backend service handles the following tasks:

- Requests information about active game sessions from Amazon GameLift.
- Joins a player to an existing game session.
- Creates a new game session and joins players to it.
- Changes metadata for an existing game session.

For more information about how game clients interact with Amazon GameLift and game servers running on Amazon GameLift, see Amazon GameLift and game client server interactions.

Prerequisites

- An AWS account.
- A game server build uploaded to Amazon GameLift.
- A fleet for hosting your games.

Topics

- Add Amazon GameLift to your game client
- Generate player IDs

Add Amazon GameLift to your game client

Integrate Amazon GameLift into game components that need game session information, create new game sessions, and add players to games. Depending on your game architecture, this functionality is in backend services that handle tasks such as player authentication, matchmaking, or game session placement.

> **Note**
>
> For detailed information about how to set up matchmaking for your Amazon GameLift hosted game, see the Amazon GameLift FlexMatch Developer Guide.

Set up Amazon GameLift on a backend service

Add code to initialize an Amazon GameLift client and store key settings. This code must run before any code dependent on Amazon GameLift.

1. Set up a client configuration. Use the default client configuration or create a custom client configuration object. For more information, see AWS::Client::ClientConfiguration (C++) or AmazonGameLiftConfig (C#).

   A client configuration specifies a target region and endpoint to use when contacting Amazon GameLift. Region identifies the set of deployed resources (fleets, queues, and matchmakers) to use. The default client configuration sets location to the US East (N. Virginia) Region. To use any other Region, create a custom configuration.
2. Initialize an Amazon GameLift client. Use Aws::GameLift::GameLiftClient() (C++) or AmazonGameLiftClient() (C#) with a default client configuration or a custom client configuration.
3. Add a mechanism to generate a unique identifier for each player. For more information, see Generate player IDs.
4. Collect and store the following information:

   - **Target fleet** – Many Amazon GameLift API requests must specify a fleet. To do so, use either a fleet ID or an alias ID that points to the target fleet. As a best practice, use fleet aliases

so that you can switch players from one fleet to another without having to update your backend services.

- **Target queue** – For games that use multi-fleet queues to place new game sessions, specify the name of the queue to use.
- **AWS credentials** – All calls to Amazon GameLift must provide credentials for the AWS account that hosts the game. You acquire these credentials by creating a player user, as described in Set up programmatic access for your game. Depending on how you manage access for the player user, do the following:
  - If you use a role to manage player user permissions, add code to assume the role before calling an Amazon GameLift API. The request to assume the role returns a set of temporary security credentials. For more information, see Switching to an IAM role (AWS API) in the *IAM User Guide*.
  - If you have long-term security credentials, configure your code to locate and use stored credentials. See Authenticate using long-term credentials in in the *AWS SDKs and Tools Reference Guide*. For information on storing credentials, see theAWS API references for (C++) and (.NET).
  - If you have temporary security credentials, add code to regularly refresh the credentials using the AWS Security Token Service (AWS STS), as described in Using temporary security credentials with the AWS SDKs in the *IAM User Guide*. The code must request new credentials before the old ones expire.

Get game sessions

Add code to discover available game sessions and manage game session settings and metadata.

Search for active game sessions

Use SearchGameSessions to get information about a specific game session, all active sessions, or sessions that meet a set of search criteria. This call returns a GameSession object for each active game session that matches your search request.

Use search criteria to get a filtered list of active game sessions for players to join. For example, you can filter sessions as follows:

- Exclude game sessions that are full: `CurrentPlayerSessionCount = MaximumPlayerSessionCount`.

- Choose game sessions based on length of time that the session has been running: Evaluate `CreationTime`.
- Find game sessions based on a custom game property: `gameSessionProperties.gameMode = "brawl"`.

Manage game 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're deploying your game in multiple AWS Regions or in a single Region.

Create a game session in a multi-location queue

Use StartGameSessionPlacement to place a request for a new game session in a queue. To use this operation, create a queue. This determines where Amazon GameLift places the new game session. For more information about queues and how to use them, see Setting up Amazon GameLift queues for game session placement.

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

Game session placement is an asynchronous process. After you've placed a request, you can let it succeed or time out. You can also cancel the request at any time using StopGameSessionPlacement. To check the status of your placement request, call DescribeGameSessionPlacement.

Create a game session on a specific fleet

Use CreateGameSession to create a new session on a specified fleet. This synchronous operation succeeds or fails depending on whether the fleet has resources available to host a new game session. After Amazon GameLift creates the new game session and returns a GameSession object, you can join players to it.

When you use this operation, provide a fleet ID or alias ID, a session name, and the maximum number of concurrent players for the game. Optionally, you can include a set of game properties. Game properties are defined in an array of key-value pairs.

If you use the Amazon GameLift resource protection feature to limit the number of game sessions that one player can create, then provide the game session creator's player ID.

Join a player to a game session

Add code to reserve a player slot in an active game session and connect game clients to game sessions.

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

   To reserve a player slot, create a new player session for the game session. For more information about player sessions, see How players connect to games.

   There are two ways to create new player sessions:

   - Use StartGameSessionPlacement to 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.

   Amazon GameLift first verifies that the game session is accepting new players and has available player slots. If successful, Amazon GameLift reserves a slot for the player, creates the new player session, and returns a PlayerSession object. This object contains 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. For more information, see Generate player IDs.

   A player session can include a set of custom player data. This data is stored in the newly created player session object, which you can retrieve by calling DescribePlayerSessions(). Amazon GameLift also passes this object to the game server when the player connects directly

to the game session. When requesting multiple player sessions, provide a string of player data for each player that's 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.

   - You can connect using the specified port and the DNS name or IP address assigned to the server process.
   - If your fleets have TLS certificate generation enabled, then connect using the DNS name and port.
   - If your game server validates incoming player connections, then reference the player session ID.

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

Use game session properties

Your game client can pass data into a game session by using a game property. Game properties are key-value pairs that your game server can add, read, list, and change. You can pass in a game property when you're creating a new game session, or later when the game session is active. A game session can contain up to 16 game properties. You cannot delete game properties.

For example, your game offers these difficulty levels: `Novice`, `Easy`, `Intermediate`, and `Expert`. A player chooses `Easy`, and then begins the game. Your game client requests new game session from Amazon GameLift by using either `StartGameSessionPlacement` or `CreateGameSession` as explained in the preceding sections. In the request, the client passes this : `{"Key": "Difficulty", "Value":"Easy"}`.

In response to the request, Amazon GameLift creates a `GameSession` object that contains the specified game property. Amazon GameLift then instructs an available game server to start the

new game session and passes the `GameSession` object. The game server starts a game session with a `Difficulty` of `Easy`.

Learn more

- GameProperty data type
- SearchGameSessions() examples
- UpdateGameSession() GameProperties parameter

Generate player IDs

Amazon GameLift uses a player session to represent a player connected to a game session. Amazon GameLift creates a player session each time a player connects to a game session using a game client integrated with Amazon GameLift. When a player leaves a game, the player session ends. Amazon GameLift doesn't reuse player sessions.

The following code example randomly generates unique player IDs:

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

For more information about player sessions, see View data on game and player sessions.

Game engines and Amazon GameLift

You can use the managed Amazon GameLift service with most major game engines that support C++ or C# libraries, including O3DE, Unreal Engine, and Unity. Build the version you need for your game; see the README files with each version for build instructions and minimum requirements. For more information on available Amazon GameLift SDKs, supported development platforms and operating systems, see Development support with Amazon GameLift for game servers.

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

- Amazon GameLift managed hosting roadmap – A six-step workflow for successfully integrating Amazon GameLift into your game and setting up hosting resources.

- Add Amazon GameLift to your game server – Detailed instructions on integrating Amazon GameLift into a game server.
- Add Amazon GameLift to your game client – Detailed instructions on integrating into a game client or service, including creating game sessions and joining players to games.

O3DE

Game servers

Prepare your game servers for hosting on Amazon GameLift using the Amazon GameLift Server SDK for C++. See Add Amazon GameLift to your game server to get help with integrating the required functionality into your game server.

Game clients and services

Enable your game clients and/or game services to interact with Amazon GameLift service, such as to find available game sessions or create new ones, and add players to games. Core client functionality is provided in the AWS SDK for C++. To integrate Amazon GameLift into your O3DE game project, see Add Amazon GameLift to an O3DE game client and server and Add Amazon GameLift to your game client.

Unreal Engine

Game servers

Prepare your game servers for hosting on Amazon GameLift by adding the Amazon GameLift Server SDK for Unreal Engine to your project and implementing the required server functionality. For help setting up the Unreal Engine plugin and adding Amazon GameLift code, see Integrate Amazon GameLift into an Unreal Engine project.

Game clients and services

Enable your game clients and/or game services to interact with Amazon GameLift service, such as to find available game sessions or create new ones, and add players to games. Core client functionality is provided in the AWS SDK for C++. To integrate Amazon GameLift into your Unreal Engine game project, see Add Amazon GameLift to your game client.

Unity

Game servers

Prepare your game servers for hosting on Amazon GameLift by adding the Amazon GameLift Server SDK for C# to your project and implementing the required server functionality. For help setting up with Unity and adding Amazon GameLift code, see Integrate Amazon GameLift into a Unity project.

Game clients and services

Enable your game clients and/or game services to interact with Amazon GameLift service, such as to find available game sessions or create new ones, and add players to games. Core client functionality is provided in the AWS SDK for .NET. To integrate Amazon GameLift into your Unity game project, see Add Amazon GameLift to your game client.

Other engines

For a full list of the Amazon GameLift SDKs available for game servers and clients, see the section called “Development support”.

Add Amazon GameLift to an O3DE game client and server

You can use O3DE, an open-source, cross-platform, real time 3D engine to create high performance interactive experiences, including games and simulations. The O3DE renderer and tools are wrapped in a modular framework that you can modify and extend with your preferred development tools.

The modular framework uses *Gems* that contain libraries with standard interfaces and assets. Select your own Gems to choose what functionality to add based on your requirements.

The Amazon GameLift Gem provides the following features:

Amazon GameLift integration

A framework to extend the O3DE networking layer and to let the Multiplayer Gem work with the Amazon GameLift dedicated server solution. The Gem provides integrations with both the Amazon GameLift server SDK and the AWS SDK client (to call the Amazon GameLift service itself).

Build and package management

Instructions to package and optionally upload the dedicated server build and an AWS Cloud Development Kit (AWS CDK) (AWS CDK) application to set up and update resources.

Amazon GameLift Gem setup

Follow the procedures in this section to set up the Amazon GameLift Gem in O3DE.

Prerequisites

- Set up your AWS account for Amazon GameLift. For more information, see Set up an AWS account.
- Set up AWS credentials for O3DE. For more information see, Configuring AWS Credentials.
- Set up the AWS CLI and AWS CDK. For more information, AWS Command Line Interface and AWS Cloud Development Kit (AWS CDK).

Turn on the Amazon GameLift Gem and its dependencies

1. Open the **Project Manager**.
2. Open the menu under your project and choose **Edit Project Setting...**.
3. Choose **Configure Gems**.
4. Turn on the Amazon GameLift Gem and the following dependent Gems:
   - AWS Core Gem – Provide the framework to use AWS services in O3DE.
   - Multiplayer Gem – Provides multiplayer functionality by extending the networking framework.

Include the Amazon GameLift Gem static library

1. Include the `Gem::AWSGameLift.Server.Static` as `BUILD_DEPENDENCIES` for your project server target.

```
ly_add_target(
    NAME YourProject.Server.Static STATIC
    ...
    BUILD DEPENDCIES
        PUBLIC
            ...
        PRIVATE
            ...
            Gem::AWSGameLift.Server.Static
)
```

2. Set `AWSGameLiftService` to required for your project server system component.

```
void
 YourProjectServerSystemComponent::GetRequiredServices(AZ::ComponentDescriptor::DependencyA
 required)
{
    ...
    required.push_back(AZ_CRC_CE("AWSGameLiftServerService"));
    ...
}
```

3. (Optional) To make Amazon GameLift service requests in C++, include `Gem::AWSGameLift.Client.Static` in the `BUILD_DEPENDENCIES` for your client target.

```
ly_add_target(
    NAME YourProject.Client.Static STATIC
    ...
    BUILD_DEPENDENCIES
    PUBLIC
        ...
    PRIVATE
        ...
        Gem::AWSCore.Static
        Gem::AWSGameLift.Client.Static
}
```

Integrate your game and dedicated server

Manage game sessions within your game and dedicated game server with the Session Management Integration. To support FlexMatch, see FlexMatch Integration.

Integrate Amazon GameLift into an Unreal Engine project

This topic explains how to set up the **Amazon GameLift C++ server SDK plugin for Unreal Engine** and integrate it into your game projects.

Additional resources:

- Server SDK plugin for Unreal download site
- Amazon GameLift Unreal Engine server SDK 5.x reference
- the section called “Development support”

Prerequisites

Before you procced, make sure you have reviewed the following prerequisites:

Prerequisites

- A computer capable of running Unreal Engine. For more information on Unreal Engine requirements, see Unreal Engine's Hardware and Software Specifications documentation.
- Microsoft Visual Studio 2019 or newer version.
- CMake version 3.1 or later.
- Python version 3.6 or later.
- A Git client available on the PATH.
- An Epic games account. Sign up for an account at the official Unreal Engine website.
- A GitHub account associated with your Unreal Engine account. For more information, see Accessing Unreal Engine source code on GitHub on the Unreal Engine website.

> **Note**
>
> Amazon GameLift currently supports the following versions of Unreal Engine:
>
> - 4.22
> - 4.23
> - 4.24
> - 4.25
> - 4.26
> - 4.27
> - 5.1.0
> - 5.1.1

Build Unreal Engine from source

Standard versions of the Unreal Engine editor, downloaded through the Epic launcher, only allow Unreal client application builds. In order to build an Unreal server application, you need to download and build Unreal Engine from source, using the Unreal Engine Github repo. For

more information, see the Building Unreal Engine from Source tutorial on the Unreal Engine documentation website.

> **Note**
>
> If you haven't already done so, follow the instructions at Accessing Unreal Engine source code on GitHub to link your GitHub account to your Epic Games account.

To clone the Unreal Engine source to your development environment

1. Clone the Unreal Engine source to your development environment in a branch of your choice.

   ```
   git clone https://github.com/EpicGames/UnrealEngine.git
   ```

2. Check out the tag of the version that you're using to develop your game. For example, the following example checks out Unreal Engine version 5.1.1:

   ```
   git checkout tags/5.1.1-release -b 5.1.1-release
   ```

3. Navigate to the root folder of the local repository. When you're in the root folder, run the following file: `Setup.bat`.
4. While in the root folder, also run the file: `GenerateProjectFiles.bat`.
5. After running the files from the previous steps, an Unreal Engine solution file, `UE5.sln`, is created. Open **Visual Studio**, and in the Visual Studio editor open the `UE5.sln` file.
6. In Visual Studio, open the **View** menu and choose the **Solution Explorer** option. This opens the context menu of the Unreal project node. In the **Solution Explorer** window, right-click the `UE5.sln` file (it can be listed as just UE5), then choose **Build** to build the Unreal project with the Development Editor Win64 target.

   > **Note**
   >
   > The build can take over an hour to complete.

Once the build is complete, you are ready to open the Unreal Development Editor and create or import a project.

Configure your Unreal project for the plugin

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

To configure your project for the plugin

1. With Visual Studio open, navigate to the **Solution Explorer** pane and choose the UE5 file to open the context menu for the Unreal project. In the context menu, choose the **Set as Startup Project** option.
2. At the top of your Visual Studio window, choose **Start Debugging** (green arrow).

   This action starts your new source-built instance of Unreal Editor. For more information about using the Unreal Editor, see Unreal Editor Interface on the Unreal Engine documentation website.
3. Close the Visual Studio window you opened, since the Unreal Editor opens a another Visual Studio window that contains the Unreal project and your game project.
4. In the Unreal editor, do one of the following:
   - Choose an existing Unreal project that you want to integrate with Amazon GameLift.
   - Create a new project. To experiment with the Amazon GameLift plugin for Unreal, try using Unreal engine's **Third Person** template. For more information about this template, see Third Person template on the Unreal Engine documentation website.

     Alternatively, configure a new project with the following settings:

     - **C++**
     - **With starter content**
     - **Desktop**
     - **A project name**. In the examples in this topic, we named our project `GameLiftUnrealApp`.
5. In Visual Studio's **Solution Explorer**, navigate to the location of your Unreal project. In the Unreal `Source` folder, find a file named *`Your-application-name`*`.Target.cs`.

   For example: `GameLiftUnrealApp.Target.cs`.
6. Make a copy of this file and name the copy: *`Your-application-name`*`Server.Target.cs`.
7. Open the new file and make the following changes:

- Change the `class` and `constructor` to match the filename.
- Change the `Type` from `TargetType.Game` to `TargetType.Server`.
- The final file will look like the following example:

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

Your project is now configured to accept the Amazon GameLift server SDK plugin.

The next task is to build the C++ server SDK libraries for Unreal so that you can import them into your project.

To build the C++ server SDK libraries for Unreal

1. Download the Amazon GameLift C++ server SDK plugin for Unreal.

   > **Note**
   >
   > Putting the SDK in the default download directory can result in build failure due to the path exceeding the 260 character limit. For example: `C:\Users\Administrator\Downloads\GameLift-SDK-Release-06_15_2023\GameLift-Cpp-ServerSDK-5.0.4`
   > We recommend that you move the SDK to another directory, for example `C:\GameLift-Cpp-ServerSDK-5.0.4`.

2. Download and install OpenSSL. For more information on downloading OpenSSL, read the Github OpenSSL build and install documentation.

   For more information, read the OpenSSL Notes for Windows platforms documentation.

> **Note**
>
> The version of OpenSSL that you use to build the Amazon GameLift server SDK should match the version of OpenSSL used by Unreal to package your game server. You can find version information in the Unreal installation directory `...Engine\Source\ThirdParty\OpenSSL`.

3. With the libraries downloaded, build the C++ server SDK libraries for Unreal Engine.

   In the `GameLift-Cpp-ServerSDK-<version>` directory in the downloaded SDK, compile with the `-DBUILD_FOR_UNREAL=1` parameter and build the server SDK. The following examples show how to compile using `cmake`.

   Run the following commands in your terminal:

   ```
   mkdir cmake-build
   cmake.exe -G "Visual Studio 17 2022" -DCMAKE_BUILD_TYPE=Release -S . -B ./cmake-build -DBUILD_FOR_UNREAL=1 -A x64
   cmake.exe --build ./cmake-build --target ALL_BUILD --config Release
   ```

   The Windows build creates the following binary files in the `out\gamelift-server-sdk\Release` folder:

   - `cmake-build\prefix\bin\aws-cpp-sdk-gamelift-server.dll`
   - `cmake-build\prefix\bin\aws-cpp-sdk-gamelift-server.lib`

   Copy the two library files to the `ThirdParty\GameLiftServerSDK\Win64` folder in the Amazon GameLift Unreal Engine plugin package.

Use the following procedure to import the Amazon GameLift plugin into your example project.

Import the Amazon GameLift plugin

1. Locate the `GameLiftServerSDK` folder that you extracted from the plugin in the earlier procedure.
2. Locate the `Plugins` in your game project root folder. (If the folder does not exist, then create it there.)

3. Copy the `GameLiftServerSDK` folder into the `Plugins`.

   This will allow the Unreal project to see the plugin.

4. Add the Amazon GameLift server SDK plugin to the game's `.uproject` file.

   In the example, the app is called `GameLiftUnrealApp`, so the file will be `GameLiftUnrealApp.uproject`.

5. Edit the `.uproject` file to add the plugin to your game project.

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

6. Make sure the game's ModuleRules takes a dependency on the plugin. Open the `.Build.cs` file and add the Amazon GameLiftServerSDK dependency. This file is found under *`Your-application-name`*`/Source/`*`/Your-application-name/`*.

   For example, the tutorial filepath is `../GameLiftUnrealApp/Source/GameLiftUnrealApp/GameLiftUnrealApp.Build.cs`.

7. Add `"GameLiftServerSDK"` to the end of the list of `PublicDependencyModuleNames`.

```
using UnrealBuildTool;
using System.Collections.Generic;
public class GameLiftUnrealApp : ModuleRules
  {
    public GameLiftUnrealApp(TargetInfo Target)
    {
      PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject",
 "Engine", "InputCore", "GameLiftServerSDK" });
      bEnableExceptions = true;
    }
  }
```

The plugin should now be working for your application. Continue with the next section to integrate Amazon GameLift functionality into your game.

Add Amazon GameLift server code to your Unreal project

You've configured and set up your Unreal Engine environment, and you can now integrate a game server with Amazon GameLift. The code presented in this topic makes required calls to the Amazon GameLift service. It also implements a set of callback functions that respond to requests from the Amazon GameLift service. For more information on each function and what the code does, see Initialize the server process. For more information on the SDK actions and datatypes uised in this code, read Amazon GameLift server SDK reference for Unreal Engine.

To initialize a game server with Amazon GameLift, use the following procedure.

> **Note**
>
> The Amazon GameLift-specific code provided in the following section depends on the use of a `WITH_GAMELIFT` preprocessor flag. This flag is true only when both of these conditions are met:
>
> - `Target.Type == TargetRules.TargetType.Server`
> - The plugins found the Amazon GameLift server SDK binaries.
>
> This ensures that only Unreal Server builds invoke Amazon GameLift's backend API. It also lets you to write code that will execute properly for all the different Unreal targets your game might produce.

Integrate a game server with Amazon GameLift

1. In Visual Studio, open the `.sln` file for your application. For our example, the file `GameLiftUnrealApp.sln` is found in the root folder.
2. With the solution open, locate your application's *`Your-application-name`*`GameMode.h` file. Example: `GameLiftUnrealAppGameMode.h`.
3. Change the header file to align with the following example code. Be sure to replace "GameLiftUnrealApp" with your own application name.

```
#pragma once

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

```
#include "GameLiftServerSDK.h"
#include "GameLiftUnrealAppGameMode.generated.h"

DECLARE_LOG_CATEGORY_EXTERN(GameServerLog, Log, All);

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

public:
    AGameLiftUnrealAppGameMode();

protected:
    virtual void BeginPlay() override;

private:
    // Process Parameters needs to remain in scope for the lifetime of the app
    FProcessParameters m_params;

    void InitGameLift();
};
```

4. Open the related source file *Your-application-name*GameMode.cpp file. In our Example: GameLiftUnrealAppGameMode.cpp. and change the code to align with the following example code. Be sure to replace "GameLiftUnrealApp" with your own application name.

   This sample shows how to add all of the required elements for integration with Amazon GameLift, as described in Add Amazon GameLift to your game server. This includes:

   - Initializing an Amazon GameLift API client.
   - Implementing callback functions to respond to requests from the Amazon GameLift service, including OnStartGameSession, OnProcessTerminate, and onHealthCheck.
   - Calling ProcessReady() with a designated port to notify the Amazon GameLiftservice when ready to host game sessions.

```
#include "GameLiftUnrealAppGameMode.h"
#include "GameLiftUnrealAppCharacter.h"

#include "UObject/ConstructorHelpers.h"
```

```
DEFINE_LOG_CATEGORY(GameServerLog);

AGameLiftUnrealAppGameMode::AGameLiftUnrealAppGameMode()
{
    // set default pawn class to our Blueprinted character
    static ConstructorHelpers::FClassFinder<APawn> PlayerPawnBPClass(TEXT("/Game/
ThirdPerson/Blueprints/BP_ThirdPersonCharacter"));
    if (PlayerPawnBPClass.Class != NULL)
    {
        DefaultPawnClass = PlayerPawnBPClass.Class;
    }
}

void AGameLiftUnrealAppGameMode::BeginPlay()
{
#if WITH_GAMELIFT
    InitGameLift();
#endif
}

void AGameLiftUnrealAppGameMode::InitGameLift()
{
    UE_LOG(GameServerLog, Log, TEXT("Initializing the GameLift Server"));

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

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

    //AuthToken returned from the "aws gamelift get-compute-auth-token" API. Note
 this will expire and require a new call to the API after 15 minutes.
    if (FParse::Value(FCommandLine::Get(), TEXT("-authtoken="),
 serverParameters.m_authToken))
    {
        UE_LOG(GameServerLog, Log, TEXT("AUTH_TOKEN: %s"),
 *serverParameters.m_authToken)
    }

    //The Host/compute-name of the GameLift Anywhere instance.
    if (FParse::Value(FCommandLine::Get(), TEXT("-hostid="),
 serverParameters.m_hostId))
```

```
    {
        UE_LOG(GameServerLog, Log, TEXT("HOST_ID: %s"), *serverParameters.m_hostId)
    }

    //The Anywhere Fleet ID.
    if (FParse::Value(FCommandLine::Get(), TEXT("-fleetid="),
serverParameters.m_fleetId))
    {
        UE_LOG(GameServerLog, Log, TEXT("FLEET_ID: %s"),
*serverParameters.m_fleetId)
    }

    //The WebSocket URL (GameLiftServiceSdkEndpoint).
    if (FParse::Value(FCommandLine::Get(), TEXT("-websocketurl="),
serverParameters.m_webSocketUrl))
    {
        UE_LOG(GameServerLog, Log, TEXT("WEBSOCKET_URL: %s"),
*serverParameters.m_webSocketUrl)
    }

    //The PID of the running process
    serverParameters.m_processId = FString::Printf(TEXT("%d"),
GetCurrentProcessId());
    UE_LOG(GameServerLog, Log, TEXT("PID: %s"), *serverParameters.m_processId);

    //InitSDK establishes a local connection with GameLift's agent to enable
further communication.
    //Use InitSDK(serverParameters) for a GameLift Anywhere fleet.
    //Use InitSDK() for a GameLift managed EC2 fleet.
    gameLiftSdkModule->InitSDK(serverParameters);

    //Implement callback function onStartGameSession
    //GameLift sends a game session activation request to the game server
    //and passes a game session object with game properties and other settings.
    //Here is where a game server takes action based on the game session object.
    //When the game server is ready to receive incoming player connections,
    //it invokes the server SDK call ActivateGameSession().
    auto onGameSession = [=](Aws::GameLift::Server::Model::GameSession gameSession)
    {
        FString gameSessionId = FString(gameSession.GetGameSessionId());
        UE_LOG(GameServerLog, Log, TEXT("GameSession Initializing: %s"),
*gameSessionId);
        gameLiftSdkModule->ActivateGameSession();
    };
```

```
   m_params.OnStartGameSession.BindLambda(onGameSession);

   //Implement callback function OnProcessTerminate
   //GameLift invokes this callback before shutting down the instance hosting this
game server.
   //It gives the game server a chance to save its state, communicate with
services, etc.,
   //and initiate shut down. When the game server is ready to shut down, it
invokes the
   //server SDK call ProcessEnding() to tell GameLift it is shutting down.
   auto onProcessTerminate = [=]()
   {
       UE_LOG(GameServerLog, Log, TEXT("Game Server Process is terminating"));
       gameLiftSdkModule->ProcessEnding();
   };

   m_params.OnTerminate.BindLambda(onProcessTerminate);

   //Implement callback function OnHealthCheck
   //GameLift invokes this callback approximately every 60 seconds.
   //A game server might want to check the health of dependencies, etc.
   //Then it returns health status true if healthy, false otherwise.
   //The game server must respond within 60 seconds, or GameLift records 'false'.
   //In this example, the game server always reports healthy.
   auto onHealthCheck = []()
   {
       UE_LOG(GameServerLog, Log, TEXT("Performing Health Check"));
       return true;
   };

   m_params.OnHealthCheck.BindLambda(onHealthCheck);

   //The game server gets ready to report that it is ready to host game sessions
   //and that it will listen on port 7777 for incoming player connections.
   m_params.port = 7777;

   //Here, the game server tells GameLift where to find game session log files.
   //At the end of a game session, GameLift uploads everything in the specified
   //location and stores it in the cloud for access later.
   TArray<FString> logfiles;
   logfiles.Add(TEXT("GameLift426Test/Saved/Logs/GameLift426Test.log"));
   m_params.logParameters = logfiles;
```

```
    //The game server calls ProcessReady() to tell GameLift it's ready to host game
 sessions.
    UE_LOG(GameServerLog, Log, TEXT("Calling Process Ready"));
    gameLiftSdkModule->ProcessReady(m_params);
}
```

5. Build game project for both of the following target types: *Development Editor* and *Development Server*.

> **Note**
>
> You don't need to rebuild the solution. Instead, build just the project under the `Games` folder that matches the name of your app. Otherwise Visual Studio rebuilds the entire UE5 project, which might take up to an hour.

6. Once both builds are complete, close Visual Studio and open your project's `.uproject` file to open it in the Unreal Editor.
7. In Unreal Editor, package the server build of your game. To choose a target, go to **Platforms**, **Windows** and select ***`Your-application-nameServer`***.
8. To start the process of building the server application, go to **Platforms**, **Windows** and select **Package Project**. When the build is complete, you should have an executable. In the case of our example, the file name is `GameLiftUnrealAppServer.exe`.
9. Building a server application in Unreal Editor produces two executables. One is located in the root of the game build folder and acts as a wrapper for the actual server executable.

   When creating an Amazon GameLift fleet with your server build, we recommend that you pass in the actual server executable as the runtime configuration launch path. For example, in your game build folder, you might have a `GameLiftFPS.exe` file at the root and another at `\GameLiftFPS\Binaries\Win64\GameLiftFPSServer.exe`. When creating a fleet, we recommend you use `C:\GameLiftFPS\Binaries\Win64\GameLiftFPSServer.exe` as the launch path of the runtime configuration.
10. Make sure to open the necessary UDP ports on the Amazon GameLift fleet, so that the game server can communicate with game clients. By default, Unreal Engine uses port 7777. For more information, see UpdateFleetPortSettings in the Amazon GameLift service API reference guide.
11. Create an `install.bat` file for your game build. This install script runs whenever the game build is deployed to a Amazon GameLift fleet. Here's an example `install.bat` file:

```
VC_redist.x64.exe /q
UE5PrereqSetup_x64.exe /q
```

For some versions of Unreal Engine, the `install.bat` should instead be

```
VC_redist.x64.exe /q
UEPrereqSetup_x64.exe /q
```

> **Note**
>
> The file path to the `<>PrereqSetup_x64.exe` file is `Engine\Extras\Redist\en-us`.

12. Now you can package and upload your game build to Amazon GameLift.

    The version of OpenSSL you package with your game build needs to match the version that the game engine used when building the game server. Make sure you package the correct OpenSSL version with your game server build. For the Windows OS, the OpenSSL format is `.dll`.

    > **Note**
    >
    > Package the OpenSSL DLLs in your game server build. Be sure to package the same version of OpenSSL that you used when building the game server.
    >
    > - `libssl-1_1-x64.dll`
    >
    >   `libcrypto-1_1-x64.dll`
    >
    > Package your dependencies along with your game server executable in the root of a zip file. For example, `openssl-lib` dlls should be in the same directory as the `.exe` file.

Next steps

You've configured and set up your Unreal Engine environment, and you can now start integrating Amazon GameLift into your game.

For more information about adding Amazon GameLift to your game, see the following:

- Add Amazon GameLift to your game server
- Amazon GameLift server SDK reference for Unreal Engine

For instructions about testing your game, see Test your integration using Amazon GameLift Anywhere fleets .

Integrate Amazon GameLift into a Unity project

This topic explains how to set up the Amazon GameLift **C# Server SDK plugin for Unity** and integrate it into your game projects.

Additional resources:

- Amazon GameLift server SDK download site
- Amazon GameLift server SDK 5.x reference for C# and Unity
- the section called “Development support”

Prerequisites

To use the Amazon GameLift C# server SDK plugin for Unity, you need the following components:

- A development environment and Unity Editor version that the plugin supports (see Development support with Amazon GameLift). For information on Unity versions, see System requirements for Unity in the Unity documentation.
- The Amazon GameLift server SDK plugin for Unity package. This package includes the server SDK 5+ for C#. You can download the package from this site: Getting Started with Amazon GameLift.
- The third party scoped registry UnityNuGet. This tool manages third-party DLLs. For more information, see the UnityNuGet Github repository.

Set up UnityNuGet

If you don't have UnityNuGet set up for your game project, use the following steps to install the tool using the Unity package manager. Alternatively, you can use the NuGet CLI to manually download the DLLs. For more information, see the Amazon GameLift C# server SDK for Unity `README`.

To integrate UnityNuGet into your game project

1. With your project open in the Unity Editor, go to the main menu and select **Edit**, **Project Settings**. From the options, choose the **Package Manager** section and open the **Scoped Registries** group.
2. Choose the **+** button and enter the following values for the UnityNuGet scoped registry:

   ```
   Name: Unity NuGet
   URL: https://unitynuget-registry.azurewebsites.net
   Scope(s): org.nuget
   ```

3. For Unity 2021 version users:

   After setting up UnityNuGet, check for `Assembly Version Validation` errors showing in the Unity console. These errors occur if binding redirects for strongly named assemblies in the NuGet packages are not resolving correctly to paths within your Unity project. To resolve this issue, configure Unity's assembly version validation:

   a. In the Unity Editor, go to the main menu and select **Edit**, **Project Settings**, and open the **Player** section.

   b. Deselect the **Assembly Version Validation** option.

Install the plugin

Use the following procedure to install the Amazon GameLift C# server SDK plugin for Unity and configure log4net logging.

To install the plugin

1. With your project open in the Unity Editor, go to the main menu and select **Window**, **Package Manager**.
2. Choose the **+** button to add a new package. Choose the option **Add package from tarball**.
3. In **Select packages on disk**, locate the Amazon GameLift C# Server SDK plugin for Unity download files, and choose the Amazon GameLift Server SDK `.tgz` file. Choose **Open** to install the plugin.

The Amazon GameLift server SDK uses the log4net framework to output log messages. It is configured to output messages to the terminal of a server build by default, but Unity requires

configuration to add file logging support. You can add this support to your project by importing the provided sample inside the Amazon GameLift Server SDK package. Use the following procedure to add the sample and configure log4net:

To configure log4net for file output

1. With your project open in the Unity Editor, go to the main menu and select **Window**, **Package Manager**.
2. From the dropdown menu, select **Packages: In Project**, and then select **Amazon GameLift Server SDK** from the list of packages. This opens the package details.
3. In the package details, select the Samples group option and press **Import**.
4. The `log4net.config` file and accompanying `LoggingConfiguration.cs` script automatically executes the configuration, which is now set up in the project's `Assets/Samples` folder.

   > **Note**
   >
   > If you need to move your `log4net.config` file to a different folder in the project, then you must also update the config file's filepath in the script `LoggingConfiguration.cs` with the new path. For more information, see the log4net manual on configuring log4net.

For more detailed instructions and testing guidance, see the `README` located in the plugin download.

Set up an Amazon GameLift Anywhere fleet for testing

You can set up your development workstation as an Amazon GameLift Anywhere hosting fleet to iteratively test your Amazon GameLift integration. With this setup, you can start game server processes on your workstation, send player join or matchmaking requests to Amazon GameLift to start game sessions, and connect clients to the new game sessions. With your own workstation set up as a hosting server, you can monitor all aspects of your game integration with Amazon GameLift.

For instructions on setting up your workstation, see Test your integration using Amazon GameLift Anywhere fleets to complete the following steps:

1. Create a custom location for your workstation.

2. Create an Amazon GameLift Anywhere fleet with your new custom location. If successful, this request returns a fleet ID. Make a note of this value, as you'll need it later.
3. Register your workstation as a compute in the new Anywhere fleet. Provide a unique compute name and specify the IP address for your workstation. If successful, this request returns a service SDK endpoint, in the form of a WebSocket URL. Make a note of this value, as you'll need it later.
4. Generate an authentication token for your workstation compute. This short-lived authentication includes the token and an expiration date. Your game server uses it to authenticate communication with the Amazon GameLift service. Store the authentication on your workstation compute so that your running game server processes can access it.

Add Amazon GameLift server code to your Unity project

Your game server communicates with the Amazon GameLift service to receive instructions and report ongoing status. To accomplish this, you add game server code that uses the Amazon GameLift server SDK.

The provided code example illustrates the basic required integration elements. It uses a `MonoBehavior` to illustrate a simple game server initialization with Amazon GameLift. The example assumes that the game server runs on an Amazon GameLift Anywhere fleet for testing. It includes code to:

- Initialize an Amazon GameLift API client. The sample uses the version of `InitSDK()` with server parameters for your Anywhere fleet and compute. Use the WebSocket URL, fleet ID, compute name (host ID), and authentication token, as defined in the previous topic Set up an Amazon GameLift Anywhere fleet for testing.
- Implement callback functions to respond to requests from the Amazon GameLift service, including `OnStartGameSession`, `OnProcessTerminate`, and `onHealthCheck`.
- Call ProcessReady() with a designated port to notify the Amazon GameLift service when the process is ready to host game sessions.

The code presented in this topic establishes communication with the Amazon GameLift service and . It also implements a set of callback functions that respond to requests from the . For more information on each function and what the code does, see Initialize the server process. For more information on the SDK actions and data types used in this code, read Amazon GameLift server SDK reference for C#.

This sample shows how to add all the required elements , as described in Add Amazon GameLift to your game server. It includes:

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

- Add Amazon GameLift to your game server
- Amazon GameLift server SDK reference for C#

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

public class ServerSDKManualTest : MonoBehaviour
{
    //This example is a simple integration that initializes a game server process
    //that is running on an Amazon GameLift Anywhere fleet.
    void Start()
    {
        //Identify port number (hard coded here for simplicity) the game server is
 listening on for player connections
        var listeningPort = 7777;

        //WebSocketUrl from RegisterHost call
        var webSocketUrl = "wss://us-west-2.api.amazongamelift.com";

        //Unique identifier for this process
        var processId = "myProcess";

        //Unique identifier for your host that this process belongs to
        var hostId = "myHost";

        //Unique identifier for your fleet that this host belongs to
        var fleetId = "myFleet";

        //Authorization token for this host process
        var authToken = "myAuthToken";

        //Server parameters are required for a GameLift Anywhere fleet.
        //They are not required for a GameLift managed EC2 fleet.
        ServerParameters serverParameters = new ServerParameters(
            webSocketUrl,
            processId,
```

```
            hostId,
            fleetId,
            authToken);

        //InitSDK establishes a local connection with an Amazon GameLift agent
        //to enable further communication.
        var initSDKOutcome = GameLiftServerAPI.InitSDK(serverParameters);
        if (initSDKOutcome.Success)
        {
            //Implement callback functions
            ProcessParameters processParameters = new ProcessParameters(
            //Implement OnStartGameSession callback
                (gameSession) => {
                    //GameLift sends a game session activation request to the game
server
                    //with game session object containing game properties and other
settings.
                    //Here is where a game server takes action based on the game
session object.
                    //When the game server is ready to receive incoming player
connections,
                    //it invokes the server SDK call ActivateGameSession().
                    GameLiftServerAPI.ActivateGameSession();
                },
                (updateGameSession) => {
                    //GameLift sends a request when a game session is updated (such as
for
                    //FlexMatch backfill) with an updated game session object.
                    //The game server can examine matchmakerData and handle new
incoming players.
                    //updateReason explains the purpose of the update.
                },
                () => {
                    //Implement callback function OnProcessTerminate
                    //GameLift invokes this callback before shutting down the instance
hosting this game server.
                    //It gives the game server a chance to save its state, communicate
with services, etc.,
                    //and initiate shut down. When the game server is ready to shut
down, it invokes the
                    //server SDK call ProcessEnding() to tell GameLift it is shutting
down.
                    GameLiftServerAPI.ProcessEnding();
                },
```

```
                () => {
                    //Implement callback function OnHealthCheck
                    //GameLift invokes this callback approximately every 60 seconds.
                    //A game server might want to check the health of dependencies,
etc.
                    //Then it returns health status true if healthy, false otherwise.
                    //The game server must respond within 60 seconds, or GameLift
records 'false'.
                    //In this example, the game server always reports healthy.
                    return true;
                },
                //The game server gets ready to report that it is ready to host game
sessions
                //and that it will listen on port 7777 for incoming player connections.
                listeningPort,
                new LogParameters(new List<string>()
                {
                    //Here, the game server tells GameLift where to find game session
log files.
                    //At the end of a game session, GameLift uploads everything in the
specified
                    //location and stores it in the cloud for access later.
                    "/local/game/logs/myserver.log"
                }));

            //The game server calls ProcessReady() to tell GameLift it's ready to host
game sessions.
            var processReadyOutcome =
GameLiftServerAPI.ProcessReady(processParameters);
            if (processReadyOutcome.Success)
            {
                print("ProcessReady success.");
            }
            else
            {
                print("ProcessReady failure : " +
processReadyOutcome.Error.ToString());
            }
        }
        else
        {
            print("InitSDK failure : " + initSDKOutcome.Error.ToString());
        }
    }
```

```
    void OnApplicationQuit()
    {
        //Make sure to call GameLiftServerAPI.ProcessEnding() and
 GameLiftServerAPI.Destroy() before terminating the server process.
        //These actions notify Amazon GameLift that the process is terminating and
 frees the API client from memory.
        GenericOutcome processEndingOutcome = GameLiftServerAPI.ProcessEnding();
        GameLiftServerAPI.Destroy();
        if (processEndingOutcome.Success)
        {
            Environment.Exit(0);
        }
        else
        {
            Console.WriteLine("ProcessEnding() failed. Error: " +
 processEndingOutcome.Error.ToString());
            Environment.Exit(-1);
        }
    }
}
```

Additional resources

Use the following resources to test your game server and expand the functionality:

- Set up your development machine as an Amazon GameLift Anywhere fleet and use it for local testing. See Test your custom server integration.
- Build your game server and upload the build to Amazon GameLift. See Upload a custom server build to Amazon GameLift.
- Deploy your game server build to an Amazon GameLift managed EC2 fleet. See Create a new Amazon GameLift fleet.

Test your integration using Amazon GameLift Anywhere fleets

You can use an Amazon GameLift Anywhere fleet to iteratively build and test your game integration with Amazon GameLift. Set up your own hardware as an Anywhere fleet with a connection to the Amazon GameLift service, then install and run your game server on it. Use a test app to run scenarios such as starting/stopping game sessions, tracking player connections, and

handling matchmaking backfills. With an Anywhere fleet, you can update your game server build as needed and have full visibility into hosting activity.

You can Amazon GameLift Anywhere fleets with games integrated with Amazon GameLift Server SDK version 5 or greater.

Topics

Initial development

You've developed your game and are integrating it with the Amazon GameLift server SDK. To test your integration, you could upload each new iteration of your game server build to Amazon GameLift and create a fleet. Alternatively, using an Anywhere fleet with your development laptop gives you a more efficient way to do iterative development and testing.

Use the following procedures to create an Anywhere fleet and start a game session on your laptop using the Amazon GameLift console or the AWS Command Line Interface (AWS CLI).

Console

1. Open the Amazon GameLift console.
2. In the navigation pane, under **Hosting**, choose **Locations**.
3. Choose **Create location**.
4. In the **Create location** dialog box, do the following:
    a. Enter a **Location name**. This labels the location of your compute resources that Amazon GameLift uses to run your games in Anywhere fleets. Custom location names must start with **custom-**.
    b. Choose **Create**.
5. To create an Anywhere fleet, do the following:
    a. In the navigation pane, under **Hosting**, choose **Fleets**.
    b. On the **Fleets** page, choose **Create fleet**.
    c. On the **Choose compute type** step, choose **Anywhere**, and then choose **Next**.

d. On the **Define fleet details** step, define your new fleet. For more information, see Create a new Amazon GameLift fleet.

e. On the **Select locations** step, select the custom location that you created.

f. Complete the remaining fleet creation steps to create your Anywhere fleet.

6. Register your laptop as a compute resource in the fleet that you created. Use the `register-compute` command (or the `RegisterCompute` API operation). Include the `fleet-id` created in the previous step and add a `compute-name` and your laptop's `ip-address`.

```
aws gamelift register-compute \
    --compute-name DevLaptop \
    --fleet-id fleet-1234 \
    --ip-address 10.1.2.3 \
    --location custom-location-1
```

Example output:

```
Compute {
    FleetId = fleet-1234,
    ComputeName = DevLaptop,
    Status = ACTIVE,
    IpAddress = 10.1.2.3,
    GameLiftServiceSdkEndpoint = wss://12345678.execute-api.amazonaws.com/,
    Location = custom-location-1
}
```

7. Start a debug session of your game server.

a. Get the authorization token for your laptop in the fleet that you created. Use the `get-compute-auth-token` command (or the `GetComputeAuthToken` API operation).

```
aws gamelift get-compute-auth-token \
    --fleet-id fleet-1234 \
    --compute-name DevLaptop
```

Example output:

```
ComputeAuthToken {
  FleetId = fleet-1234,
```

```
  ComputeName = DevLaptop,
  AuthToken = abcdefg123,
  ExpirationTime = 1897492857.11
}
```

b. Run a debug instance of your game server executable. To run the debug instance, your game server must call InitSDK(). After the process is ready to host a game session, the game server calls ProcessReady().

8. Create a game session to test out your first integration with Amazon GameLift Anywhere. Use the `create-game-session` command (or the `CreateGameSession` API operation). Specify the fleet's custom location.

```
aws gamelift create-game-session \
    --fleet-id fleet-1234 \
    --name DebugSession \
    --maximum-player-session-count 2 \
    --location custom-location-1
```

Example output:

```
GameSession {
    FleetId = fleet-1234,
    GameSessionId = 1111-1111,
    Name = DebugSession,
    IpAddress = 10.1.2.3,
    Port = 1024,
    ...
}
```

Amazon GameLift sends an onStartGameSession() message to your registered server process. The message contains the GameSession object from the previous step with game properties, game sessions data, matchmaker data, and more about the game session.

9. Add logic to your game server so that your server process responds to the onStartGameSession() message with ActivateGameSession(). The operation sends an acknowledgement to Amazon GameLift that your server received and accepted the create game session message. For more information see, Amazon GameLift server SDK reference.

Your game server is now running a game session for you to test out and use for iteration. To learn how to iterate on your game server, continue to the next section.

AWS CLI

1. Create a custom location using the `create-location` command (or the `CreateLocation` API operation). A custom location labels the location of your hardware that Amazon GameLift uses to run your games in Anywhere fleets.

```
aws gamelift create-location \
    --location-name custom-location-1
```

Example output:

```
{
    Location {
        LocationName = custom-location-1
    }
}
```

2. Create an Anywhere fleet with your custom location using the `create-fleet` command (or the `CreateFleet` API operation). Amazon GameLift creates the fleet in your home Region and the custom locations that you provide.

```
aws gamelift create-fleet \
    --name LaptopFleet \
    --compute-type ANYWHERE \
    --locations "location=custom-location-1"
```

Example output:

```
Fleet {
    Name = LaptopFleet,
    ComputeType = ANYWHERE,
    FleetId = fleet-1234,
    Status = ACTIVE
    ...
}
```

3. Register your laptop as a compute resource in the fleet that you created. Use the `register-compute` command (or the `RegisterCompute` API operation). Include the

`fleet-id` created in the previous step and add a `compute-name` and your laptop's public `ip-address`.

```
aws gamelift register-compute \
    --compute-name DevLaptop \
    --fleet-id fleet-1234 \
    --ip-address 10.1.2.3 \
    --location custom-location-1
```

Example output:

```
Compute {
    FleetId = fleet-1234,
    ComputeName = DevLaptop,
    Status = ACTIVE,
    IpAddress = 10.1.2.3,
    GameLiftServiceSdkEndpoint = wss://12345678.execute-api.amazonaws.com/,
    Location = custom-location-1
}
```

4. Start a debug session of your game server.

   a. Get the authorization token for your laptop in the fleet that you created. Use the [get-compute-auth-token] command (or the [GetComputeAuthToken] API operation).

   ```
   aws gamelift get-compute-auth-token \
       --fleet-id fleet-1234 \
       --compute-name DevLaptop
   ```

   Example output:

   ```
   ComputeAuthToken {
     FleetId = fleet-1234,
     ComputeName = DevLaptop,
     AuthToken = abcdefg123,
     ExpirationTime = 1897492857.11
   }
   ```

   b. Run a debug instance of your game server executable. To run the debug instance, your game server must call `InitSDK()`. After the process is ready to host a game session, the game server calls `ProcessReady()`.

5. Create a game session to test out your first integration with Amazon GameLift Anywhere. Use the `create-game-session` command (or the `CreateGameSession` API operation).

```
aws gamelift create-game-session \
    --fleet-id fleet-1234 \
    --name DebugSession \
    --maximum-player-session-count 2
```

Example output:

```
GameSession {
    FleetId = fleet-1234,
    GameSessionId = 1111-1111,
    Name = DebugSession,
    IpAddress = 10.1.2.3,
    Port = 1024,
    ...
}
```

Amazon GameLift sends an onStartGameSession() message to your registered server process. The message contains the GameSession object from the previous step with game properties, game sessions data, matchmaker data, and more about the game session.

6. Add logic to your game server so that your server process responds to the onStartGameSession() message with ActivateGameSession(). The operation sends an acknowledgement to Amazon GameLift that your server received and accepted the create game session message. For more information see, Amazon GameLift server SDK reference.

Your game server is now running a game session for you to test out and use for iteration. To learn how to iterate on your game server, continue to the next section.

Iterate on your game server

In this use case, consider a scenario where you've set up and tested your game server and found a bug. With Amazon GameLift Anywhere, you can iterate on your code and avoid the heavy setup of using an Amazon EC2 fleet.

1. Clean up your existing GameSession, if possible. If the game server crashes or it won't call ProcessEnding(), Amazon GameLift cleans up the GameSession after the game server stops sending health checks.
2. Make the code changes to your game server, compile, and prepare for the next test.
3. Your previous Anywhere fleet is still active and your laptop is still registered as a compute resource in the fleet. To begin testing again, create a new debug instance.

   a. Retrieve the authorization token for your laptop in the fleet that you created. Use the get-compute-auth-token command (or the GetComputeAuthToken API operation).

   ```
   aws gamelift get-compute-auth-token \
       --fleet-id fleet-1234 \
       --compute-name DevLaptop
   ```

   Example output:

   ```
   ComputeAuthToken {
     FleetId = fleet-1234,
     ComputeName = DevLaptop,
     AuthToken = hijklmnop456,
     ExpirationTime = 1897492857.11
   }
   ```

   b. Run a debug instance of your game server executable. To run the debug instance, your game server must call InitSDK(). After the process is ready to host a game session, the game server calls ProcessReady().

4. Your fleet now has an available server process. Create your game session and perform your next tests. Use the create-game-session command (or the CreateGameSession API operation).

   ```
   aws gamelift create-game-session \
       --fleet-id fleet-1234 \
       --name SecondDebugSession \
       --maximum-player-session-count 2
   ```

   Amazon GameLift sends an onStartGameSession() message to your registered server process. The message contains the GameSession object from the previous step with game properties, game session data, matchmaker data, and more about the game session.

5. Add logic to your game server so that your server process responds to the `onStartGameSession()` message with `ActivateGameSession()`. The operation sends an acknowledgement to Amazon GameLift that your server received and accepted the create game session message. For more information see, Amazon GameLift server SDK reference.

After you finish testing your game server, you can continue to use Amazon GameLift for your fleet and game server management. For more information, see Create a Amazon GameLift Anywhere fleet.

Test your integration using Amazon GameLift Local

> **Note**
>
> Use this testing procedure if you're using a version of the Amazon GameLift server SDK that is version 4.x or earlier. Your server SDK package includes a compatible version of Amazon GameLift Local. If you're using server SDK version 5.x, see Test your integration using Amazon GameLift Anywhere fleets for local testing with an Amazon GameLift Anywhere fleet.

Use Amazon GameLift Local to run a limited version of the managed Amazon GameLift service on a local device and test your game integration against it. This tool is useful when doing iterative development on your game integration. The alternative—uploading each new build to Amazon GameLift and configuring a fleet to host your game—can take 30 minutes or more each time.

With Amazon GameLift Local, you can verify the following:

- Your game server is correctly integrated with the Server SDK and is properly communicating with the Amazon GameLift service to start new game sessions, accept new players, and report health and status.
- Your game client is correctly integrated with the AWS SDK for Amazon GameLift and is able to retrieve information on existing game sessions, start new game sessions, join players to games and connect to the game session.

Amazon GameLift Local is a command-line tool that starts a self-contained version of the managed Amazon GameLift service. Amazon GameLift Local also provides a running event log of server process initialization, health checks, and API calls and responses. Amazon GameLift Local

recognizes a subset of the AWS SDK actions for Amazon GameLift. You can make calls from the AWS CLI or from your game client. All API actions perform locally just as they do in the Amazon GameLift web service.

Each server process should only host a single game session. The game session is the executable you use to connect to Amazon GameLift Local. When the game session is completed, you should call `GameLiftServerSDK::ProcessEndning` and then exit the process. When testing locally with Amazon GameLift Local, you can start multiple server processes. Each process will connect to Amazon GameLift Local. You can then create one game session for each server process. When your game session ends, your game server process should exit. You must then manually start another server process.

Amazon GameLift local supports the following APIs:

- CreateGameSession
- CreatePlayerSession
- CreatePlayerSessions
- DescribeGameSessions
- DescribePlayerSessions

Set up Amazon GameLift local

Amazon GameLift Local is provided as an executable `.jar` file bundled with the Server SDK. It can be run on Windows or Linux and used with any Amazon GameLift-supported language.

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

- A build of the Amazon GameLift Server SDK version 3.1.5 to 4.x.
- Java 8

Test a game server

If you want to test your game server only, you can use the AWS CLI to simulate game client calls to the Amazon GameLift Local service. This verifies that your game server is performing as expected with the following:

- The game server launches properly and initializes the Amazon GameLift Server SDK.

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

1. **Start Amazon GameLift Local.**

   Open a command prompt window, navigate to the directory containing the file *GameLiftLocal.jar* and run it. By default, Local listens for requests from game clients on port 8080. To specify a different port number, use the `-p` parameter, as shown in the following example:

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

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

2. **Start your game server.**

   Start your Amazon GameLift-integrated game server locally. You don't need to change the endpoint for the game server.

   In the Local command prompt window, log messages indicate that your game server has connected to the Amazon GameLift Local service. This means that your game server successfully initialized the Amazon GameLift Server SDK (with `InitSDK()`). It has called `ProcessReady()` with the log paths shown and, if successful, is ready to host a game session. While the game server is running, Amazon GameLift logs each health status report from the game server. The following log messaging example shows a successfully integrated game server:

   ```
   16:50:53,217  INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - SDK
    connected: /127.0.0.1:64247
   16:50:53,217  INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - SDK pid is 17040,
    sdkVersion is 3.1.5 and sdkLanguage is CSharp
   16:50:53,217  INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - NOTE: Only SDK
    versions 3.1.5 and above are supported in GameLiftLocal!
   ```

```
16:50:53,451  INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - onProcessReady
 received from: /127.0.0.1:64247 and ackRequest requested? true
16:50:53,543  INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - onProcessReady
 data: logPathsToUpload: "C:\\game\\logs"
logPathsToUpload: "C:\\game\\error"
port: 1935

16:50:53,544  INFO || - [HostProcessManager] nioEventLoopGroup-3-1 - Registered new
 process true, true,
16:50:53,558  INFO || - [SDKListenerImpl] nioEventLoopGroup-3-1 - onReportHealth
 received from /127.0.0.1:64247 with health status: healthy
```

Potential error and warning messages include the following:

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

3. **Start the AWS CLI.**

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

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

   In the AWS CLI command prompt window, `AWS gamelift` commands result in responses as documented in the AWS CLI Command Reference.

4. **Create a game session.**

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

   ```
   AWS gamelift create-game-session --endpoint-url http://localhost:9080 --maximum-
   player-session-count 2 --fleet-id
   ```

```
    fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d
```

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

```
13:57:36,129  INFO || - [SDKInvokerImpl]
        Thread-2 - Finished sending event to game server to start a game session:
        arn:aws:gamelift:local::gamesession/
fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d/gsess-ab423a4b-b827-4765-
aea2-54b3fa0818b6.
        Waiting for ack response.13:57:36,143  INFO || - [SDKInvokerImpl]
        Thread-2 - Received ack response: true13:57:36,144  INFO || -
        [CreateGameSessionDispatcher] Thread-2 - GameSession with id:
        arn:aws:gamelift:local::gamesession/
fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d/gsess-ab423a4b-b827-4765-
aea2-54b3fa0818b6
        created13:57:36,227  INFO || - [SDKListenerImpl]
        nioEventLoopGroup-3-1 - onGameSessionActivate received
 from: /127.0.0.1:60020 and ackRequest
        requested? true13:57:36,230  INFO || - [SDKListenerImpl]
        nioEventLoopGroup-3-1 - onGameSessionActivate data: gameSessionId:
        "arn:aws:gamelift:local::gamesession/
fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d/gsess-abcdef12-3456-7890-abcd-
ef1234567890"
```

In the AWS CLI window, Amazon 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().

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

```
    }
}
```

Test a game server and client

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

- The game client is successfully making AWS SDK requests to the Amazon GameLift Local service, including to create game sessions, retrieve information on existing game sessions, and create player sessions.
- The game server is correctly validating players when they try to join a game session. For validated players, the game server may retrieve player data (if implemented).
- The game server reports a dropped connection when a player leaves the game.
- The game server reports ending a game session.

1. **Start Amazon GameLift Local.**

   Open a command prompt window, navigate to the directory containing the file *GameLiftLocal.jar* and run it. By default, Local listens for requests from game clients on port 8080. To specify a different port number, use the `-p` parameter, as shown in the following example.

   ```
   ./gamelift-local -p 9080
   ```

   Once Local starts, you see logs showing that two local servers were started, one listening for your game server and one listening for your game client or the AWS CLI.

2. **Start your game server.**

   Start your Amazon GameLift-integrated game server locally. See Test a game server for more detail on message logs.

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

To use your game client with the Amazon GameLift Local service, you must make the following changes to your game client's setup, as described in Set up Amazon GameLift on a backend service:

- Change the `ClientConfiguration` object to point to your Local endpoint, such as `http://localhost:9080`.
- Set a target fleet ID value. For Local, you do not need a real fleet ID; set the target fleet to any valid string (`^fleet-\S+`), such as `fleet-1a2b3c4d-5e6f-7a8b-9c0d-1e2f3a4b5c6d`.
- Set AWS credentials. For Local, you do not need real AWS credentials; you can set the access key and secret key to any string.

In the Local command prompt window, once you start the game client, log messages should indicate that it has initialized the `GameLiftClient` and is successfully communicated with the Amazon GameLift service.

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

   Verify that your game client is successfully making any or all of the following API calls:

   - CreateGameSession()
   - DescribeGameSessions()
   - CreatePlayerSession()
   - CreatePlayerSessions()
   - DescribePlayerSessions()

   In the Local command prompt window, only calls to `CreateGameSession()` result in log messages. Log messages show when Amazon GameLift Local prompts your game server to start a game session (on`StartGameSession` callback) and gets a successful `ActivateGameSession` when your game server invokes it. In the AWS CLI window, all API calls result in responses or error messages as documented.

5. **Verify that your game server is validating new player connections.**

   After creating a game session and a player session, establish a direct connection to the game session.

In the Local command prompt window, log messages should show that the game server has sent an `AcceptPlayerSession()` request to validate the new player connection. If you use the AWS CLI to call `DescribePlayerSessions()`, the player session status should change from Reserved to Active.

6. **Verify that your game server is reporting game and player status to the Amazon GameLift service.**

   For Amazon GameLift to manage player demand and correctly report metrics, your game server must report various statuses back to Amazon GameLift. Verify that Local is logging events related to following actions. You may also want to use the AWS CLI to track status changes.

   - **Player disconnects from a game session** – Amazon GameLift Local log messages should show that your game server calls `RemovePlayerSession()`. An AWS CLI call to `DescribePlayerSessions()` should reflect a status change from `Active` to `Completed`. You might also call `DescribeGameSessions()` to check that the game session's current player count decreases by one.
   - **Game session ends** – Amazon GameLift Local log messages should show that your game server calls `TerminateGameSession()`.

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

   - **Server process is terminated** – Amazon 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 Amazon GameLift Local, keep in mind the following:

- Unlike the Amazon GameLift web service, Local does not track a server's health status and initiate the on`ProcessTerminate` 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

This topic provides an overview of the managed Amazon GameLift with Realtime Servers solution. The overview explains when this solution is a good fit for your game, and how Realtime Servers supports multiplayer gaming.

For a complete roadmap to getting your game up and running, see Amazon GameLift managed hosting roadmap.

> **Tip**
>
> To try out Amazon GameLift game server hosting, see Getting started with Amazon GameLift.

What are Realtime servers?

Realtime servers are lightweight, ready-to-go game servers that Amazon GameLift provides for you to use with your multiplayer games. Realtime servers remove the development, testing, and deployment process of a custom game server. This solution can help minimize the time and effort required to complete your game.

Key features

- Full network stack for game client and server interaction
- Core game server functionality
- Customizable server logic
- Live updates to Realtime configurations and server logic
- FlexMatch matchmaking

- Flexible control of hosting resources

Set up Realtime servers by creating a fleet and providing a configuration script. For more information about creating Realtime servers and how to prepare your game client, see Prepare your Realtime server.

How Realtime Servers manages game sessions

You can add custom logic for game session management by building it into the Realtime script. You can write code to access server-specific objects, add event-driven logic using callbacks, or add logic based on non-event scenarios.

How Realtime clients and servers interact

During a game session, game clients interact by sending messages to the Realtime server through a backend service. The backend service then relays the messages among game clients to exchange activity, game state, and relevant game data.

In addition, you can customize how clients and servers interact by adding game logic to the Realtime script. With custom game logic, a Realtime server might implement callbacks to start event-driven responses.

Communication protocol

Realtime servers and connected game clients communicate through 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 isn't available, Amazon GameLift sends messages 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, and the payload provides additional data related to the operation code. Both of these elements are developer-defined. Your game client acts based on the opCodes in the messages that it receives.

Player groups

Realtime Servers provides functionality to manage groups of players. By default, Amazon GameLift places all players who connect to a game 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 and share game data with all players in the group. One possible use for groups is to set up player teams and manage team communication.

Realtime Servers with TLS certificates

With Realtime Servers, server authentication and data packet encryption are built into the service. These security features are enabled when you turn on TLS certificate generation. When a game client tries to connect with a Realtime server, the server automatically responds with the TLS certificate, which the client validates. Amazon GameLift handles encryption using TLS for TCP (WebSockets) communication and DTLS for UDP traffic.

Customizing a Realtime server

A Realtime server performs as a stateless relay server. The Realtime server relays packets of messages and game data between the game clients connected to the game. However, the Realtime server doesn't 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 through the relay server. Each game client is responsible for incorporating these updates and reconciling its own game state.

You can customize your servers by adding to the Realtime script functionality. With game logic, for example, you can build a stateful game with a server-authoritative view of the game state.

Amazon GameLift defines a set of server-side callbacks for Realtime scripts. Implement these callbacks to add event-driven functionality to your server. For example, you can:

- Authenticate a player when a game client tries to connect to the server.
- Validate whether a player can join a group upon request.
- Determine when to deliver messages from a certain player or to a target player, or perform additional processing in response.
- Notify all players when a player leaves a group or disconnects from the server.
- View the content of game session objects or message objects, and use the data.

Deploying and updating Realtime Servers

A key advantage of Realtime Servers is the ability to update your scripts at any time. When you update a script, Amazon GameLift distributes the new version to all hosting resources within

minutes. After Amazon GameLift deploys the new script, all new game sessions created after that point will use the new script version. (Existing game sessions will continue to use the original version.)

Get started integrating your game with Realtime Servers:

- Integrating a game client for Realtime Servers
- Creating a Realtime script

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 backend service and use it to make the direct requests to the Amazon GameLift service when triggered by a game client action. The backend service then relays relevant responses back to the game client.

1. Add the AWS SDK to your game client, initialize an Amazon GameLift client, and configure it to use the hosting resources in your fleets and queues. The AWS SDK is available in several languages; see the Amazon GameLift SDKs For custom client services.
2. Add GameLift functionality to your backend service. For more detailed instructions, see Add Amazon GameLift to your game client 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 backend 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 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() 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 for more information.
   - Connect to and disconnect from a game
     - Connect()
     - Disconnect()
   - Send messages to target recipients
     - SendMessage()
   - Receive and process messages
     - OnDataReceived()
   - Join groups and leave player groups
     - JoinGroup()
     - RequestGroupMembership()
     - LeaveGroup()

4. Set up event handlers for the client callbacks as needed. See Realtime Servers client API (C#) reference: Asynchronous callbacks.

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.

```
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
 ClientConfiguration
          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 i.e. 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)
        {
        }

        /**
         * Handle connection close events
         */
        public void OnCloseEvent(object sender, EventArgs e)
        {
        }

        /**
```

```
         * Handle Group membership update events
         */
        public void OnGroupMembershipUpdate(object sender, GroupMembershipEventArgs e)
        {
        }

        /**
         *  Handle data received from the Realtime server
         */
        public virtual void OnDataReceived(object sender, DataReceivedEventArgs e)
        {
            switch (e.OpCode)
            {
                // handle message based on OpCode
                default:
                    break;
            }
        }

        /**
         * Helper method to simplify task of sending/receiving payloads.
         */
        public static byte[] StringToBytes(string str)
        {
            return Encoding.UTF8.GetBytes(str);
        }

        /**
         * Helper method to simplify task of sending/receiving payloads.
         */
        public static string BytesToString(byte[] bytes)
        {
            return Encoding.UTF8.GetString(bytes);
        }
    }
}
```

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 Amazon GameLift).

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 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.

- **Add event-driven logic.** Implement the callback functions to respond to client-server events. See Script callbacks for Realtime Servers 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).
- Enable game logic to access your other AWS resources. For details, see Communicate with other AWS resources from your fleets.

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

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.

```
// 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 to 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, playerId);
        }
    });
    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
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
// Use gameMessage.getPayloadAsText() to get the message contents
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
};
```

Integrating games with the Amazon GameLift plugin for Unity

The topics in this section describe the Amazon GameLift plugin for Unity and how to use it to prepare your multiplayer game project for hosting with Amazon GameLift. Work entirely in your Unity development environment with the plugin's guided workflows to complete the basic requirements for hosting with Amazon GameLift.

Amazon GameLift is a fully managed service that lets game developers manage and scale dedicated game servers for session-based multiplayer games. For more information about Amazon GameLift hosting, see How Amazon GameLift works.

- Amazon GameLift plugin for Unity guide for server SDK 5.x, version 2.0.0, works with server SDK 5.x and supports Amazon GameLift Anywhere.

- Amazon GameLift plugin for Unity guide for server SDK 4.x, version 1.0.0, works with server SDK 4.x or earlier. This version uses Amazon GameLift Local for integration testing.

Amazon GameLift plugin for Unity guide for server SDK 5.x

Amazon GameLift provides tools for preparing your multiplayer game servers to work with Amazon GameLift. The Amazon GameLift plugin for Unity makes it easier to integrate Amazon GameLift into your Unity game projects, test your integration with Amazon GameLift Anywhere, and deploy Amazon GameLift resources for cloud hosting.

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

Topics

About the plugin

The plugin for Unity provides a streamlined getting started experience for integrating and hosting your Unity multiplayer games with Amazon GameLift. You can take advantage of plugin functionality and pre-built components to quickly get your games up and running.

The plugin adds tools and functionality to the Unity editor. Use the guided workflows to integrate Amazon GameLift into your game project, test it locally, and then deploy the game server to Amazon GameLift cloud hosting.

Use the plugin's pre-built hosting solutions to deploy your game. Set up an Amazon GameLift Anywhere fleet with your local workstation as a host. For cloud hosting, choose from two common deployment scenarios that balance player latency, game session availability, and cost in different ways. One scenario includes a simple FlexMatch matchmaker and rule set. Use these scenarios to put a basic production-ready hosting solution in place, and then optimize and customize as needed.

The plugin includes these components:

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

Plugin workflow

The following steps describe a typical approach to integrating and deploying a game project with the Amazon GameLift plugin for Unity. You complete these steps by working in the Unity editor and your game code.

1. Create a user profile that links to your AWS account and provides access credentials for a valid account user with permissions to use Amazon GameLift.
2. Add server code to your game project to establish communication between a running game server and the with Amazon GameLift service.
3. Add client code to your game project that lets game clients send requests to Amazon GameLift to start or join a game session and then connect to the game server.
4. Use the Anywhere workflow to set up your local workstation as an Anywhere host for your game server. Launch your game server and client locally, connect to a game session, and test your integration.
5. Use the EC2 hosting workflow to upload your integrated game server and deploy a cloud hosting solution. When your game server is ready, launch your game client locally, connect to a game session and log in, and play the game.

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

Install the plugin for Unity

This section describes how to add the plugin to a Unity project. After the plugin is installed, plugin functionality is available when you have the project open in the Unity editor.

Before you start

Here's what you need to use the Amazon GameLift plugin for Unity:

- Unity for Windows 2022 LTS or Unity for MacOS
- Amazon GameLift plugin for Unity download. [Download site] The download includes two packages:
  - Amazon GameLift standalone plugin for Unity
  - Amazon GameLift C# server SDK for Unity
- Microsoft Visual Studio 2019 or newer.
- A multiplayer game project with C# game code.
- The third party scoped registry UnityNuGet. This tool manages third-party DLLs. For more information, see the UnityNuGet Github repository.

Add the plugin to your game project

Complete the following tasks, working in the Unity editor and your game project files.

Step 1: Add UnityNuGet to your game project

If you don't have UnityNuGet set up for your game project, use the following steps to install the tool using the Unity package manager. Alternatively, you can use the NuGet CLI to manually download the DLLs. For more information, see the Amazon GameLift C# server SDK for Unity README.

1. With your project open in the Unity editor, go to the main menu and select **Edit, Project Settings**. From the options, choose the **Package Manager** section and open the **Scoped Registries** group.
2. Choose the **+** button and enter the following values for the UnityNuGet scoped registry:

```
Name: Unity NuGet
URL: https://unitynuget-registry.azurewebsites.net
```

```
Scope(s): org.nuget
```

For Unity 2021 version users:

After setting up UnityNuGet, check for `Assembly Version Validation` errors showing in the Unity console. These errors occur if binding redirects for strongly named assemblies in the NuGet packages are not resolving correctly to paths within your Unity project. To resolve this issue, configure Unity's assembly version validation:

1. In the Unity editor, go to the main menu and select **Edit, Project Settings**, and open the **Player** section.
2. Deselect the **Assembly Version Validation** option.

Step 2: Add the plugin and C# server SDK packages

1. Unzip the Amazon GameLift plugin for Unity download, which contains both packages.
2. With your project open in the Unity Editor, go to the main menu and select **Window, Package Manager**.
3. Choose the **+** button to add a new package. Choose the option **Add package from tarball**.
4. In **Select packages on disk**, locate the Amazon GameLift C# Server SDK plugin for Unity download files, and choose the `com.amazonaws.gameliftserver.sdk-<version>.tgz` file. Choose **Open** to install the plugin.
5. In **Select packages on disk**, locate the Amazon GameLift standalone plugin for Unity download files, and choose the file `com.amazonaws.gamelift-<version>.tgz`. Choose **Open** to install the plugin.
6. Verify that the standalone plugin is added to your project. Return to the Unity editor window. Check the main menu for the new **Amazon GameLift** menu button.

Step 3: Import the sample game (optional)

The plugin for Unity comes with a set of sample game assets, including scenes, that you can add to your game project. Importing the sample game gives you a fast path to testing, building, and deploying a simple multiplayer game with Amazon GameLift. The sample game is already fully integrated with Amazon GameLift SDKs, so you can skip the integration tasks and complete the remaining workflow tasks.

When using the sample game, you can set up and join a locally hosted Amazon GameLift Anywhere fleet in just a few minutes. You can deploy the game to Amazon GameLift and join a live, cloud-hosted game in under an hour.

To import the sample game:

1. With your game project open in the Unity Editor, go to the **Amazon GameLift** menu and select **Sample Game, Import Sample Game**.
2. After the files are imported, go to the **Amazon GameLift** menu again and select **Sample Game, Initialize Settings**. This step configures your project for building the game client and server.

When installation is complete, you'll see two new scenes added to your game project. You'll also see some additional project assets, including a **GameLiftClientSettings** asset.

For more details on the sample's UI and gameplay, see the sample game readme.

Set up an AWS user profile

After installing the plugin, set up a profile and link it to a valid AWS account user. You can maintain multiple profiles, but you can only have one profile active at a time. Whenever you work in the plugin, select a profile to use.

Maintaining multiple profiles gives you the ability to switch between different hosting scenarios. For example, you might set up profiles with the same AWS credentials but different AWS Regions. Or you might set up profiles with different AWS accounts or with different users/permission sets.

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

To set up your AWS profile

1. In the Unity editor main menu, choose **Amazon GameLift** and select **Set AWS Account Profiles**. This action opens the plugin window. Open the page **AWS User Profiles**.

2. If the plugin detects an existing profile, you won't be prompted to create one. Select an existing profile or choose **Add another profile** to create a new one.
3. If the plugin doesn't detect an existing profile, it prompts you to create one. You can create a new profile using either a new or existing AWS account.

   > **Note**
   >
   > You need to use the AWS Management Console to create a new AWS account and create or update a user with the proper permission set.

   When setting up a profile, you need the following information:

   - An AWS account. If you need to create a new AWS account, follow the prompts to create the account. See Create an AWS account for more details.
   - An AWS account user with permissions to use Amazon GameLift and other required AWS services. See Set up an AWS account for instructions on setting up an AWS Identity and Access Management (IAM) user with Amazon GameLift permissions.
   - Credentials for your AWS user. This user also needs programmatic access with long-term credentials. These credentials consist of an AWS access key ID and AWS secret key. See Get your access keys for more details.
   - AWS Region. This is a geographic location where you want to create your AWS resources for hosting. During development, we recommend using a region close to your physical location to minimize latency. See the list of supported AWS regions.
4. When you selected or created a profile, check the profile's bootstrap status and take action as needed. All profiles must be bootstrapped to use Amazon GameLift plugin functionality.

To bootstrap your profile:

Bootstrapping designates an Amazon S3 bucket for use with the selected user profile. Amazon S3 is a core AWS service for data and object storage. The bucket used to store project configurations, build artifacts, and other dependencies. Buckets are not shared between other profiles.

> **Note**
>
> Bootstrapping creates new AWS resources and might incur costs.

1. When viewing your profiles in the plugin window **AWS User Profiles**, select the profile you want to use. A warning message is displayed if the profile hasn't been bootstrapped yet.
2. In the **Bootstrap your profile** section, select a profile from the dropdown list and check the bootstrap status. If the status indicates that no bucket exists, choose the button **Bootstrap profile**. You can set the bucket name to a new bucket name, enter an existing bucket that you have access to, or keep the auto-generated name.
3. Wait for bootstrap status to change to "Active". This can take a few minutes. When the status is "Active", you can use the profile to work with plugin features

Set up your game for local testing with Amazon GameLift Anywhere

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

To start the Amazon GameLift Anywhere workflow:

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

Set your profile

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

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

Integrate your game with Amazon GameLift

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

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

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

Integrate your server code

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

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

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

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

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

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

- Add Amazon GameLift to your game server
- Amazon GameLift server SDK 5.x reference for C# and Unity

Integrate your client code

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

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

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

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

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

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

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

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

Connect to an Anywhere fleet

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

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

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

Register a compute

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

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

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

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

Launch game

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

1. Configure your game client. In this step, you prompt the plugin to update a `GameLiftClientSettings` asset for your game project. The plugin uses this asset to store certain information that your game client needs to connect to the Amazon GameLift service.

   a. If you didn't import and initialize the sample game, create a new `GameLiftClientSettings` asset. In the Unity editor main menu, choose **Assets, Create, GameLift, Client Settings**. If you create multiple copies of GameLiftClientSettings in your project, the plugin automatically detects this and notifies you which asset the plugin will update.

   b. In **Launch Game**, choose **Configure Client: Apply Anywhere Settings**. This action updates your game client settings to use the Anywhere fleet that you just set up.

2. Build and run your game client.

   a. Build a client executable using the standard Unity build process. In **File, Build Settings**, switch the platform to **Windows, Mac, Linux**. If you imported the sample game and initialized the settings, the build list and build target are automatically updated.

   b. Launch one or more instances of the newly built game client executable.

3. Launch a game server in your Anywhere fleet. Choose **Server: Launch Server** in Editor. This task starts a live server that your client can connect to as long as the Unity editor remains open.

4. Start or join a game session. In your game client instances, use the UI to join each client to a game session. How you do this depends on how you added functionality to the client.

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

- A player login component. When connecting to a game server on an Anywhere fleet, there is no player validation. You can enter any values to join the game session.
- A simple join game UI. When a client attempts to join a game, the client automatically looks for an active game session with an available player slot. If no game session is available, the

client requests a new game session. If a game session is available, the client requests to join the available game session. When testing your game with multiple concurrent clients, the first client starts the game session, and the remaining clients automatically join the existing game session.

- Game sessions with four player slots. You can launch up to four game client instances concurrently and they will join the same game session.

Launch from a server executable (optional)

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

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

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

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

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

Deploy your game to cloud hosting with managed EC2 fleets

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

To start the Amazon GameLift managed Amazon EC2 workflow:

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

Set your profile

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

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

Integrate your game with Amazon GameLift

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

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

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

Integrate your server code

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

1. In your game project files, open the `Assets/Scripts/Server` folder. If it doesn't exist, create it.

2. Go to the GitHub repo aws/amazon-gamelift-plugin-unity and open the path `Samples~/SampleGame/Assets/Scripts/Server`.
3. Locate the file `GameLiftServer.cs` and copy it into your game project's `Server` folder. When you build a server executable, use this file as the build target.

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

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

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

- Add Amazon GameLift to your game server
- Amazon GameLift server SDK 5.x reference for C# and Unity

Integrate your client code

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

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

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

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

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

Use the following resources as a guide.

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

Select deployment scenario

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

- **Single-region fleet:** Deploys your game server to a single fleet of hosting resources in the active profile's default AWS region. This scenario is a good starting point for testing your server integration with AWS and server build configuration. It deploys the following resources:
  - AWS fleet (On-Demand) with your game server build installed and running.
  - Amazon Cognito user pool and client to enable players to authenticate and start a game.
  - API gateway authorizer that links user pool with APIs.
  - WebACl for throttling excessive player calls to API gateway.
  - API gateway + Lambda function for players to request a game slot. This function calls `CreateGameSession()` if none are available.
  - API gateway + Lambda function for players to get connection info for their game request.
- **FlexMatch fleet:** Deploys your game server to a set of fleets and sets up a FlexMatch matchmaker with rules to create player matches. This scenario uses low-cost Spot hosting with a multi-fleet, multi-location structure for durable availability. This approach is useful when you're ready to start designing a matchmaker component for your hosting solution. In this scenario,

you'll create the basic resources for this solution, which you can customize later as needed. It deploys the following resources:

- FlexMatch matchmaking configuration and matchmaking rule set to accept player requests and form matches.
- Three AWS fleets with your game server build installed and running in multiple locations. Includes two Spot fleets and one On-Demand fleet as a backup.
- AWS game session placement queue that fulfills requests for proposed matches by finding the best possible hosting resource (based on viability, cost, player latency, etc.) and starting a game session.
- Amazon Cognito user pool and client to enable players to authenticate and start a game.
- API gateway authorizer that links user pool with APIs.
- WebACl for throttling excessive player calls to API gateway.
- API gateway + Lambda function for players to request a game slot. This function calls `StartMatchmaking()`.
- API gateway + Lambda function for players to get connection info for their game request.
- Amazon DynamoDB tables to store matchmaking tickets for players and game session information. .
- SNS topic + Lambda function to handle GameSessionQueue events.

Set game parameters

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

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

Deploy scenario

In this step, you deploy your game to a cloud hosting solution based on the deployment scenario you chose. This process can take as long as 40 minutes while AWS validates your server build, provisions hosting resources, installs your game server, launches server processes, and gets them ready to host game sessions.

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

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

Launch game client

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

1. Configure your game client. In this step, you prompt the plugin to update a `GameLiftClientSettings` asset for your game project. The plugin uses this asset to store certain information that your game client needs to connect to the Amazon GameLift service.

    a. If you didn't import and initialize the sample game, create a new `GameLiftClientSettings` asset. In the Unity editor main menu, choose **Assets, Create, GameLift, Client Settings**. If you create multiple copies of GameLiftClientSettings in your project, the plugin automatically detects this and notifies you which asset the plugin will update.

    b. In **Launch Game**, choose **Configure Client: Apply Managed EC2 Settings**. This action updates your game client settings to use the managed EC2 fleet that you just deployed.
2. Build your game client. Build a client executable using the standard Unity build process. In File, Build Settings, switch the platform to Windows, Mac, Linux. If you imported the sample game and initialized the settings, the build list and build target are automatically updated.
3. Launch the newly build game client executable. To start playing the game, start two to four client instances and use the UI in each to join a game session.

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

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

Amazon GameLift plugin for Unity guide for server SDK 4.x

> **Note**
>
> This topic provides information for an earlier version of the Amazon GameLift plugin for Unity. Version 1.0.0 (released in 2021) uses the Amazon GameLift server SDK 4.x or earlier. For documentation on the latest version of the plugin, which uses server SDK 5.x and supports Amazon GameLift Anywhere, see Amazon GameLift plugin for Unity guide for server SDK 5.x.

Amazon GameLift provides tools for preparing your multiplayer game servers to run on Amazon GameLift. The Amazon GameLift plugin for Unity makes it easier to integrate Amazon GameLift into your Unity game projects and deploy Amazon GameLift resources for cloud hosting. Use the plugin for Unity to access Amazon GameLift APIs and deploy AWS CloudFormation templates for common gaming scenarios.

After you've set up the plugin, you can try out the Amazon GameLift Unity sample on GitHub.

Topics

- Integrate games with Amazon GameLift in Unity
- Import and run a sample game

Integrate Amazon GameLift with a Unity game server project

> **Note**
>
> This topic provides information for an earlier version of the Amazon GameLift plugin for Unity. Version 1.0.0 (released in 2021) uses the Amazon GameLift server SDK 4.x or earlier. For documentation on the latest version of the plugin, which uses server SDK 5.x and supports Amazon GameLift Anywhere, see Amazon GameLift plugin for Unity guide for server SDK 5.x.

This topic helps you prepare your custom game server for hosting on Amazon GameLift. The game server must be able to notify Amazon GameLift about its status, to start and stop game sessions when prompted, and to perform other tasks. For more information, see Add Amazon GameLift to your game server.

Prerequisites

Before integrating your game server, complete the following tasks:

- Set up an IAM service role for Amazon GameLift
- Install the plugin for Unity

Set up a new server process

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

Set up communication with Amazon GameLift and report that the server process is ready to host a game session.

1. Initialize the server SDK by calling `InitSDK()`.

2. To prepare the server to accept a game session, call `ProcessReady()` with the connection port and game session location details. Include the names of callback functions that Amazon GameLift service invokes, such as `OnGameSession()`, `OnGameSessionUpdate()`, `OnProcessTerminate()`, `OnHealthCheck()`. Amazon GameLift might take a few minutes to provide a callback.
3. Amazon GameLift updates the status of the server process to `ACTIVE`.
4. Amazon GameLift calls `onHealthCheck` periodically.

The following code example shows how to set up a simple server process with Amazon GameLift.

```
//initSDK
var initSDKOutcome = GameLiftServerAPI.InitSDK();

//processReady
// Set parameters and call ProcessReady
var processParams = new ProcessParameters(
    this.OnGameSession,
    this.OnProcessTerminate,
    this.OnHealthCheck,
    this.OnGameSessionUpdate,
    port,
    // Examples of log and error files written by the game server
    new LogParameters(new List<string>()
        {
            "C:\\game\\logs",
            "C:\\game\\error"
        })
);

var processReadyOutcome = GameLiftServerAPI.ProcessReady(processParams);

// Implement callback functions
void OnGameSession(GameSession gameSession)
{
    // game-specific tasks when starting a new game session, such as loading map
    // When ready to receive players
    var activateGameSessionOutcome = GameLiftServerAPI.ActivateGameSession();
}

void OnProcessTerminate()
{
```

```
    // game-specific tasks required to gracefully shut down a game session,
    // such as notifying players, preserving game state data, and other cleanup
    var ProcessEndingOutcome = GameLiftServerAPI.ProcessEnding();
}

bool OnHealthCheck()
{
    bool isHealthy;
    // complete health evaluation within 60 seconds and set health
    return isHealthy;
}
```

Start a game session

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

After game initialization is complete, you can start a game session.

1. Implement the callback function on`StartGameSession`. Amazon GameLift invokes this method to start a new game session on the server process and receive player connections.
2. To activate a game session, call `ActivateGameSession()`. For more information about the SDK, see Amazon GameLift server SDK (C#) reference: Actions.

The following code example illustrates how to start a game session with Amazon GameLift.

```
void OnStartGameSession(GameSession gameSession)
{
    // game-specific tasks when starting a new game session, such as loading map
    ...
    // When ready to receive players
    var activateGameSessionOutcome = GameLiftServerAPI.ActivateGameSession();
}
```

End a game session

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

Notify Amazon GameLift when a game session is ending. As a best practice, shut down server processes after game sessions complete to recycle and refresh hosting resources.

1. Set up a function named on`ProcessTerminate` to receive requests from Amazon GameLift and call `ProcessEnding()`.
2. The process status changes to `TERMINATED`.

The following example describes how to end a process for a game session.

```
var processEndingOutcome = GameLiftServerAPI.ProcessEnding();

if (processReadyOutcome.Success)
   Environment.Exit(0);

// otherwise, exit with error code
Environment.Exit(errorCode);
```

Create server build and upload to Amazon GameLift

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

After you integrate your game server with Amazon GameLift, upload the build files to a fleet so that Amazon GameLift can deploy it for game hosting. For more information on how to upload your server to Amazon GameLift, see Upload a custom server build to Amazon GameLift.

Integrate Amazon GameLift with a Unity game client project

> **Note**
>
> This topic provides information for an earlier version of the Amazon GameLift plugin for Unity. Version 1.0.0 (released in 2021) uses the Amazon GameLift server SDK 4.x or earlier. For documentation on the latest version of the plugin, which uses server SDK 5.x and supports Amazon GameLift Anywhere, see Amazon GameLift plugin for Unity guide for server SDK 5.x.

This topic helps you set up a game client to connect to Amazon GameLift hosted game sessions through a backend service. Use Amazon GameLift APIs to initiate matchmaking, request game session placement, and more.

Add code to the backend service project to allow communication with the Amazon GameLift service. A backend service handles all game client communication with the GameLift service. For more information about backend services, see Design your game client service.

A backend server handles the following game client tasks:

- Customize authentication for your players.
- Request information about active game sessions from the Amazon GameLift service.
- Create a new game session.
- Add a player to an existing game session.
- Remove a player from an existing game session.

Topics

Prerequisites

Before setting up game server communication with the Amazon GameLift client, complete the following tasks:

- Set up an AWS account
- Install the plugin for Unity
- Integrate Amazon GameLift with a Unity game server project
- Setting up Amazon GameLift fleets

Initialize a game client

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

Add code to initialize a game client. Run this code on launch, it's necessary for other Amazon GameLift functions.

1. Initialize `AmazonGameLiftClient`. Call `AmazonGameLiftClient` with either a default client configuration or a custom configuration. For more information on how to configure a client, see Set up Amazon GameLift on a backend service.
2. Generate a unique player id for each player to connect to a game session. For more information see Generate player IDs.

   The following examples shows how to set up a Amazon GameLift client.

```
public class GameLiftClient
{
    private GameLift gl;
    //A sample way to generate random player IDs.
    bool includeBrackets = false;
    bool includeDashes = true;
    string playerId = AZ::Uuid::CreateRandom().ToString<string>(includeBrackets,
 includeDashes);
```

```
    private Amazon.GameLift.Model.PlayerSession psession = null;
    public AmazonGameLiftClient aglc = null;

    public void CreateGameLiftClient()
    {
        //Access Amazon GameLift service by setting up a configuration.
        //The default configuration specifies a location.
        var config = new AmazonGameLiftConfig();
        config.RegionEndpoint = Amazon.RegionEndpoint.USEast1;

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

    }
}
```

Create game session on a specific fleet

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

Add code to start new game sessions on your deployed fleets and make them available to players. After Amazon GameLift has created the new game session and returned a `GameSession`, you can add players to it.

- Place a request for a new game session.
  - If your game uses fleets, call `CreateGameSession()` with a fleet or alias ID, a session name, and maximum number of concurrent players for the game.
  - If your game uses queues, call `StartGameSessionPlacement()`.

The following example shows how to create a game session.

```
public Amazon.GameLift.Model.GameSession()
{
    var cgsreq = new Amazon.GameLift.Model.CreateGameSessionRequest();
    //A unique identifier for the alias with the fleet to create a game session in.
    cgsreq.AliasId = aliasId;
    //A unique identifier for a player or entity creating the game session
    cgsreq.CreatorId = playerId;
    //The maximum number of players that can be connected simultaneously to the game
 session.
    cgsreq.MaximumPlayerSessionCount = 4;

    //Prompt an available server process to start a game session and retrieves
 connection information for the new game session
    Amazon.GameLift.Model.CreateGameSessionResponse cgsres =
 aglc.CreateGameSession(cgsreq);
    string gsid = cgsres.GameSession != null ? cgsres.GameSession.GameSessionId : "N/
A";
    Debug.Log((int)cgsres.HttpStatusCode + " GAME SESSION CREATED: " + gsid);
    return cgsres.GameSession;
}
```

Add players to game sessions

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

After Amazon GameLift has created the new game session and returned a `GameSession` object, you can add players to it.

1. Reserve a player slot in a game session by creating a new player session. Use `CreatePlayerSession` or `CreatePlayerSessions` with the game session ID and a unique ID for each player.
2. Connect to the game session. Retrieve the `PlayerSession` object to get the game session's connection information. You can use this information to establish a direct connection to the server process:

    a. Use the specified port and either the DNS name or IP address of the server process.

b. Use the DNS name and port of your fleets. The DNS name and port are required if your fleets have TLS certificate generation enabled.

c. Reference the player session ID. The player session ID is required if your game server validates incoming player connections.

The following examples demonstrates how to reserve a player spot in a game session.

```
public Amazon.GameLift.Model.PlayerSession
 CreatePlayerSession(Amazon.GameLift.Model.GameSession gsession)
{
    var cpsreq = new Amazon.GameLift.Model.CreatePlayerSessionRequest();
    cpsreq.GameSessionId = gsession.GameSessionId;
    //Specify game session ID.
    cpsreq.PlayerId = playerId;
    //Specify player ID.
    Amazon.GameLift.Model.CreatePlayerSessionResponse cpsres =
 aglc.CreatePlayerSession(cpsreq);
    string psid = cpsres.PlayerSession != null ? cpsres.PlayerSession.PlayerSessionId :
 "N/A";
    return cpsres.PlayerSession;
}
```

The following code illustrates how to connect a player with the game session.

```
public bool ConnectPlayer(int playerIdx, string playerSessionId)
{
    //Call ConnectPlayer with player ID and player session ID.
    return server.ConnectPlayer(playerIdx, playerSessionId);
}
```

Remove a player from a game session

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

You can remove the players from the game session when they leave the game.

1. Notify the Amazon GameLift service that a player has disconnected from the server process. Call `RemovePlayerSession` with the player's session ID.
2. Verify that `RemovePlayerSession` returns `Success`. Then, Amazon GameLift changes the player slot to be available, which Amazon GameLift can assign to a new player.

The following example illustrates how to remove a player session.

```
public void DisconnectPlayer(int playerIdx)
{
    //Receive the player session ID.
    string playerSessionId = playerSessions[playerIdx];
    var outcome = GameLiftServerAPI.RemovePlayerSession(playerSessionId);
    if (outcome.Success)
    {
        Debug.Log (":) PLAYER SESSION REMOVED");
    }
    else
    {
        Debug.Log(":(PLAYER SESSION REMOVE FAILED. RemovePlayerSession()
        returned " + outcome.Error.ToString());
    }
}
```

Install and set up the plugin

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

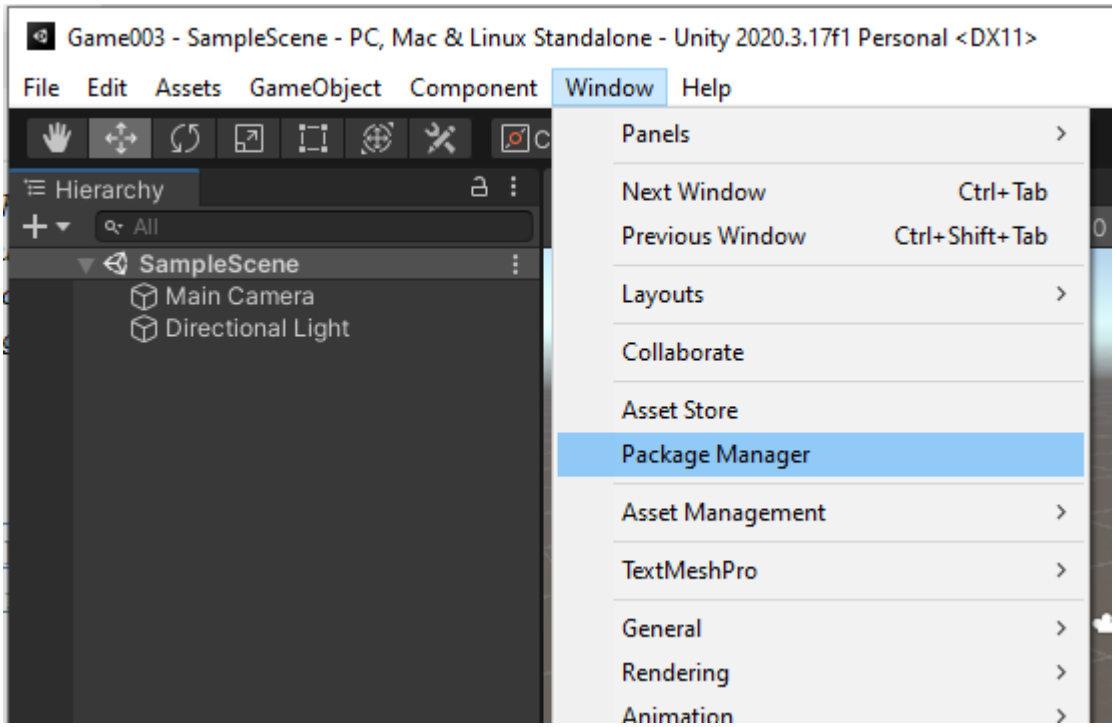
This section describes how to download, install, and set up the Amazon GameLift plugin for Unity, version 1.0.0.

Prerequisites

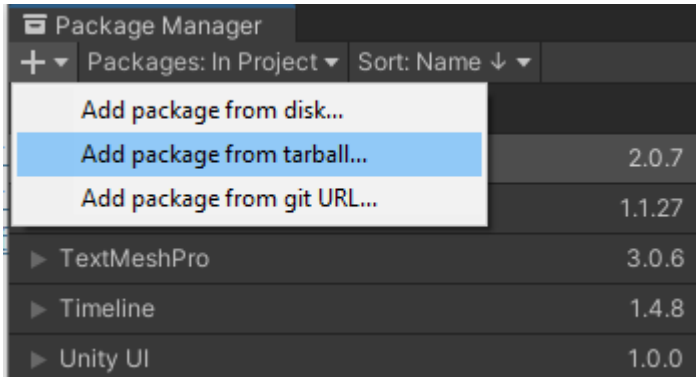
- Unity for Windows 2019.4 LTS, Windows 2020.3 LTS, or Unity for MacOS
- Current version of Java
- Current version of .NET 4.x

To download and install the plugin for Unity

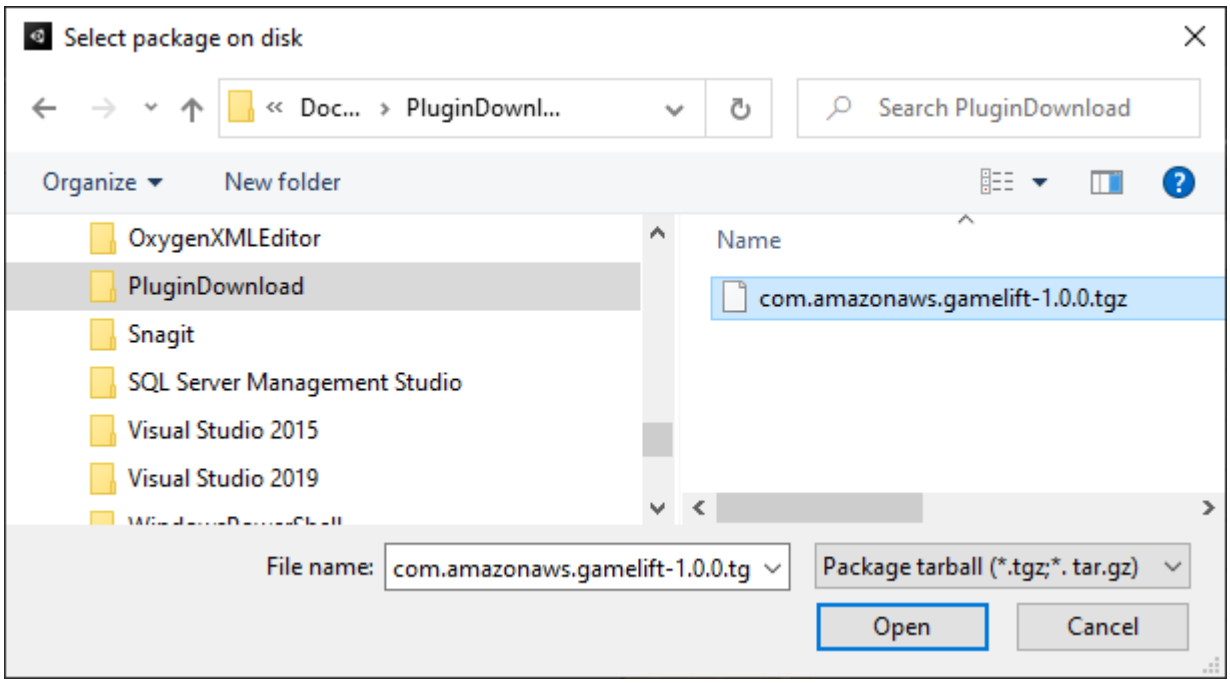
1. Download the Amazon GameLift plugin for Unity. You can find the latest version on the Amazon GameLift plugin for Unity repository page. Under the latest release, choose **Assets**, and then download the `com.amazonaws.gamelift-version.tgz` file.
2. Launch Unity and choose a project.
3. In the top navigation bar, under **Window** choose **Package Manager**:



4. Under the **Package Manager** tab choose **+**, and then choose **Add package from tarball...**:



5. In the **Select packages on disk** window, navigate to the `com.amazonaws.gamelift` folder, choose the file `com.amazonaws.gamelift-version.tgz` , and then choose **Open**:



6. After Unity has loaded the plug-in, **Amazon GameLift** appears as a new item in the Unity menu. It may take a few minutes to install and recompile scripts. The **Amazon GameLift Plugin Settings** tab automatically opens.

7. In the **SDK** pane, choose **Use .NET 4.x**.

   When configured, the status changes from **Not Configured** to **Configured**.

Test your game locally

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

Use Amazon GameLift Local to run Amazon GameLift on your local device. You can use Amazon GameLift Local to verify code changes in seconds, without a network connection.

Configure local testing

1. In the plugin for Unity window, choose the **Test** tab.
2. In the **Test** pane, choose **Download Amazon GameLift Local**. The plugin for Unity opens a browser window and downloads the `GameLift_06_03_2021.zip` file to your downloads folder.

   The download includes the C# Server SDK, .NET source files, and a .NET component compatible with Unity.
3. Unzip the downloaded file `GameLift_06_03_2021.zip`.
4. In the **Amazon GameLift Plugin Settings** window, choose **Amazon GameLift Local Path**, navigate to the unzipped folder, choose the file `GameLiftLocal.jar`, and then choose **Open**.

   When configured, local testing status changes from **Not Configured** to **Configured**.
5. Verify the status of the JRE. If the status is **Not Configured**, choose **Download JRE** and install the recommended Java version.

   After you install and configure the Java environment, the status changes to **Configured**.

Run your local game

1. In the plugin for Unity tab, choose the **Test** tab.
2. In the **Test** pane, choose **Open Local Test UI**.
3. In the **Local Testing** window, specify a **Server executable path**. Select **...** to select the path and executable name of your server application.
4. In the **Local Testing** window, specify a **GL Local port**.

5. Choose **Deploy & Run** to deploy and run the server.
6. To stop your game server, choose **Stop** or close the game server windows.

Deploy a scenario

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

A scenario uses an AWS CloudFormation template to create the resources you need to deploy a cloud hosting solution for your game. This section describes the scenarios Amazon GameLift provides and how to use them.

Prerequisites

To deploy the scenario, you need an IAM role for the Amazon GameLift service. For information on how to create a role for Amazon GameLift, see Set up an AWS account.

Each scenario requires permissions to the following resources:

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

Scenarios

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

The Amazon GameLift Plug-in for Unity includes the following scenarios:

Auth only

This scenario creates a game backend service that performs player authentication without game server capability. The template creates the following resources in your account:

- An Amazon Cognito user pool to store player authentication information.
- An Amazon API Gateway REST endpoint-backed AWS Lambda handler that starts games and views game connection information.

Single-Region fleet

This scenario creates a game backend service with a single Amazon GameLift fleet. It creates the following resources:

- An Amazon Cognito user pool for a player to authenticate and start a game.
- An AWS Lambda handler to search for an existing game session with an open player slot on the fleet. If it can't find a open slot, it creates a new game session.

Multi-Region fleet with a queue and custom matchmaker

This scenario forms matches by using Amazon GameLift queues and a custom matchmaker to group together the oldest players in the waiting pool. It creates the following resources:

- An Amazon Simple Notification Service topic that Amazon GameLift publishes messages to. For more information on SNS topics and notifications, see Set up event notification for game session placement.
- A Lambda function that's invoked by the message that communicates placement and game connection details.
- An Amazon DynamoDB table to store placement and game connection details. `GetGameConnection` calls read from this table and return the connection information to the game client.

Spot fleets with a queue and custom matchmaker

This scenario forms matches by using Amazon GameLift queues and a custom matchmaker and configures three fleets. It creates the following resources:

- Two Spot fleets that contain different instance types to provide durability for Spot unavailability.
- An On-Demand fleet that acts as a backup for the other Spot fleets. For more information on designing your fleets, see Amazon GameLift fleet design guide.
- A Amazon GameLift queue to keep server availability high and cost low. For more information and best practices about queues, see Design a game session queue.

FlexMatch

This scenario uses FlexMatch, a managed matchmaking service, to match game players together. For more information about FlexMatch, see What is Amazon GameLift FlexMatch. This scenario creates the following resources:

- A Lambda function to create a matchmaking ticket after it receives `StartGame` requests.
- A separate Lambda function to listen to FlexMatch match events.

To avoid unnecessary charges on your AWS account, remove the resources created by each scenario after you are done using them. Delete the corresponding AWS CloudFormation stack.

Update AWS credentials

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

The Amazon GameLift plugin for Unity requires security credentials to deploy a scenario. You can either create new credentials or use existing credentials.

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

To update AWS credentials

1. In Unity, in the plugin for Unity tab, choose the **Deploy** tab.
2. In the **Deploy** pane, choose **AWS Credentials**.
3. You can create new AWS credentials or choose existing credentials.

- To create credentials, choose **Create new credentials profile**, and then specify the **New Profile Name**, **AWS Access Key ID**, **AWS Secret Key**, and **AWS Region**.
- To choose existing credentials, choose **Choose existing credentials profile** and then select a profile name and **AWS Region**.

4. In the **Update AWS Credentials** window, choose **Update Credentials Profile**.

Update account bootstrap

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

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

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

To update the account bootstrap location

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

Deploy a game scenario

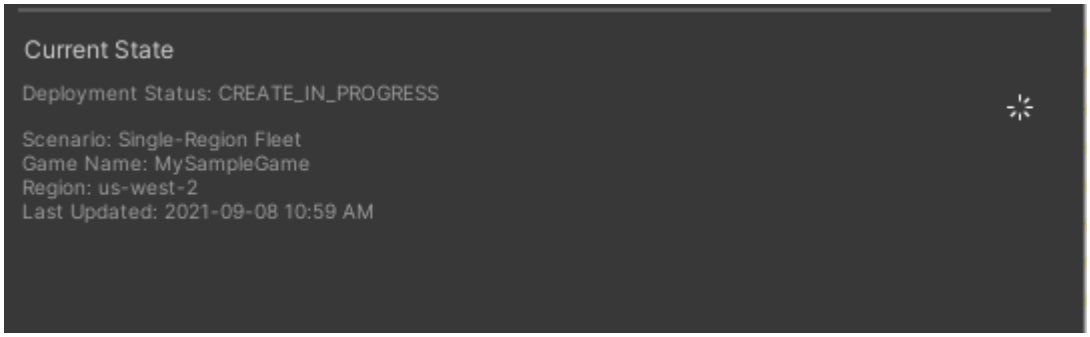
> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

You can use a scenario to test your game with Amazon GameLift. Each scenario uses a AWS CloudFormation template to create a stack with the required resources. Most of the scenarios require a game server executable and build path. When you deploy the scenario, Amazon GameLift copies game assets to the bootstrap location as part of deployment.

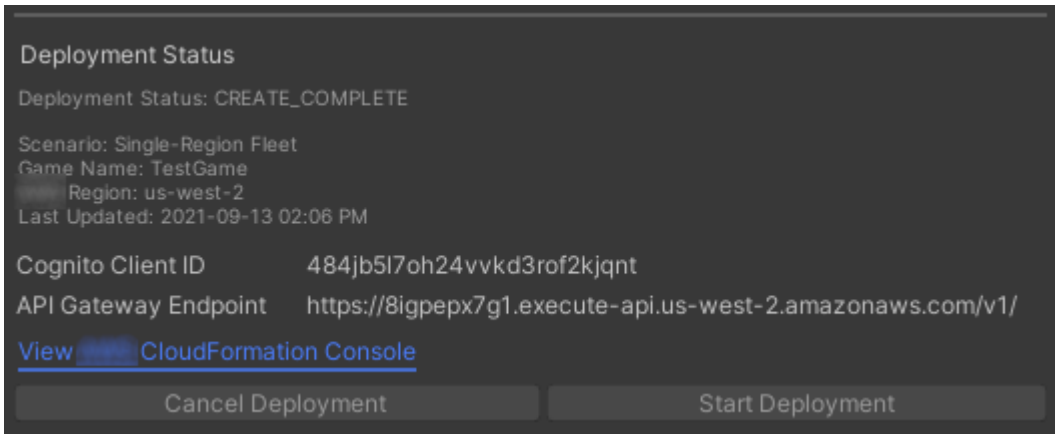
You must configure AWS credentials and an AWS account bootstrap to deploy a scenario.

To deploy a scenario

1. In Unity, in the plugin for Unity tab, choose the **Deploy** tab.
2. In the **Deploy** pane, choose **Open Deployment UI**.
3. In the **Deployment** window, choose a scenario.
4. Enter a **Game Name**. It must be unique. The game name is part of the AWS CloudFormation stack name when you deploy the scenario.
5. Choose the **Game Server Build Folder Path**. The build folder path points to the folder containing the server executable and dependencies.
6. Choose the **Game Server Build .exe File Path**. The build executable file path points to the game server executable.
7. Choose **Start Deployment** to begin deploying a scenario. You can follow the status of the update in the **Deployment** window under **Current State**.Scenarios can take up to 30 minutes to deploy.



8. When the scenario completes deployment, the **Current State** updates to include the **Cognito Client ID** and **API Gateway Endpoint** that you can copy and paste into the game.



9. To update game settings, on the Unity menu, choose **Go To Client Connection Settings**. This displays an **Inspector** tab on the right side of the Unity screen.
10. Deselect **Local Testing Mode**.
11. Enter the **API Gateway Endpoint** and the **Coginito Client ID**. Choose the same AWS Region you used for the scenario deployment. You can then rebuild and run the game client using the deployed scenario resources.

Deleting resources created by the scenario

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

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

To delete resources created by the scenario

1. In the Amazon GameLift plugin for Unity **Deployment** window, select **View AWS CloudFormation Console** to open the AWS CloudFormation console.
2. In the AWS CloudFormation console, choose **Stacks**, and then choose the stack that includes the game name specified during deployment.

3. Choose **Delete** to delete the stack. It may take a few minutes to delete a stack. After AWS CloudFormation deletes the stack used by the scenario, its status changes to `ROLLBACK_COMPLETE`.

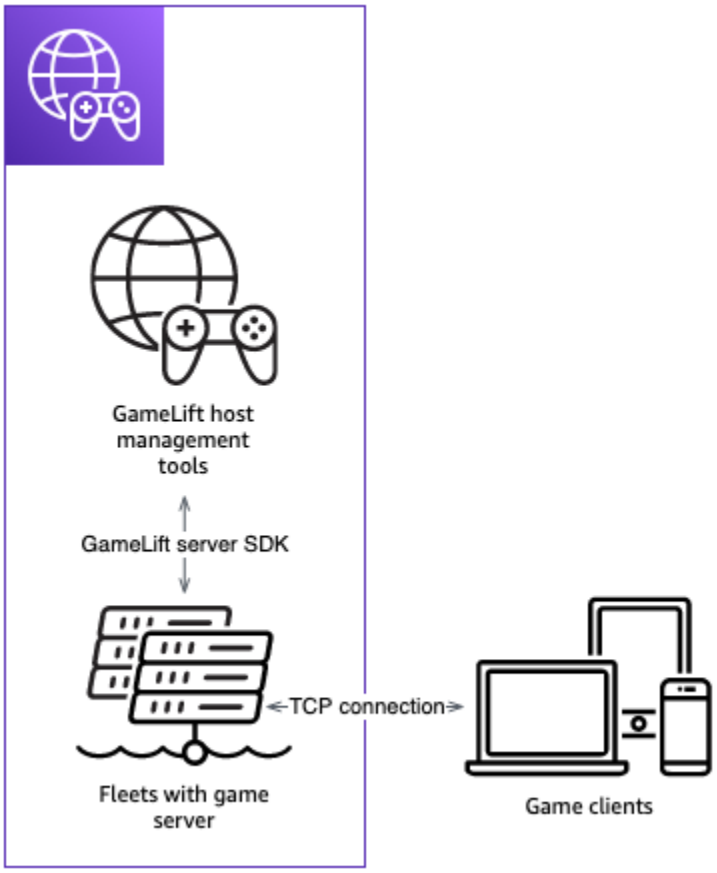
Integrate games with Amazon GameLift in Unity

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

Integrate your Unity game with Amazon GameLift by completing the following tasks:

- Integrate Amazon GameLift with a Unity game server project
- Integrate Amazon GameLift with a Unity game client project

The following diagram shows an example flow of integrating a game. In the diagram, a fleet with the game server is deployed to Amazon GameLift. The game client communicates with the game server, which communicates with Amazon GameLift.



Import and run a sample game

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

The Amazon GameLift plugin 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 client and game server and then test locally using Amazon GameLift Local.

Prerequisites

- Set up an AWS account
- Install and set up the plugin

Build and run the sample game server

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

Set up the game server files of the sample game.

1. In Unity, on the menu, choose **Amazon GameLift**, and then choose **Import Sample Game**.
2. In the **Import Sample Game** window, choose **Import** to import the game, its assets and dependencies.
3. Build the game server. In Unity, on the menu, choose **Amazon GameLift**, and then choose **Apply Windows Sample Server Build Settings** or **Apply MacOS Sample Server Build Settings**. After you configure the game server settings, Unity recompiles the assets.
4. In Unity, on the menu, choose **File**, and then choose **Build**. Choose **Server Build**, choose **Build**, and then choose a build folder specifically for server files.

   Unity builds the sample game server, placing the executable and required assets in the specified build folder.

Build and run the sample game client

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

Set up the game client files of the sample game.

1. In Unity, on the menu, choose **Amazon GameLift**, and then choose **Apply Windows Sample Client Build Settings** or **Apply MacOS Sample Client Build Settings**. After the game client settings are configured, Unity will recompile assets.
2. In Unity, on the menu, select **Go To Client Settings**. This will display an **Inspector** tab on the right side of the Unity screen. In the **Amazon GameLift Client Settings** tab, select **Local Testing Mode**.

3. Build the game client. In Unity, on the menu, choose **File**. Confirm **Server Build** is not checked, choose **Build**, and then select a build folder specifically for client files.

   Unity builds the sample game client, placing the executable and required assets in the specified client build folder.
4. You've no built the game server and client. In the next steps, you run the game and see how it interacts with Amazon GameLift.

Test the sample game locally

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

Run the sample game you imported using Amazon GameLift Local.

1. Launch the game server. In Unity, in the plugin for Unity tab, choose the **Deploy** tab.
2. In the **Test** pane, choose **Open Local Test UI**.
3. In the **Local Testing** window, specify a **Game Server .exe File Path**. The path must include the executable name. For example, `C:/MyGame/GameServer/MyGameServer.exe`.
4. Choose **Deploy and Run**. The plugin for Unity launches the game server and opens a Amazon GameLift Local log window. The windows contains log messages including messages sent between the game server and Amazon GameLift Local.
5. Launch the game client. Find the build location with the sample game client and choose the executable file .
6. In the **Amazon GameLift Sample Game**, provide an email and password and then choose **Log In**. The email and password aren't validated or used.
7. In the **Amazon GameLift Sample Game**, choose **Start**. The game client looks for a game session. If it can't find a session, it creates one. The game client then starts the game session. You can see game activity in the logs.

Sample game server logs

```
...
```

```
2021-09-15T19:55:3495 PID:20728 Log :) GAMELIFT AWAKE
2021-09-15T19:55:3512 PID:20728 Log :) I AM SERVER
2021-09-15T19:55:3514 PID:20728 Log :) GAMELIFT StartServer at port 33430.
2021-09-15T19:55:3514 PID:20728 Log :) SDK VERSION: 4.0.2
2021-09-15T19:55:3556 PID:20728 Log :) SERVER IS IN A GAMELIFT FLEET
2021-09-15T19:55:3577 PID:20728 Log :) PROCESSREADY SUCCESS.
2021-09-15T19:55:3577 PID:20728 Log :) GAMELIFT HEALTH CHECK REQUESTED (HEALTHY)
...
2021-09-15T19:55:3634 PID:20728 Log :) GAMELOGIC AWAKE
2021-09-15T19:55:3635 PID:20728 Log :) GAMELOGIC START
2021-09-15T19:55:3636 PID:20728 Log :) LISTENING ON PORT 33430
2021-09-15T19:55:3636 PID:20728 Log SERVER: Frame: 0 HELLO WORLD!
...
2021-09-15T19:56:2464 PID:20728 Log :) GAMELIFT SESSION REQUESTED
2021-09-15T19:56:2468 PID:20728 Log :) GAME SESSION ACTIVATED
2021-09-15T19:56:3578 PID:20728 Log :) GAMELIFT HEALTH CHECK REQUESTED (HEALTHY)
2021-09-15T19:57:3584 PID:20728 Log :) GAMELIFT HEALTH CHECK REQUESTED (HEALTHY)
2021-09-15T19:58:0334 PID:20728 Log SERVER: Frame: 8695 Connection accepted: playerIdx
 0 joined
2021-09-15T19:58:0335 PID:20728 Log SERVER: Frame: 8696 Connection accepted: playerIdx
 1 joined
2021-09-15T19:58:0338 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from playerIdx 0 Msg:
 CONNECT: server IP localhost
2021-09-15T19:58:0338 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from player 0:CONNECT:
 server IP localhost
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 CONNECT: player index 0
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from playerIdx 1 Msg:
 CONNECT: server IP localhost
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 Msg rcvd from player 1:CONNECT:
 server IP localhost
2021-09-15T19:58:0339 PID:20728 Log SERVER: Frame: 8697 CONNECT: player index 1
```

Sample Amazon GameLift Local logs

```
12:55:26,000  INFO || - [SocketIOServer] main - Session store / pubsub factory used:
 MemoryStoreFactory (local session store only)
12:55:28,092  WARN || - [ServerBootstrap] main - Unknown channel option 'SO_LINGER' for
 channel '[id: 0xe23d0a14]'
12:55:28,101  INFO || - [SocketIOServer] nioEventLoopGroup-2-1 - SocketIO server
 started at port: 5757
12:55:28,101  INFO || - [SDKConnection] main - GameLift SDK server (communicates with
 your game server) has started on http://localhost:5757
```

```
12:55:28,120  INFO || - [SdkWebSocketServer] WebSocketSelector-20 - WebSocket Server
 started on address localhost/127.0.0.1:5759
12:55:28,166  INFO || - [StandAloneServer] main - GameLift Client server (listens for
 GameLift client APIs) has started on http://localhost:8080
12:55:28,179  INFO || - [StandAloneServer] main - GameLift server sdk http listener has
 started on http://localhost:5758
12:55:35,453  INFO || - [SdkWebSocketServer] WebSocketWorker-12 - onOpen
 socket: /?pID=20728&sdkVersion=4.0.2&sdkLanguage=CSharp and handshake /?
pID=20728&sdkVersion=4.0.2&sdkLanguage=CSharp
12:55:35,551  INFO || - [HostProcessManager] WebSocketWorker-12 - client connected with
 pID 20728
12:55:35,718  INFO || - [GameLiftSdkHttpHandler] GameLiftSdkHttpHandler-thread-0 -
 GameLift API to use: ProcessReady for pId 20728
12:55:35,718  INFO || - [ProcessReadyHandler] GameLiftSdkHttpHandler-thread-0 -
 Received API call for processReady from 20728
12:55:35,738  INFO || - [ProcessReadyHandler] GameLiftSdkHttpHandler-thread-0 -
 onProcessReady data: port: 33430
 12:55:35,739  INFO || - [HostProcessManager] GameLiftSdkHttpHandler-thread-0 -
 Registered new process with pId 20728
12:55:35,789  INFO || - [GameLiftSdkHttpHandler] GameLiftSdkHttpHandler-thread-0 -
 GameLift API to use: ReportHealth for pId 20728
12:55:35,790  INFO || - [ReportHealthHandler] GameLiftSdkHttpHandler-thread-0 -
 Received API call for ReportHealth from 20728
12:55:35,794  INFO || - [ReportHealthHandler] GameLiftSdkHttpHandler-thread-0 -
 ReportHealth data: healthStatus: true
 12:56:24,098  INFO || - [GameLiftHttpHandler] Thread-12 - API to use:
 GameLift.DescribeGameSessions
12:56:24,119  INFO || - [DescribeGameSessionsDispatcher] Thread-12 - Received API call
 to describe game sessions with input: {"FleetId":"fleet-123"}
12:56:24,241  INFO || - [GameLiftHttpHandler] Thread-12 - API to use:
 GameLift.CreateGameSession
12:56:24,242  INFO || - [CreateGameSessionDispatcher] Thread-12 - Received API call to
 create game session with input: {"FleetId":"fleet-123","MaximumPlayerSessionCount":4}
12:56:24,265  INFO || - [HostProcessManager] Thread-12 - Reserved process:
 20728 for gameSession: arn:aws:gamelift:local::gamesession/fleet-123/
gsess-59f6cc44-4361-42f5-95b5-fdb5825c0f3d
12:56:24,266  INFO || - [WebSocketInvoker] Thread-12 - StartGameSessionRequest:
 gameSessionId=arn:aws:gamelift:local::gamesession/fleet-123/
gsess-59f6cc44-4361-42f5-95b5-fdb5825c0f3d, fleetId=fleet-123, gameSessionName=null,
 maxPlayers=4, properties=[], ipAddress=127.0.0.1, port=33430, gameSessionData?=false,
 matchmakerData?=false, dnsName=localhost
12:56:24,564  INFO || - [CreateGameSessionDispatcher] Thread-12 - GameSession with
 id: arn:aws:gamelift:local::gamesession/fleet-123/gsess-59f6cc44-4361-42f5-95b5-
fdb5825c0f3d created
```

```
12:56:24,585  INFO || - [GameLiftHttpHandler] Thread-12 - API to use:
 GameLift.DescribeGameSessions
12:56:24,585  INFO || - [DescribeGameSessionsDispatcher] Thread-12 - Received API call
 to describe game sessions with input: {"FleetId":"fleet-123"}
12:56:24,660  INFO || - [GameLiftSdkHttpHandler] GameLiftSdkHttpHandler-thread-0 -
 GameLift API to use: GameSessionActivate for pId 20728
12:56:24,661  INFO || - [GameSessionActivateHandler] GameLiftSdkHttpHandler-thread-0 -
 Received API call for GameSessionActivate from 20728
12:56:24,678  INFO || - [GameSessionActivateHandler] GameLiftSdkHttpHandler-thread-0
 - GameSessionActivate data: gameSessionId: "arn:aws:gamelift:local::gamesession/
fleet-123/gsess-59f6cc44-4361-42f5-95b5-fdb5825c0f3d"
```

Shut down server process

> **Note**
>
> This topic refers to Amazon GameLift plugin for Unity version 1.0.0, which uses server SDK 4.x or earlier.

After you're done with your sample game, shut down the server in Unity.

1. In the game client, choose **Quit** or close the window to stop the game client.
2. In Unity, in the **Local Testing** window, choose **Stop** or close the game server windows to stop the server.

Integrating games with the Amazon GameLift plugin for Unreal Engine

The topics in this section describe the Amazon GameLift plugin for Unreal Engine (UE) and how to use it to prepare your multiplayer game project for hosting with Amazon GameLift. Work entirely in your UE development environment with the plugin's guided workflows to complete the basic requirements for hosting with Amazon GameLift.

Amazon GameLift is a fully managed service that lets game developers manage and scale dedicated game servers for session-based multiplayer games. For more information about Amazon GameLift hosting, see How Amazon GameLift works.

Topics

About the plugin

The plugin adds Amazon GameLift tools and functionality to the UE editor. The plugin's guided workflows to integrate Amazon GameLift into your game project, designate a workstation as a local host for testing, and deploy the game server to Amazon GameLift cloud hosting.

Use the plugin's pre-built hosting solutions to deploy your game. Set up an Amazon GameLift Anywhere fleet with your local workstation as a host. For cloud hosting, choose from two common deployment scenarios that balance player latency, game session availability, and cost in different ways. One scenario includes a simple FlexMatch matchmaker and rule set. Use these solutions to get started quickly with a production-ready hosting structure in place, and then optimize and customize as needed.

The plugin includes these components:

- Plugin modules for the UE editor. When the plugin is installed, a new main menu button gives you access to Amazon GameLift functionality.
- C++ libraries for the Amazon GameLift service API with client-side functionality.
- Unreal libraries for the Amazon GameLift server SDK (version 5).
- Content for testing, including a startup game map and two testing maps with basic blueprints and UI elements for use with testing a server integration.
- Editable configurations, in the form of AWS CloudFormation templates, that the plugin uses when deploying your game server for hosting.

Plugin workflow

The following steps describe a typical approach to integrating and deploying a game project with the Amazon GameLift plugin for Unreal Engine. You complete these steps by working in the UE editor and your game code.

1. Create a user profile that links to your AWS account and provides access credentials for valid account user with permissions to use Amazon GameLift.
2. Add server code to your game project to establish communication between a running game server and the with Amazon GameLift service.
3. Add client code to your game project that lets game clients send requests to Amazon GameLift to start new game sessions and then connect to them.
4. Use the Anywhere workflow to set up your local workstation as an Anywhere host for your game server. Launch your game server and client locally through the plugin, connect to a game session, and test your integration.
5. Use the EC2 hosting workflow to upload your integrated game server and deploy a cloud hosting solution, When your game server is ready, launch your game client locally through the plugin, connect to a game session and play the game.

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

Install the plugin for Unreal

This section describes the initial installation tasks to add the plugin to an Unreal Engine project. The plugin functionality is available when you have the project open in the Unreal editor.

> **Note**
>
> You can use the Amazon GameLift plugin with a standard version of the UE editor, but you need to use a source-built version when you package your game server build.

Before you start

Here's what you need to use the Amazon GameLift plugin for Unreal Engine:

- Amazon GameLift plugin for Unreal Engine release package. [Download site].
- Microsoft Visual Studio 2019 or newer.
- A source-built version of the Unreal Engine editor. You need a source-built version to package the server components for a multiplayer game. For more details, including additional prerequisites, see the Unreal Engine documentation:

- Accessing Unreal Engine source code on GitHub You'll need GitHub and Epic Games accounts.
- Building Unreal Engine from Source tutorial.

- A multiplayer game project with C++ game code. If you're working with a Blueprint project, see Unreal documentation on how to generate C++ source code for your project.

Add the plugin to your game project

Complete the following tasks to add the plugin to your game project.

Build the Amazon GameLift C++ server SDK

1. Unzip the Amazon GameLift plugin for Unreal Engine release package to extract two zip files:

   - `amazon-gamelift-plugin-unreal-<>-sdk-<>.zip`
   - `GameLift-Cpp-ServerSDK-<>.zip.`

   Unzip these files.
2. Open the `GameLift-Cpp-ServerSDK-<>` folder, and then complete the following instructions for your platform: Linux or Microsoft Windows.

Linux

1. Run the following commands:

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

   These commands build the `/lib/aws-cpp-sdk-gamelift-server.so` file.
2. Copy `/lib/aws-cpp-sdk-gamelift-server.so` to the `amazon-gamelift-plugin-unreal/GameLiftPlugin/Source/GameliftServer/ThirdParty/GameLiftServerSDK/Linux/x86_64-unknown-linux-gnu/` directory.

Microsoft Windows

1. Run the following commands:

```
mkdir out
cd out
cmake -G "Visual Studio 17 2022" -DBUILD_FOR_UNREAL=1 ..
msbuild ALL_BUILD.vcxproj /p:Configuration=Release
```

   These commands build the following binary files.

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

2. Copy the files to the `amazon-gamelift-plugin-unreal\GameLiftPlugin\Source\GameliftServer\ThirdParty\GameLiftServerSDK\Win64\` directory.

Complete the following tasks, working in your game project files.

1. **Install the plugin files.**

   a. Locate your game project root folder, such as `... > Unreal Projects/[project-name]/`. If the Plugins folder doesn't exist there, then create it.

   b. Go to the `amazon-gamelift-plugin-unreal` folder unzipped from `amazon-gamelift-plugin-unreal-<>-sdk-<>.zip`. Copy the `GameLiftPlugin` folder from the `gamelift-plugin-unreal` folder to the `Plugins` folder in the game project directory.

2. **Add the plugin to the `.uproject` file.**

   a. In your game project root folder, open the `.uproject` file.

   b. Update the file to add "GameLiftPlugin" and "WebBrowserWidget" to the `Plugins` section and enable them. The following code shows the updated `.uproject` file for a game called "MyGame".

```
UnrealProjects > MyGame > MyGame.uproject
{
  ...
  "Plugins": [
```

```
    {
      "Name": "ModelingToolsEditorMode",
      "Enabled": true,
      "TargetAllowList": [ "Editor" ]
    },
    {
      "Name": "GameLiftPlugin",
      "Enabled": true
    },
    {
      "Name": "WebBrowserWidget",
      "Enabled": true
    }
  ]
}
```

3. **Change the UE editor version for your project.**

   If you created a project for one editor version and now want to change to another version (such as a source-build version), you need to update the project.

   In your game project root folder, select the `.uproject` file and choose the option **Switch Unreal Engine Version**. Select a new editor version.

4. **Rebuild the project solution with your updates.**

   a. In the project root folder, look for a solution (`*.sln`) file. If none exists, select the `.uproject` file and choose the option **Generate Visual Studio project files**.

   b. Open the solution file and build or rebuild the project.

5. **Verify that the plugin is enabled in the UE editor.**

   > **Note**
   >
   > If you If you already have the editor open, you might need to restart the editor before it recognizes the new plugin.

   a. Open the project in your chosen UE editor.

   b. Check the main editor toolbar for the new Amazon GameLift menu button [need image].

   c. Look in the **Content Browser** for the Amazon GameLift plugin assets. Make sure that your **View Options** setting has the **Show Plugin Content** option selected.

Set up an AWS user profile

After installing the plugin, set up a profile and link it to a valid AWS account user. You can maintain multiple profiles, but you can only have one profile active at a time. Whenever you work in the plugin, select a profile to use.

Maintaining multiple profiles gives you the ability to switch between different hosting scenarios. For example, you might set up profiles with the same AWS credentials but different AWS Regions. Or you might set up profiles with different AWS accounts or with different users/permission sets.

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

To manage your AWS profiles

1. In the Unreal editor main toolbar, choose the Amazon GameLift menu, and select **Set AWS User Profiles**. This action opens Project Settings for the plugin. Expand the section **AWS User Profiles**.
2. If the plugin doesn't detect an existing profile, it prompts you to create one. You can create a new profile using either a new or existing AWS account.

   > **Note**
   >
   > You need to use the AWS Management Console to create a new AWS account and create or update a user with the proper permission set.

   When setting up a profile, you need the following information:

   - An AWS account. If you need to create a new AWS account, follow the prompts to create the account. See Create an AWS account for more details.
   - An AWS user with permissions to use Amazon GameLift and other required AWS services. See Set up an AWS account for instructions on setting up an AWS Identity and Access

Management (IAM) user with Amazon GameLift permissions and programmatic access with long-term credentials.

- Credentials for your AWS user. These credentials consist of an AWS access key ID and AWS secret key. See Get your access keys for more details.
- AWS region. This is a geographic location where you want to create your AWS resources for hosting. During development, we recommend using a region close to your physical location. See the list of supported AWS regions.

3. If the plugin detects an existing profile, you aren't prompted to create one. If you want to update a profile or create a new one, choose Manage your profile.

To bootstrap your profile:

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

Bootstrapping involves creating new AWS resources and might incur costs.

1. In the Unreal editor main toolbar, choose the Amazon GameLift icon, and select **Set AWS User Profiles**. This action opens Project Settings for the plugin. Expand the section **AWS User Profiles**.
2. In the **Bootstrap your profile** section, select a profile from the dropdown list and check the bootstrap status. If the status indicates that no bucket exists, choose the button **Bootstrap and create profile** to create an Amazon S3 bucket for the selected profile.
3. Wait for bootstrap status to change to "Active". This can take a few minutes.

Set up your game for testing with Amazon GameLift Anywhere

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

To start the Amazon GameLift Anywhere workflow:

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

Step 1: Set your profile

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

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

Step 2: Set up your game code

In this step, you make a series of updates to your client and server code to add hosting functionality. If you haven't already set up a source-built version of the Unreal editor, the plugin provides links to instructions and source code.

With the plugin, can take advantage of some conveniences when integrating your game code. You can do a minimal integration to set up basic hosting functionality. You can also do a more extensive custom integration. The information in this section describes the minimal integration option. Use the test maps included with the plugin to add client Amazon GameLift functionality to your game project. For server integration, use the provided code sample to update your project's game mode.

Integrate your server game mode

Add server code to your game that enables communication between your game server and the Amazon GameLift service. Your game server must be able to respond to requests from Amazon GameLift, such as to start a new game session, and also report status on game server health and player connections.

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

we recommend that you make a backup copy of the original game mode files for use with your client.

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

#pragma once

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

struct FProcessParameters;

DECLARE_LOG_CATEGORY_EXTERN(GameServerLog, Log, All);

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

public:
    AGameLiftServerGameMode();

protected:
    virtual void BeginPlay() override;

private:
    void InitGameLift();

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

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

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

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

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

#include "GameLiftServerGameMode.h"

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

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

#include "GenericPlatform/GenericPlatformOutputDevices.h"

DEFINE_LOG_CATEGORY(GameServerLog);

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

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

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

```
void AGameLiftServerGameMode::BeginPlay()
{
    Super::BeginPlay();

#if WITH_GAMELIFT
    InitGameLift();
#endif
}

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

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

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

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

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

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

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

        FString glAnywhereProcessId = "";
        if (FParse::Value(FCommandLine::Get(), TEXT("glAnywhereProcessId="),
glAnywhereProcessId))
        {
            ServerParametersForAnywhere.m_processId =
TCHAR_TO_UTF8(*glAnywhereProcessId);
        }
        else
        {
            // If no ProcessId is passed as a command line argument, generate a
randomized unique string.
            ServerParametersForAnywhere.m_processId =
                TCHAR_TO_UTF8(
                    *FText::Format(
                        FText::FromString("ProcessId_{0}"),
                        FText::AsNumber(std::time(nullptr))
                    ).ToString()
                );
        }

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

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

```
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_YELLOW);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Web Socket URL: %s"),
 *ServerParametersForAnywhere.m_webSocketUrl);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Fleet ID: %s"),
 *ServerParametersForAnywhere.m_fleetId);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Process ID: %s"),
 *ServerParametersForAnywhere.m_processId);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Host ID (Compute Name): %s"),
 *ServerParametersForAnywhere.m_hostId);
        UE_LOG(GameServerLog, Log, TEXT(">>>> Auth Token: %s"),
 *ServerParametersForAnywhere.m_authToken);
        UE_LOG(GameServerLog, SetColor, TEXT("%s"), COLOR_NONE);
    }

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

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

    ProcessParameters = MakeShared<FProcessParameters>();

    //When a game session is created, GameLift sends an activation request to the
 game server and passes along the game session object containing game properties
 and other settings.
    //Here is where a game server should take action based on the game session
 object.
```

```
    //Once the game server is ready to receive incoming player connections, it
 should invoke GameLiftServerAPI.ActivateGameSession()
    ProcessParameters->OnStartGameSession.BindLambda([=]
(Aws::GameLift::Server::Model::GameSession InGameSession)
    {
        FString GameSessionId = FString(InGameSession.GetGameSessionId());
        UE_LOG(GameServerLog, Log, TEXT("GameSession Initializing: %s"),
 *GameSessionId);
        GameLiftSdkModule->ActivateGameSession();
    });

    //OnProcessTerminate callback. GameLift will invoke this callback before
 shutting down an instance hosting this game server.
    //It gives this game server a chance to save its state, communicate with
 services, etc., before being shut down.
    //In this case, we simply tell GameLift we are indeed going to shutdown.
    ProcessParameters->OnTerminate.BindLambda([=]()
    {
        UE_LOG(GameServerLog, Log, TEXT("Game Server Process is terminating"));
        GameLiftSdkModule->ProcessEnding();
    });

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

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

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

    for (FString SwitchStr : CommandLineSwitches)
```

```
    {
        FString Key;
        FString Value;

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

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

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

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

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

```
}
```

Integrate your client game map

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

The startup map has the following characteristics:

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

To use the startup game map

1. In the UE editor, open the **Project Settings, Maps & Modes** page, and expand the **Default Maps** section.
2. For **Editor Startup Map**, select "StartupMap" from the dropdown list. You might need to search for the file, which is located in `... > Unreal Projects/[project-name]/Plugins/Amazon GameLift Plugin Content/Maps`.
3. For **Game Default Map**, select the same "StartupMap" from the dropdown list.
4. For **Server Default Map**, select "ThirdPersonMap". This is a default map included in your game project. This map is designed for two players in the game.
5. Open the details panel for the server default map. Set **GameMode Override** to "None".
6. Expand the **Default Modes** section, and set **Global Default Server Game Mode** to the game mode you updated for your server integration.

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

Build your game components

1. Create new server and client target files

   a. In your game project folder, go to the Source folder and find the `Target.cs` files.

   b. Copy the file `[project-name]Editor.Target.cs` to two new files named `[project-name]Client.Target.cs` and `[project-name]Server.Target.cs`.

   c. Edit each of the new files to update the class name and target type values, as shown:

```
UnrealProjects > MyGame > Source > MyGameClient.Target.cs
// Copyright Epic Games, Inc. All Rights Reserved.

using UnrealBuildTool;
using System.Collections.Generic;

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

```
UnrealProjects > MyGame > Source > MyGameServer.Target.cs
// Copyright Epic Games, Inc. All Rights Reserved.

using UnrealBuildTool;
using System.Collections.Generic;

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

```
}
```

2. Update the `.Build.cs` file.

   a. Open the `.Build.cs` file for your project. This file is located in `UnrealProjects/[project name]/Source/[project name]/[project name].Build.cs`.

   b. Update the `ModuleRules` class as shown in the following code sample.

```
public class MyGame : ModuleRules
{
  public GameLiftUnrealApp(TargetInfo Target)
  {
    PublicDependencyModuleNames.AddRange(new string[] { "Core", "CoreUObject",
 "Engine", "InputCore" });
    bEnableExceptions = true;

    if (Target.Type == TargetRules.TargetType.Server)
    {
      PublicDependencyModuleNames.AddRange(new string[]
 { "GameLiftServerSDK" });
      PublicDefinitions.Add("WITH_GAMELIFT=1");
    }
    else
    {
      PublicDefinitions.Add("WITH_GAMELIFT=0");
    }
  }
}
```

3. Rebuild your game project solution.
4. Open your game project in a source-built version of the Unreal Engine editor.
5. Do the following for both your client and server:

   a. Choose a target. Go to **Platforms, Windows** and select one of the following:

      - Server: `[your-application-name]Server`
      - Client: `[your-application-name]Client`

   b. Start the build. Go to **Platform, Windows, Package Project**.

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

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

Step 3: Connect to an Anywhere fleet

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

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

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

Step 4: Register your workstation

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

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

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

Step 5: Generate auth token

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

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

Step 6: Launch game

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

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

Deploy your game to cloud hosting with managed EC2 fleets

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

To start the Amazon GameLift managed Amazon EC2 workflow:

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

Step 1: Set your profile

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

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

Step 2: Set up your game code

In this step, you make a series of updates to your client and server code to add hosting functionality. If you haven't already set up a source-built version of the Unreal editor, the plugin provides links to instructions and source code.

If you integrated your game for use with an Anywhere fleet, you don't need to make any changes to your game code. If you're using the startup game map, this works with EC2 deployments also.

- Set up your game code (Anywhere)
- Build your game components

After building your game server, complete the following tasks to prepare it for uploading to Amazon GameLift.

To package your server build for cloud deployment

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

1. Copy the install script, included in the plugin download, into the root of the `WindowsServer` folder. Look for the file `[project-name]/Plugins/Resources/CloudFormation/extra_server_resources/install.bat`. Amazon GameLift uses this file to install the server build on each EC2 hosting resource.

2. Copy the `VC_redist.x64.exe` file, included in your Visual Studio installation, into the root of the `WindowsServer` folder. This file is commonly located at `C:/Program Files (x86)/Microsoft Visual Studio/2019/Professional/VC/Redist/MSVC/v142`.
3. Copy the OpenSSL DLLs for your game server build into the folder `WindowsServer/MyGame/Binaries/Win64`. Make sure the DLLs are for same version used in the server build. Copy the following files:

   - `libssl-3-x64.dll`
   - `libcrypto-3-x64.dll`

Step 3: Select deployment scenario

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

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

- AWS game session placement queue that fulfills requests for proposed matches by finding the best possible hosting resource (based on viability, cost, player latency, etc.) and starting a game session.
- Amazon Cognito user pool and client to enable players to authenticate and start a game.
- API gateway authorizer that links user pool with APIs.
- WebACl for throttling excessive player calls to API gateway.
- API gateway + Lambda function for players to request a game slot. This function calls `StartMatchmaking()`.
- API gateway + Lambda function for players to get connection info for their game request.
- Amazon DynamoDB tables to store matchmaking tickets for players and game session information. .
- SNS topic + Lambda function to handle GameSessionQueue events.

Step 4: Set game parameters

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

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

Step 5: Deploy scenario

In this step, you deploy your game to a cloud hosting solution based on the deployment scenario you chose. This process can take as long as 40 minutes while AWS validates your server build, provisions hosting resources, installs your game server, launches server processes, and gets them ready to host game sessions.

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

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

Step 6: Launch client

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

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

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

Get fleet data for a Amazon GameLift instance

There are some situations where your custom game build or Realtime Servers script may require information about the Amazon GameLift fleet. For example, your game build or script might include code to:

- Monitor activity based on fleet data.
- Roll up metrics to track activity by fleet data. (Many games use this data for LiveOps activities.)
- Provide relevant data to custom game services, such as for matchmaking, additional capacity scaling, or testing.

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

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

The `gamelift-metadata.json` file includes the attributes of a Amazon GameLift fleet resource.

Example JSON file:

```
{
    "buildArn":"arn:aws:gamelift:us-west-2:123456789012:build/
build-1111aaaa-22bb-33cc-44dd-5555eeee66ff",
    "buildId":"build-1111aaaa-22bb-33cc-44dd-5555eeee66ff",
    "fleetArn":"arn:aws:gamelift:us-west-2:123456789012:fleet/
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
    "fleetDescription":"Test fleet for Really Fun Game v0.8",
    "fleetId":"fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
    "fleetName":"ReallyFunGameTestFleet08",
    "fleetType":"ON_DEMAND",
    "instanceRoleArn":"arn:aws:iam::123456789012:role/S3AccessForGameLift",
    "instanceType":"c5.large",
    "serverLaunchPath":"/local/game/reallyfungame.exe"
}
```

Adding FlexMatch matchmaking

Use Amazon GameLift FlexMatch to add player matchmaking functionality to your Amazon 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. Amazon GameLift uses queues to locate the best possible hosting locations for games across multiple regions and computing types. In particular, Amazon 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.

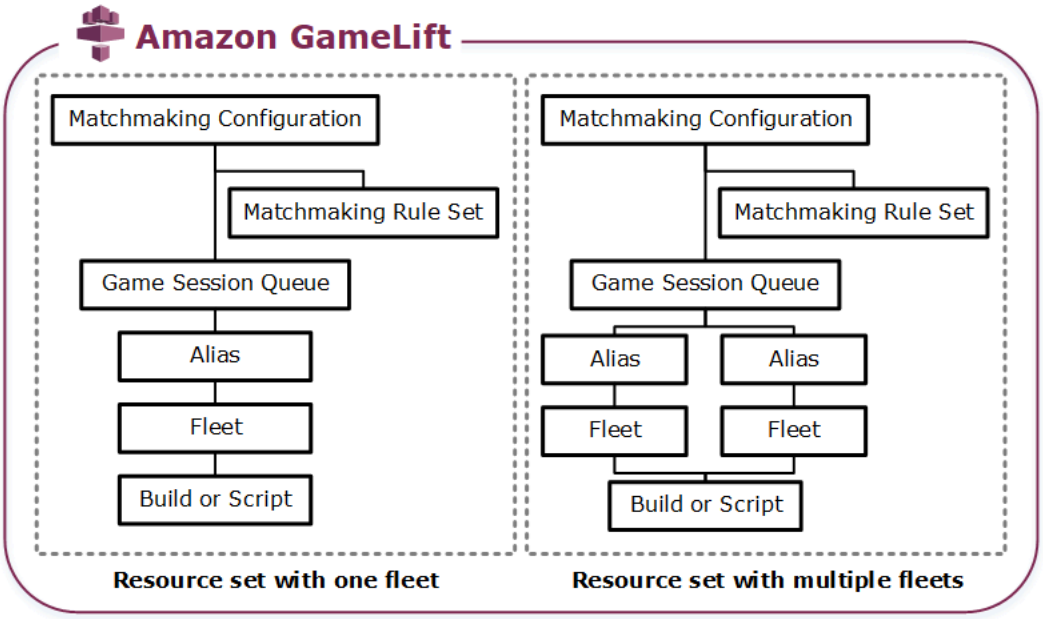
For more information on FlexMatch including detailed help with integrating matchmaking into your games, see these Amazon GameLift FlexMatch Developer Guide topics:

- How Amazon GameLift FlexMatch works
- FlexMatch integration steps

Managing Amazon GameLift hosting resources

This section provides detailed information about setting up Amazon GameLift managed resources to run your game servers and host game sessions for players. You must configure and deploy resources, scale capacity to meet player demand, and locate available resources to host game sessions.

The following diagram illustrates how Amazon GameLift resource objects relate to each other. Use a build or script to create a fleet, give a fleet an alias, and add fleets to a game session queue using their alias. For games that use FlexMatch matchmaking, use the game session queue and a matchmaking rule set to create a matchmaking configuration.



Game server code

- **Build** – Your custom-built game server software that runs on Amazon 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, and that you must integrate with Amazon GameLift. Upload game build files to Amazon GameLift in the AWS Regions where you plan to set up fleets. For more information, see Upload a custom server build to Amazon GameLift.
- **Script** – Your configuration and custom game logic for use with Realtime Servers. Configure Realtime Servers for your game clients by creating a script using JavaScript, and add custom game logic to host game sessions for your players. For more information, see Upload a Realtime Servers script to Amazon GameLift.

Fleet

A collection of compute resources that run your game servers and host game sessions for your players. For information about where you can deploy fleets, see Amazon GameLift hosting locations. For information about creating fleets, see Setting up Amazon GameLift fleets.

Alias

An abstract identifier for a fleet that you can use to change the fleet that your players are connected to at any time. For more information, see Add an alias to a Amazon GameLift fleet.

Game session queue

A game session placement mechanism that receives requests for new game sessions and searches for available game servers to host the new sessions. For more information about game session queues, see Setting up Amazon GameLift queues for game session placement.

Uploading builds and scripts to Amazon GameLift

Before deploying your multiplayer game servers for hosting with Amazon GameLift, 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.

Topics

- Upload a custom server build to Amazon GameLift
- Upload a Realtime Servers script to Amazon GameLift

Upload a custom server build to Amazon GameLift

After you integrate your game server with Amazon GameLift, upload the build files to Amazon GameLift. This topic covers how to package your game's build files, create an optional build install script, and then upload the files using the AWS Command Line Interface (AWS CLI) or an AWS SDK.

Topics

- Package your game build files
- Create a Amazon GameLift build
- Update your build files
- Add a build install script

Package your game build files

Before uploading your configured game server to Amazon GameLift, 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 built to run on the same platform. For a list of supported platforms, see Development support with Amazon GameLift.
- **Dependencies** – Any dependent files that your game server executables require to run. Examples include assets, configuration files, and dependent libraries.

> **Note**
>
> For game builds created with the Amazon GameLift server SDK for C++ (including those created with the Unreal plugin), include the OpenSSL DLL for the same version of OpenSSL that you built the server SDK with. See the server SDK README file for more details.

- **Install script** (Optional) – A script file to handle tasks that install your game build on Amazon GameLift hosting servers. Place this file at the root of the build directory. Amazon GameLift runs the install script as part of fleet creation.

You can set up any application in your build, including your install script, to access your resources securely on other AWS services. For information about ways to do this, see Communicate with other AWS resources from your fleets.

After you've packaged your build files, make sure that your game server can run on a clean installation of your target OS. This verifies that you include all required dependencies in your package and that your install script is accurate.

Create a Amazon GameLift build

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

- Create a build from a file directory. This is the simplest and most commonly used option.
- Create a build with files in Amazon Simple Storage Service (Amazon S3). With this option, you can manage your build versions in Amazon S3.

With both methods, Amazon GameLift creates a new build resource with a unique build ID and other metadata. The build starts in the **Initialized** status. After Amazon GameLift acquires the game server files, the build moves to **Ready** status.

When the build is ready, you can deploy it to a new Amazon GameLift fleet. For more information, see Create a Amazon GameLift managed fleet.When Amazon GameLift sets up the new fleet, it downloads the build files to each fleet instance and installs the build files.

Create a build from a file directory

To create a game build stored in any location, including a local directory, use the `upload-build` AWS CLI command. This command creates a new build record in Amazon GameLift and uploads files from a location that you specify.

Send an upload request. In a command line window, enter the following **upload-build** command and parameters.

```
aws gamelift upload-build \
    --name user-defined name of build \
    --operating-system supported OS \
    --server-sdk-version Amazon GameLift server SDK version \
    --build-root build path \
    --build-version user-defined build number \
    --region region name
```

- **operating-system** – The game server build's runtime environment. You must specify an OS value. You can't update this later.
- **server-sdk-version** – The version of the Amazon GameLift server SDK that your game server is integrated with. If you don't provide a value, Amazon GameLift uses the default value `4.0.2`. If you specify an incorrect server SDK version, the game server build might fail when calling `InitSdk` to establish a connection to the Amazon GameLift service.
- **build-root** – The directory path of your build files.
- **name** – A descriptive name for the new build.
- **build-version** – The version details for the build files.
- **region** – The AWS Region where you want to create your build. Create the build in the Region where you plan to deploy fleets. If you're deploying your game in multiple Regions, create a build in each Region.

> **Note**
>
> View your current default Region using the **aws configure get region**. To change your default Region, use the **aws configure set region *region name*** command.

Examples

```
aws gamelift upload-build \
    --operating-system AMAZON_LINUX_2023 \

    --server-sdk-version "5.0.0" \
    --build-root "~/mygame" \
    --name "My Game Nightly Build" \
    --build-version "build 255" \
    --region us-west-2
```

```
aws gamelift upload-build \
    --operating-system WINDOWS_2016 \
    --server-sdk-version "5.0.0" \
    --build-root "C:\mygame" \
    --name "My Game Nightly Build" \
    --build-version "build 255" \
    --region us-west-2
```

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

Create a build with files in Amazon S3

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

To create a build resource

1. **Store your build files in Amazon S3.** Create a .zip file containing the packaged build files and upload it to an S3 bucket in your AWS account. Take note of the bucket label and the file name, you'll need these when creating a Amazon GameLift build.

2. **Give Amazon GameLift access to your build files.** Create an IAM role by following the instructions in Access a game build file in Amazon S3. After you've created the role, take note of the new role's Amazon Resource Name (ARN), you'll need this when creating a build.
3. **Create a build.** Use the Amazon GameLift console or the AWS CLI to create a new build record. You must have the `PassRole` permission, as described in IAM permission examples for Amazon GameLift.

Console

1. In the Amazon GameLift console, in the navigation pane, choose **Hosting**, **Builds**.
2. On the **Builds** page, choose **Create build**.
3. On the **Create build** page, under **Build settings**, do the following:
   a. For **Name**, enter a script name.
   b. For **Version**, enter a version. Because you can update the content of a build, version data can help you track updates.
   c. For **Operating system (OS)**, choose the OS of your game server build. You can't update this value later.
   d. For **Game server build**, enter the **S3 URI** of the build object that you uploaded to Amazon S3, and choose the **Object version**. If you don't remember the Amazon S3 URI and object version, choose **Browse S3** and search for the build object.
   e. For **IAM role**, choose the role that you created that gives Amazon GameLift access to your S3 bucket and build object.
4. (Optional) Under **Tags**, add tags to the build by entering **Key** and **Value** pairs.
5. Choose **Create**.

Amazon GameLift assigns an ID to the new build and uploads the designated .zip file. You can view the new build, including the status, on the **Builds** page.

AWS CLI

To define the new build and upload your server build files, use the `create-build` command.

1. Open a command line window and switch to a directory where you can use the AWS CLI.
2. Enter the following **create-build** command:

```
aws gamelift create-build \
    --name user-defined name of build \
    --server-sdk-version Amazon GameLift server SDK version \
    --operating-system supported OS \
    --build-version user-defined build number \
    --storage-location "Bucket"=S3 bucket label,"Key"=Build .zip file
 name,"RoleArn"=Access role ARN} \
    --region region name
```

- **name** – A descriptive name for the new build.
- **server-sdk-version** – The version of the Amazon GameLift server SDK you used to integrate your game server with Amazon GameLift. If you don't provide a value, Amazon GameLift uses the default value `4.0.2`.
- **operating-system** – The game server build's runtime environment. You must specify an OS value. You can't update this later.
- **build-version** – The version details for the build files. This information can be useful because each new version of your game server requires a new build resource.
- **storage-location**
  - **Bucket** – The name of the S3 bucket that contains your build. For example, "my_build_files".
  - **Key** – The name of the .zip file that contains your build files. For example, "my_game_build_7.0.1, 7.0.2".
  - **RoleARN** – The ARN assigned to the IAM role that you created. For example, "arn:aws:iam::111122223333:role/GameLiftAccess". For an example policy, see Access a game build file in Amazon S3.
- **region** – Create the build in the AWS Region where you plan to deploy fleets. If you're deploying your game in multiple Regions, create a build in each Region.

> **Note**
>
> We recommend checking your current default Region using the `configure get` command. To change your default Region, use the `configure set` command.

Example

```
aws gamelift create-build \
    --operating-system WINDOWS_2016 \
    --storage-location
 "Bucket"="my_game_build_files","Key"="mygame_build_101.zip","RoleArn"="arn:aws:iam::111
gamelift" \
    --name "My Game Nightly Build" \
    --build-version "build 101" \
    --region us-west-2
```

3. To view the new build, use the `describe-build` command.

Update your build files

You can update the metadata for a build resource using the Amazon GameLift console or the `update-build` AWS CLI command.

After you've created a Amazon GameLift build, you can't update the build files associated with it. For each new set of files, create a new Amazon GameLift build. Using the `upload-build` command, Amazon GameLift automatically creates a new build record for each request. If you provide build files using the `create-build` command, upload a new build .zip file with a different name to Amazon S3 and create a build by referencing the new file name.

Try these tips for deploying updated builds:

- **Use queues and swap out fleets as needed.** When setting up your game client with Amazon GameLift, specify a queue instead of a fleet. With queues, you can add the new fleets with the new build to your queue and remove the old fleets. For more information, see Setting up Amazon GameLift queues for game session placement.
- **Use aliases to transfer players to a new game build.** When integrating your game client with Amazon GameLift, specify a fleet alias instead of a fleet ID. For more information, see Add an alias to a Amazon GameLift fleet.
- **Set up automated build updates.** For sample scripts and information about incorporating Amazon GameLift deployments into your build system, see Automating Deployments to Amazon GameLift on the AWS Game Tech Blog.

Add a build install script

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

- Windows: Create a batch file named `install.bat`.
- Linux: Create a shell script file named `install.sh`.

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

- The script can't take any user input.
- Amazon GameLift installs the build and recreates the file directories in your build package on a hosting server in the following locations:
  - Windows fleets: `C:\game`
  - Linux fleets: `/local/game`
- During the installation process for Linux fleets, the run-as user has limited access to the instance file structure. This user has full rights to the directory where your build files are installed. If your install script performs actions that require administrator permissions, then specify admin access using **sudo**. The run-as user for Windows fleets has administrator permissions by default. Permission failures related to the install script generate an event message that indicates a problem with the script.
- On Linux, Amazon GameLift supports common shell interpreter languages such as bash. Add a shebang (such as `#!/bin/bash`) to the top of your install script. To verify support for your preferred shell commands, remotely access an active Linux instance and open a shell prompt. For more information, see Remotely connect to Amazon GameLift fleet instances.
- The install script can't rely on a VPC peering connection. A VPC peering connection isn't available until after Amazon GameLift installs the build on fleet instances.

Example Windows install bash file

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

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

Example Linux install shell script

This example `install.sh` file uses bash in the install script and writes results to a log file.

```
#!/bin/bash
```

```
echo 'Hello World' > install.log
```

This example `install.sh` file shows how you can use the Amazon CloudWatch agent to collect system-level and custom metrics, and handle log rotation. Because Amazon GameLift runs in a service VPC, you must grant Amazon GameLift permissions to assume an AWS Identity and Access Management (IAM) role on your behalf. To allow Amazon GameLift to assume a role, create a role that includes the AWS managed policy `CloudWatchAgentAdminPolicy`, and use that role when you create a fleet.

```
sudo yum install -y amazon-cloudwatch-agent
sudo yum install -y https://dl.fedoraproject.org/pub/epel/epel-release-
latest-7.noarch.rpm
sudo yum install -y collectd
cat <<'EOF' > /tmp/config.json
{
    "agent": {
        "metrics_collection_interval": 60,
        "run_as_user": "root",
        "credentials": {
            "role_arn": "arn:aws:iam::account#:role/rolename"
        }
    },
    "logs": {
        "logs_collected": {
            "files": {
                "collect_list": [
                    {
                        "file_path": "/tmp/log",
                        "log_group_name": "gllog",
                        "log_stream_name": "{instance_id}"
                    }
                ]
            }
        }
    },
    "metrics": {
       "namespace": "GL_Metric",
        "append_dimensions": {
            "ImageId": "${aws:ImageId}",
            "InstanceId": "${aws:InstanceId}",
            "InstanceType": "${aws:InstanceType}"
        },
```

```
        "metrics_collected": {
            // Configure metrics you want to collect.
            // For more information, see Manually create or edit the CloudWatch agent
 configuration file.
        }
    }
}
EOF
sudo mv /tmp/config.json /opt/aws/amazon-cloudwatch-agent/bin/config.json
sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -
m ec2 -s -c file:/opt/aws/amazon-cloudwatch-agent/bin/config.json
sudo systemctl enable amazon-cloudwatch-agent.service
```

Upload a Realtime Servers script to Amazon GameLift

When you're ready to deploy Realtime Servers for your game, upload completed Realtime server script files to Amazon GameLift. Do this by creating a Amazon 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, Amazon GameLift assigns it a unique script ID (for 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 on your connection speed.

After you create the script resource, Amazon GameLift deploys the script with a new Realtime Servers fleet. Amazon GameLift installs your server script onto each instance in the fleet, placing the script files in `/local/game`.

To troubleshoot fleet activation problems related to the server script, see Debug Amazon GameLift fleet issues.

Package script files

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

You can store your zipped script files in either a local file directory or in an Amazon Simple Storage Service (Amazon S3) bucket.

Upload script files from a local directory

If you have your script files stored locally, you can upload them to Amazon GameLift from there. To create the script resource, use either the Amazon GameLift console or the AWS Command Line Interface (AWS CLI).

Amazon GameLift console

To create a script resource

1. Open the Amazon GameLift console.
2. In the navigation pane, choose **Hosting**, **Scripts**.
3. On the **Scripts** page, choose **Create script**.
4. On the **Create script** page, under **Script settings,** do the following:
   a. For **Name**, enter a script name.
   b. (Optional) For **Version**, enter version information. Because you can update the content of a script, version data can be helpful in tracking updates.
   c. For **Script source**, choose **Upload a .zip file**.
   d. For **Script files**, choose **Choose file**, browse for the .zip file that contains your script, and then choose that file.
5. (Optional) Under **Tags**, add tags to the script by entering **Key** and **Value** pairs.
6. Choose **Create**.

   Amazon GameLift assigns an ID to the new script and uploads the designated .zip file. You can view the new script, including its status, on the **Scripts** page.

AWS CLI

Use the `create-script` AWS CLI command to define the new script and upload your server script files.

To create a script resource

1. Place the .zip file into a directory where you can use the AWS CLI.
2. Open a command line window and switch to the directory where you placed the .zip file.

3. Enter the following **create-script** command and parameters. For the **--zip-file** parameter, be sure to add the string `fileb://` to the name of the .zip file. It identifies the file as binary so that Amazon GameLift processes the compressed content.

```
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
```

Example

```
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
```

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

4. To view the new script, call `describe-script`.

Upload script files from Amazon S3

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

To create a script resource

1. **Store your script files in an S3 bucket.** Create a .zip file containing your server script files and upload it to an S3 bucket in an AWS account that you control. Take note of the object URI—you need this when creating a Amazon GameLift script.

> **Note**
>
> Amazon GameLift doesn't support uploading from S3 buckets with names that contain a period (.).

2. **Give Amazon GameLift access to your script files.** To create an AWS Identity and Access Management (IAM) role that allows Amazon GameLift to access the S3 bucket containing your server script, follow the instructions in Set up an IAM service role for Amazon GameLift. After you create the new role, take note of its name, which you need when creating a script.
3. **Create a script.** Use the Amazon GameLift console or the AWS CLI to create a new script record. To make this request, you must have the IAM `PassRole` permission, as described in IAM permission examples for Amazon GameLift.

Amazon GameLift console

1. In the Amazon GameLift console, in the navigation pane, choose **Hosting**, **Scripts**.
2. On the **Scripts** page, choose **Create script**.
3. On the **Create script** page, under **Script settings**, do the following:
   a. For **Name**, enter a script name.
   b. (Optional) For **Version**, enter version information. Because you can update the content of a script, version data can be helpful in tracking updates.
   c. For **Script source**, choose **Amazon S3 URI**.
   d. Enter the **S3 URI** of the script object that you uploaded to Amazon S3, and then choose the **Object version**. If you don't remember the Amazon S3 URI and object version, choose **Browse S3**, and then search for the script object.
4. (Optional) Under **Tags**, add tags to the script by entering **Key** and **Value** pairs.
5. Choose **Create**.

   Amazon GameLift assigns an ID to the new script and uploads the designated .zip file. You can view the new script, including its status, on the **Scripts** page.

AWS CLI

Use the `create-script` AWS CLI command to define the new script and upload your server script files.

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

```
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
```

Example

```
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
```

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

3. To view the new script, call `describe-script`.

Update script files

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

You can also update the script content for a script resource. Amazon GameLift deploys script content to all fleet instances that use the updated script resource. When the updated script is deployed, instances use it when starting new game sessions. Game sessions that are already running at the time of the update don't use the updated script.

To update script files

- For script files stored locally, to upload the updated script .zip file, use either the Amazon GameLift console or the **update-script** command.
- For script files stored in an Amazon S3 bucket, upload the updated script files to the S3 bucket. Amazon GameLift periodically checks for updated script files and retrieves them directly from the S3 bucket.

Setting up Amazon GameLift fleets

This section provides detailed information about designing, building, and maintaining fleets for use with Amazon GameLift. You can use Amazon GameLift fleets to deploy custom game servers and Realtime Servers.

A fleet represents your hosting resources as a set of Amazon Elastic Compute Cloud (Amazon EC2) instances or physical hardware. A fleet's location determines where instances or hardware are deployed to host game sessions for your players. The size of a fleet, and the number of game sessions and players that it can support, depends on the number of instances or the amount of hardware that you give it. You can adjust virtual instances manually or by using automatic scaling.

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

To learn how to create fleets designed for your game's needs, start with Amazon GameLift fleet design guide. After your fleet is running, see Scaling Amazon GameLift hosting capacity, Add an alias to a Amazon GameLift fleet, and Setting up Amazon GameLift queues for game session placement.

Topics

Amazon GameLift fleet design guide

This design guide covers best practices for creating a fleet of hosting resources for use with Amazon GameLift. Choose a combination of hosting resources and learn how to configure them to suit your game.

Topics

- Choosing Amazon GameLift compute resources
- Manage how Amazon GameLift launches game servers
- Use Spot Instances with Amazon GameLift

Choosing Amazon GameLift compute resources

To deploy your game servers and host game sessions for your players, Amazon GameLift uses Amazon Elastic Compute Cloud (Amazon EC2) resources called *instances*, or your physical hardware. When setting up a new fleet using instances, decide what type of instances you need and how to run game server processes on them. When an managed EC2 fleet is active and ready to host game sessions, you can add or remove instances as needed to accommodate player demand.

You can deploy your Amazon GameLift game servers on a combination of two compute types:

- **Managed EC2** – Managed EC2 fleets use Amazon EC2 instances to host your game servers. Amazon GameLift manages the instances and removes the burden of hardware and software management from hosting your games.
- **Amazon GameLift Anywhere** – Amazon GameLift Anywhere fleets use your existing infrastructure to host game servers while Amazon GameLift manages your matchmaking and queues.

When you choose the compute resources for your fleet, consider the following factors:

- Available hardware
- Fleet location
- On-Demand Instances versus Spot Instances
- Operating systems
- Instance types
- Service quotas

Available hardware

Consider the existing infrastructure in your implementation. While you migrate games to Amazon GameLift, you can continue to use your infrastructure. With Amazon GameLift Anywhere, you can use your own infrastructure along with Amazon GameLift managed EC2 instances. You can also use your existing infrastructure to host games closer to your players than supported Amazon GameLift

locations can allow. For more information about setting up Amazon GameLift Anywhere fleets, see Create a Amazon GameLift Anywhere fleet.

Fleet location

Consider the geographic locations where you plan to deploy your game servers. Instance type availability varies by AWS Region and Local Zone.

For multi-location fleets, instance availability and quotas depend on a combination of the fleet's home Region and selected remote locations. For more information about fleet locations, see Amazon GameLift hosting locations.

For Amazon GameLift Anywhere fleets, you determine the location of your physical hardware. For more information about custom locations, see Amazon GameLift Anywhere.

On-Demand Instances versus Spot Instances

Amazon EC2 On-Demand Instances and Spot Instances offer the same hardware and performance, but they differ in availability and cost.

On-Demand Instances

You can acquire an On-Demand Instance when you need it, and keep it for as long as you want. On-Demand Instances have a fixed cost, meaning you pay for the amount of time that you use them, and there are no long-term commitments.

Spot Instances

Spot Instances can offer a cost-efficient alternative to On-Demand Instances by utilizing unused AWS computing capacity. Spot Instance prices fluctuate based on the supply and demand for each instance type in each location. AWS can interrupt Spot Instances whenever it needs the capacity back. Amazon GameLift uses queues and the FleetIQ algorithm to determine that AWS is going to interrupt a Spot Instance, it puts the instance in a recycling state. Then, when there are no active game sessions on the instance, Amazon GameLift tries to replace it.

For more information about how to use Spot Instances, see Use Spot Instances with Amazon GameLift.

Operating systems

Amazon GameLift instances support game server builds that run on Microsoft Windows or Amazon Linux. When you upload a game build to Amazon GameLift, specify the operating system for

the game. When you create an Amazon EC2 fleet to deploy the game build, Amazon GameLift automatically sets up instances with the build's operating system. For more information about supported game server operating systems, see Development support with Amazon GameLift.

When using a Amazon GameLift Anywhere fleet, you can use any operating system that your hardware supports. Amazon GameLift Anywhere fleets require you to deploy your game build to the hardware while using Amazon GameLift to manage your resources in one place.

Instance types

An Amazon EC2 fleet's instance type determines the kind of hardware that the instances use. Different instance types offer different combinations of computing power, memory, storage, and networking capabilities.

When choosing from available instance types for your game, consider:

- The compute architecture of your game server: x64 or Arm (AWS Graviton).

> **Note**
>
> Graviton Arm instances require an Amazon GameLift server build on Linux OS. Server SDK 5.1.1 or newer is required for C++ and C#. Server SDK 5.0 or newer is required for Go. These instances provide no out-of-the-box support for Mono installation on Amazon Linux 2023 (AL2023) or Amazon Linux 2 (AL2).

- The computing, memory, and storage requirements of your game server build.
- The number of server processes that you plan to run per instance.

By using a larger instance type, you may be able to run multiple server processes on each instance. This can reduce the number of instances required to meet player demand.

For more information:

- About instance types, see Amazon EC2 Instance Types.
- About running multiple processes per instance, see Manage how Amazon GameLift launches game servers.

Service quotas

To see the default service quotas for Amazon GameLift, and the current quotas for your AWS account, do the following:

- For general service quota information for Amazon GameLift, see Amazon GameLift endpoints and quotas in the *AWS General Reference*.
- For a list of available instance types per location for your account, open the Service quotas page of the Amazon GameLift console. This page also displays your account's current usage for each instance type in each location.
- For a list of your account's current quotas for instance types per Region, run the AWS Command Line Interface (AWS CLI) command `describe-ec2-instance-limits`. This command returns the number of active instances that you have in your default Region (or in another Region that you specify).

As you prepare to launch you game, fill out a launch questionnaire in the Amazon GameLift console. The Amazon GameLift team uses the launch questionnaire to determine the correct quotas and limits for your game.

Manage how Amazon GameLift launches game servers

You can set up an managed EC2 fleet's runtime configuration to run multiple game server processes per instance. This uses your hosting resources more efficiently.

How a fleet manages multiple processes

Amazon GameLift uses a fleet's runtime configuration to determine the type and number of processes to run on each instance. A runtime configuration contains at least one server process configuration that represents one game server executable. You can define additional server process configurations to run other types of processes related to your game. Each server process configuration contains the following information:

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

When an instance in the fleet activates, it launches the set of server processes defined in the runtime configuration. With multiple processes, Amazon GameLift staggers the launch of each

process. Server processes have a limited life span. As they end, Amazon GameLift launches new processes to maintain the number and type of server processes defined in the runtime configuration.

You can change the runtime configuration at any time by adding, changing, or removing server process configurations. Each instance regularly checks for updates to the fleet's runtime configuration to implement the changes. Here's how Amazon GameLift adopts runtime configuration changes:

1. The instance sends a request to Amazon GameLift for the latest version of the runtime configuration.
2. The instance compares its active processes to the latest runtime configuration, and then does the following:
   - If the updated runtime configuration removes a server process type, then active server processes of this type continue to run until they end. The instance doesn't replace these server processes.
   - If the updated runtime configuration decreases the number of concurrent processes for a server process type, then excess server processes of this type continue to run until they end. The instance doesn't replace these excess server processes.
   - If the updated runtime configuration adds a new server process type or increases the concurrent processes for an existing type, then the instance starts new server processes, up to the Amazon GameLift maximum. In this case, the instance launches new server processes as existing processes end.

Optimize a fleet for multiple processes

To use multiple processes on a fleet, do the following:

- Create a build that contains the game server executables that you want to deploy to a fleet, and then upload the build to Amazon GameLift. All game servers in a build must run on the same platform and use the Amazon GameLift Server SDK.
- Create a runtime configuration with one or more server process configurations and multiple concurrent processes.
- Integrate game clients with the AWS SDK version 2016-08-04 or later.

To optimize fleet performance, we recommend that you do the following:

- Handle server process shutdown scenarios so that Amazon GameLift can recycle processes efficiently. For example:
  - Add a shutdown procedure to your game server code that calls the server API `ProcessEnding()`.
  - Implement the callback function `OnProcessTerminate()` in your game server code to handle termination requests from Amazon GameLift.
- Make sure that Amazon GameLift shuts down and relaunches unhealthy server processes. Report the health status back to Amazon GameLift by implementing the `OnHealthCheck()` callback function in your game server code. Amazon GameLift automatically shuts down server processes that are reported unhealthy for three consecutive reports. If you don't implement `OnHealthCheck()`, then Amazon GameLift assumes that a server process is healthy, unless the process fails to respond to a communication.

Choose the number of processes per instance

When deciding on the number of concurrent processes to run on an instance, keep in mind the following:

- Amazon GameLift limits each instance to a maximum number of concurrent processes. The sum of all concurrent processes for a fleet's server process configurations can't exceed this quota.
- To maintain acceptable performance levels, the Amazon EC2 instance type might limit the number of processes that can run concurrently. Test different configurations for your game to find the right number of processes for your preferred instance type.
- Amazon GameLift doesn't run more concurrent processes than the total number configured. This means that the transition from the previous runtime configuration to the new configuration might happen gradually.

Use Spot Instances with Amazon GameLift

When setting up yourAmazon GameLift managed EC2 fleet, you can use Spot Instances, On-Demand Instances, or a combination. Learn more about how Amazon GameLift uses Spot Instances in On-Demand Instances versus Spot Instances. To use spot fleets, your game integration requires the adjustments listed on this page.

Are you using FlexMatch for matchmaking? You can add Spot fleets to your existing game session queues for matchmaking placements.

1. **Design your game session queue for Spot instances.**

   Managing game session placement with a queue is best practice, and it's required when using Spot Instances. To design your queue consider the following:

   - Locations – To achieve the best player experience, choose locations geographically close to your players.
   - Instance types – Consider your game servers hardware requirements and availability of instances in the locations you chose.

   To try a queue that optimizes Spot availability and resiliency, see Tutorial: Set up a game session queue for Spot Instances.

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. See Create a Amazon GameLift managed fleet for help creating and configuring new fleets.

3. **Create your game session queue.**

   Add the fleet destinations, configure the game session placement process, and define placement priorities. See Create a game session queue for help creating and configuring the new queue.

4. **Update your game client service to use the queue.**

   When your game client uses a queue to request resources, the queue avoids resources with a high chance of interruption and selects the location that matches your defined priorities. For help implementing game session placements in your game client, see Create game sessions.

5. **Update your game server to handle a Spot interruption.**

   AWS can interrupt Spot Instances with a 2 minute notification, when it needs the capacity back. Set up your game server to handle interruption to minimize player impact.

   Before AWS reclaims a Spot Instance, it sends a termination notification. Amazon GameLift passes the notification to all affected server processes by invoking the Amazon GameLift Server SDK callback function `onProcessTerminate()`. Implement this callback to end the game session or move the game session and players to a new instance. See Respond to a server process shutdown notification for help implementing `onProcessTerminate()`.

> **Note**
>
> AWS makes every effort to provide the notification before it reclaims and instance, but it's possible that AWS reclaims the Spot Instance before the warning arrives. Prepare your game server to handle unexpected interruptions.

6. **Review the performance of your Spot fleets and queues.**

   View Amazon GameLift metrics in the Amazon GameLift console or with Amazon CloudWatch to review performance. For more information about Amazon GameLift metrics, see Monitor Amazon GameLift with Amazon CloudWatch. Key metrics include:

   - Interruption rate – Use the `InstanceInterruptions` and `GameSessionInterruptions` metrics to track the number and frequency of Spot-related interruptions for instances and game sessions. Game sessions that reclaimed by AWS have a status of `TERMINATED` and a status reason of `INTERRUPTED`.
   - Queue effectiveness – Track placement success rates, average wait time, and queue depth to confirm that Spot fleets don't impact your queue performance.
   - Fleet usage – Monitor data on instances, game sessions and player sessions. Usage for your On-Demand fleets can be an indicator that queues are avoiding placements into your Spot fleets to avoid disruption.

Create a new Amazon 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 you upload to Amazon GameLift.

Topics

- How Amazon GameLift fleet creation works
- Create a Amazon GameLift managed fleet
- Create a Amazon GameLift Anywhere fleet

How Amazon GameLift fleet creation works

When you create a new fleet, Amazon GameLift starts a workflow that creates a fleet with one Amazon Elastic Compute Cloud (Amazon EC2) instance in each fleet location. As Amazon GameLift

completes each step of the workflow, the fleet emits events and Amazon GameLift updates the fleet's status. You can track all events using the Amazon GameLift console or by calling the Amazon GameLift API operation DescribeFleetEvents. You can also track the status of individual locations using DescribeFleetLocationAttributes.

EC2 fleet creation workflow:

- Amazon GameLift creates a fleet resource in the fleet's home Region and in each remote location defined in the fleet.
- Amazon GameLift sets the desired capacity to one instance.
- Amazon GameLift sets the fleet and location status to **New**.
- Amazon GameLift begins writing events to the fleet event log.
- Amazon GameLift allocates requested computing resources for one new instance in each fleet location.
- Amazon GameLift downloads the game server files to each instance and sets the fleet status to **Downloading**.
- Amazon GameLift validates the downloaded game server files on each instance to verify that no errors occurred during downloading. Amazon GameLift sets the fleet status to **Validating**.
- Amazon GameLift builds the game server on each instance and sets the fleet status to **Building**.
- Amazon GameLift begins launching server processes on each instance, following instructions in the fleet's runtime configuration. If you configured the fleet to run multiple concurrent server processes per instance, then Amazon GameLift staggers the process launches by a few seconds. As each process comes online, it reports readiness back to Amazon GameLift. Amazon GameLift sets the fleet status to **Activating**.
- Amazon GameLift sets the fleet statuses and the location statuses to **Active** as server processes report readiness.

Amazon GameLift Anywhere fleet creation

- Amazon GameLift creates a fleet resource. For the fleet's home Region and each custom location defined in the fleet, Amazon GameLift sets the fleet and location status to **New**.
- Amazon GameLift begins writing events to the fleet event log.
- After one server process in a fleet notifies Amazon GameLift that it's ready, Amazon GameLift sets the fleet status and the location status to **Active**. As server processes in other fleet locations report readiness, Amazon GameLift sets the status of each fleet location to **Active**.

For help with troubleshooting fleet creation issues, see Debug Amazon GameLift fleet issues.

Create a Amazon GameLift managed fleet

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

After you create a new managed EC2 fleet, the fleet's status passes through several stages as Amazon GameLift deploys the fleet and installs and starts the game servers. The fleet is ready to host game sessions, after it reaches `ACTIVE` status. For help with fleet creation issues, see Debug Amazon GameLift fleet issues.

Console

To create a managed EC2 fleet

1. In the Amazon GameLift console, in the navigation pane, choose **Fleets**.
2. On the **Fleets** page, choose **Create fleet**.
3. Choose **Managed EC2**.
4. On the **Fleet details** page do the following:
    a. For **Name**, enter a fleet name. We 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.
    b. For **Description**, provide a short description of the fleet.
    c. For **Binary type**, select **Build** or **Script** to define the game server type that Amazon GameLift deploys to this fleet.
    d. Select a **Script** or **Build** from the dropdown list of uploaded scripts or builds.
5. (Optional) Under **Additional details** for the following:
    a. For **Instance role**, specify an IAM role that authorizes applications in your game build to access other AWS resources in your account. For more information, see Communicate with other AWS resources from your fleets. To create a fleet with an instance role, your account must have the IAM `PassRole` permission. For more information, see IAM permission examples for Amazon GameLift.

       If you want to authorize applications that are not server executables, such as a CloudWatch agent, enable the shared credentials option.

You can't update these settings after fleet creation.

b. For **Certification generation**, choose to have Amazon 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, Amazon GameLift also creates a new DNS entry with the certificate. Use these resources to set up authentication and encryption for your game.

c. For **Metric group**, Enter the name of a new or existing fleet metric group. You can aggregate the metrics for multiple fleets by adding them to the same metric group.

You can't update the metric group after fleet creation.

6. Choose **Next**.
7. On the **Select locations** page, select one or more additional remote locations to deploy instances to. The home Region is automatically selected based on the Region you are accessing the console from. If you select additional locations, fleet instances are also deployed in these locations.

> **Important**
>
> To use Regions that aren't enabled by default, enable them in your AWS account.
>
> - Fleets with Regions that aren't enabled that you created before February 28, 2022 are unaffected.
> - To create new multi-location fleets or to update existing multi-location fleets, first enable any Regions that you choose to use.
>
> For more information about Regions that aren't enabled by default and how to enable them, see Managing AWS Regions in the *AWS General Reference*.

8. Choose **Next**.
9. On the **Define instance details page**, choose

a. **On-demand** or **Spot** instances for this fleet. For more information about fleet types, see On-Demand Instances versus Spot Instances.

b. From the **Filter architecture** menu choose **x64** or **Arm**.

> **Note**
>
> Graviton Arm instances require an Amazon GameLift server build on Linux OS. Server SDK 5.1.1 or newer is required for C++ and C#. Server SDK 5.0 or newer is required for Go. These instances provide no out-of-the-box support for Mono installation on Amazon Linux 2023 (AL2023) or Amazon Linux 2 (AL2).

For information on Amazon EC2 Arm architectures, see AWS Graviton Processor and Amazon EC2 instance types.

For information on the instance types supported by Amazon GameLift, see the `EC2InstanceType` values under CreateFleet() request parameters.

10. Select an Amazon EC2 **Instance type** from the list. For more information about choosing an instance type, see Instance types. After you create the fleet, you can't change the instance type.
11. Choose **Next**.
12. On the **Configure runtime** page, under **Runtime configuration** do the following:

    a. For **Launch path**, enter the path to the game executable in your build or script. On Windows instances, game servers are built to the path `C:\game`. On Linux instances, game servers are built to `/local/game`. Examples: **`C:\game\MyGame\server.exe`**, **`/local/game/MyGame/server.exe`**, or **`MyRealtimeLaunchScript.js`**.

    b. (Optional) For **Launch parameters**, enter information to pass to your game executable as a set of command line parameters. Example: **`+sv_port 33435 +start_lobby`**.

    c. For **Concurrent processes**, choose the number of server processes to run concurrently on each instance in the fleet. Review the Amazon GameLift limits on number of concurrent server processes.

    Limits on concurrent server processes per instance apply to the total of concurrent processes for all configurations. If you configure the fleet to exceed the limit, the fleet can't activate.
13. Under **Game session activation**, provide limits for activating new game sessions on the instances in this fleet:

a. For **Max concurrent game session activation**, enter the number of game sessions on an instance that activate 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.

b. For **New activation timeout**, enter how long to wait for a session to activate. If the game session doesn't move to `ACTIVE` status before the timeout, Amazon GameLift terminates the game session activation.

14. (Optional) Under **EC2 port settings**, do the following:

    a. Choose **Add port setting** to define access permissions for inbound traffic connecting to the server process deployed on the fleet.

    b. For **Type**, choose **Custom TCP** or **Custom UDP**.

    c. For **Port range**, Enter a range of port numbers that allow inbound connections. A port range must use the format `nnnnn[-nnnnn]`, with values between 1026 and 60000. Example: **`1500`** or **`1500-20000`**.

    d. For **IP address range**, Enter a range of IP addresses. Use CIDR notation. Example: **`0.0.0.0/0`** (This example allows access to anyone trying to connect.)

15. (Optional) Under **Game session resource settings** do the following:

    a. For **Game scaling protection policy**, Turn on or off scaling protection. Amazon GameLift won't terminate instance with protection during a scale down event if they're hosting an active game session.

    b. For **Resource creation limit**, enter a maximum number of game sessions a player can create during the policy period.

16. Choose **Next**.

17. (Optional) Add tags to the build by entering **Key** and **Value** pairs. Choose **Next** to continue to fleet creation review.

18. Choose **Create**. Amazon 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. For more information, see Manage your Amazon GameLift fleets. You can update fleet capacity after the fleet has reached ACTIVE status. For more information, see Scaling Amazon 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. For more information about the `create-fleet` command, see `create-fleet` in the *AWS CLI Command Reference*.

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

- The fleet uses c5.large On-Demand Instances with the operating system that's appropriate for the selected game build.
- It deploys 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.
- On each instance, Amazon GameLift allows two new game sessions to activate at the same time. It also terminates any activating game session if they aren't ready to host players within 300 seconds.
- All game sessions hosted on instances in this fleet have game session protection turned on.
- Individual players can create three new game sessions within a 15-minute period.
- Each game session hosted on this fleet has a connection point that falls within the specified IP address and port ranges.
- Amazon GameLift adds metrics for this fleet 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"
```

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 then initiates the fleet activation process and sets the fleet status and the location statuses to **New**. You can track the fleet's status and view other fleet information using these CLI commands:

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

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

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

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

Create a Amazon GameLift Anywhere fleet

Use Amazon GameLift to integrate hardware from your environment into your Amazon GameLift game hosting. Amazon GameLift Anywhere registers your hardware with Amazon GameLift in an Anywhere fleet. You can integrate Anywhere and managed EC2 fleets in matchmaker and game session queues to manage matchmaking and game placement.

For more information about testing your game servers with Amazon GameLift Anywhere, see Test your integration using Amazon GameLift Anywhere fleets.

To get started, Development support with Amazon GameLift version 5 or greater and review the following important concepts for using a Amazon GameLift Anywhere fleet.

Custom locations

Amazon GameLift Anywhere fleets use custom locations to represent the physical locations of your infrastructure.

Device registration

For a Amazon GameLift Anywhere fleet to communicate with your compute resources, first register your device. You can complete device registration from the Amazon GameLift AWS SDK using the `RegisterCompute` operation. This operation uses the IP address of the device to associate it with a fleet location and communicate with Amazon GameLift.

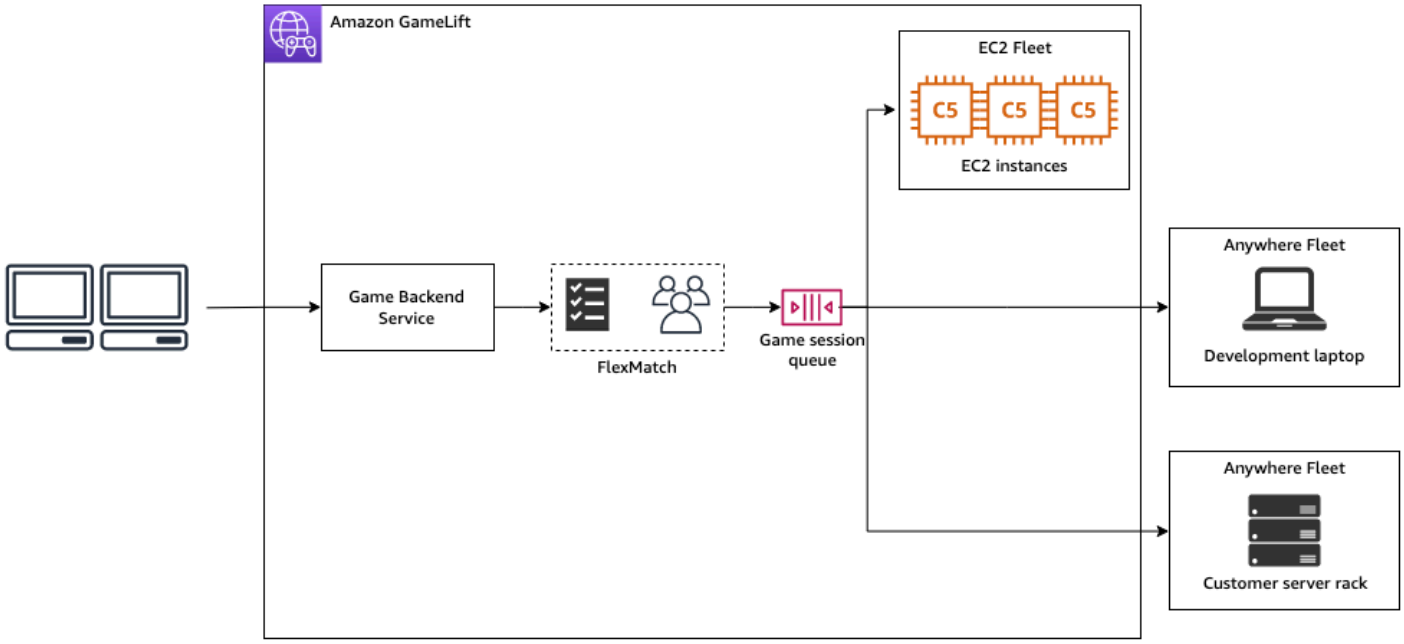
Authentication tokens

When you initialize a game server on your compute, the Amazon GameLift Server SDK uses an auth token to authenticate your game server to Amazon GameLift. You can re-use the same auth token for all game servers on the same compute, up to the auth token expiration time. To retrieve the auth token, call the `get-compute-auth-token` AWS Command Line Interface (AWS CLI) command. Pass the token to each game server as needed.

Game sessions

Each game session on a compute uses the same authentication token created while registering the compute to a fleet location.

The following diagram shows a game session queue that uses FlexMatch matchmaking and multiple fleets. The fleets include an EC2 fleet with C5 instances, an Anywhere fleet with a development laptop, and an Anywhere fleet with a customer-hosted server rack.



Topics

- Create a custom location
- Create a fleet
- Register your compute
- Run a server process
- Create game sessions
- Migrate to managed EC2

Create a custom location

To get started hosting games on your compute resources, create a custom location describing where your compute resides.

Console

To create a custom location

1. Open the Amazon GameLift console.
2. In the navigation pane, under **Hosting**, choose **Locations**.
3. On the **Locations** page, choose **Create location**.
4. In the **Create location** dialog box, do the following:

   a. Enter a **Location name**. This labels the location of your hardware that Amazon GameLift uses to run your games in Anywhere fleets. Amazon GameLift appends the name of your custom location with **custom-**.

   b. (Optional) Add tags as key-value pairs to your custom location. Choose **Add new tag** for each tag that you want to add.

   c. Choose **Create**.

AWS CLI

Create a custom location using the `create-location` command. The `location-name` labels the location of your hardware that Amazon GameLift uses to run your games in Anywhere fleets. When creating your custom location, the location name must start with `custom-`.

```
aws gamelift create-location \
    --location-name custom-location-1
```

Output

```
{
    "Location": {
        "LocationName": "custom-location-1",
        "LocationArn": "arn:aws:gamelift:us-east-1:111122223333:location/custom-
location-1"
    }
}
```

Create a fleet

Use either the Amazon GameLift console or the AWS CLI to create an Anywhere fleet.

After you create a new Anywhere fleet, the fleet's status moves from `NEW` to `ACTIVE`. When it reaches `ACTIVE` status, the fleet is ready to host game sessions. For help with fleet creation issues, see Debug Amazon GameLift fleet issues.

Console

To create an Anywhere fleet

1. Open the Amazon GameLift console.
2. In the navigation pane, under **Hosting**, choose **Fleets**.
3. On the **Fleets** page, choose **Create fleet**.
4. On the **Compute type** step, choose **Anywhere**, and then choose **Next**.
5. On the **Fleet details** step, define the details, and then choose **Next**.
6. On the **Custom locations** step, select the custom location that you created, and then choose **Next**. Amazon GameLift automatically selects the home AWS Region as the Region that you're creating the fleet in. You can use the home Region to access and use your resources.
7. Complete the remaining fleet creation steps, and then choose **Submit** to create your Anywhere fleet.

AWS CLI

Create an Anywhere fleet using the `create-fleet` command. Include your custom location in `locations`. Amazon GameLift creates the fleet in your home Region and in the custom locations that you provide. In the following example, replace *`FleetName`* and *`custom-location-1`* with your own information. The variable `custom-location-1` is the name of the location created in the Create a custom location step.

```
aws gamelift create-fleet \
--name FleetName \
--compute-type ANYWHERE \
--locations "Location=custom-location-1"
```

Example output

```
{
    "FleetAttributes": {
        "FleetId": "fleet-cebb4da2-52a8-4c27-9b85-587f945c6445",
```

```
        "FleetArn": "arn:aws:gamelift:us-east-1:111122223333:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445",
        "Name": "HardwareAnywhere",
        "CreationTime": "2023-02-23T17:57:42.293000+00:00",
        "Status": "ACTIVE",
        "MetricGroups": [
            "default"
        ],
        "CertificateConfiguration": {
            "CertificateType": "DISABLED"
        },
        "ComputeType": "ANYWHERE"
    }
}
```

Register your compute

To register your compute resource in the fleet that you created, use the [register-compute] command. Replace the *fleet-id* with the `fleet-id` returned in the previous step or fleet ARN found in the details page of your fleet in the console. Replace the *compute-name*, and *ip-address* with the IP address of your compute resource.

> **Note**
>
> Amazon GameLift recommends calling both the `register-compute` and `get-compute-auth-token` commands from a script or process manager separate from your game server.

```
aws gamelift register-compute \
    --compute-name HardwareAnywhere \
    --fleet-id arn:aws:gamelift:us-east-1:111122223333:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445 \
    --ip-address 10.1.2.3 \
    --location custom-location-1
```

Example output

```
{
    "Compute": {
        "FleetId": "fleet-cebb4da2-52a8-4c27-9b85-587f945c6445",
```

```
        "FleetArn": "arn:aws:gamelift:us-east-1:111122223333:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445",
        "ComputeName": "HardwareAnywhere",
        "ComputeArn": "arn:aws:gamelift:us-east-1:111122223333:compute/
HardwareAnywhere",
        "IpAddress": "10.1.2.3",
        "ComputeStatus": "Active",
        "Location": "custom-location-1",
        "CreationTime": "2023-02-23T18:09:26.727000+00:00",
        "GameLiftServiceSdkEndpoint": "wss://us-east-1.api.amazongamelift.com"
    }
}
```

Run a server process

1. Get the authentication token for your compute resource from the fleet that you created.

   Your game server uses the authentication token to authenticate with Amazon GameLift. Each authentication token has an expiration token. To continue using the compute resource to host your game server, retrieve a new authentication token before the expiration.

   > **Note**
   >
   > Amazon GameLift recommends calling both the `register-compute` and `get-compute-auth-token` commands from a script or process manager separate from your game server.

   In the following example, replace the `fleet-id` with the ARN or fleet ID of the fleet created in the previous steps. Replace the `compute-name` with the name of the compute you created using the `register-compute` command in a previous step.

   ```
   aws gamelift get-compute-auth-token \
       --fleet-id arn:aws:gamelift:us-east-1:111122223333:fleet/fleet-
   cebb4da2-52a8-4c27-9b85-587f945c6445 \
       --compute-name HardwareAnywhere
   ```

   Example output:

   ```
   {
   ```

```
    "FleetId": "fleet-cebb4da2-52a8-4c27-9b85-587f945c6445",
    "FleetArn": "arn:aws:gamelift:us-east-1:111122223333:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445",
    "ComputeName": "HardwareAnywhere",
    "ComputeArn": "arn:aws:gamelift:us-east-1:111122223333:compute/
HardwareAnywhere",
    "AuthToken": "0c728041-3e84-4aaa-b927-a0fb202684c0",
    "ExpirationTimestamp": "2023-02-23T18:47:54+00:00"
}
```

2. Run an instance of your game server executable.

   To run your game server, initialize your game server by calling `InitSDK()` and passing it your server parameters. For more information about server parameters, see ServerParameters.

   Server SDK input:

```
//Define the server parameters
ServerParameters serverParameters = new ServerParameters(
    webSocketUrl=wss://us-east-1.api.amazongamelift.com,
    processId=PID1234,
    hostId=HardwareAnywhere,
    fleetId=arn:aws:gamelift:us-east-1:111122223333:fleet/fleet-
cebb4da2-52a8-4c27-9b85-587f945c6445,
    authToken=0c728041-3e84-4aaa-b927-a0fb202684c0);

//InitSDK establishes a connection with GameLift's websocket server for
 communication.
var initSDKOutcome = GameLiftServerAPI.InitSDK(serverParameters);
```

3. After the server process is ready to host a game session, call `ProcessReady()` from your game server to Amazon GameLift. For more information about process parameters, see ProcessParameters

```
// Set parameters and call ProcessReady
var processParams = new ProcessParameters(
   this.OnStartGameSession,
   this.OnProcessTerminate,
   this.OnHealthCheck,
   this.OnUpdateGameSession,
   port=1024,
   new LogParameters(new List<string>()         // Examples of log and error files
 written by the game server
```

```
   {
      "C:\\game\\logs",
      "C:\\game\\error"
   })
);

var processReadyOutcome = GameLiftServerAPI.ProcessReady(processParams);
```

Create game sessions

1. Add logic to your game server so that your server process responds to the onStartGameSession() message with ActivateGameSession(). This operation has no parameters, but it sends an acknowledgement to Amazon GameLift that your server received and accepted the create game session message.

   ```
   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();
   }
   ```

2. From your game client backend service, start your game session using the start-matchmaking, start-game-session-placement, or create-game-session command.

   ```
   aws gamelift create-game-session \
       --fleet-id arn:aws:gamelift:us-east-1:682428703967:fleet/fleet-
   cebb4da2-52a8-4c27-9b85-587f945c6445 \
       --name GameSession1 \
       --maximum-player-session-count 2 \
       --location custom-location-1
   ```

   Example output:

   ```
   GameSession {
     FleetId = arn:aws:gamelift:us-east-1:682428703967:fleet/fleet-
   cebb4da2-52a8-4c27-9b85-587f945c6445,
     GameSessionId = 4444-4444,
     Name = GameSession1,
   ```

```
  Location = custom-location-1,
  IpAddress = 10.2.3.4,
  Port = 1024,
  ...
}
```

Amazon GameLift sends an onStartGameSession() message to your registered server process. The message contains the GameSession object from the previous step with game properties, game sessions data, matchmaker data, and more about the game session.

3. When the game session is complete, end the game server process.

   Server SDK input:

```
var processEndingOutcome = GameLiftServerAPI.ProcessEnding();
if (processReadyOutcome.Success)
   Environment.Exit(0);
// otherwise, exit with error code
Environment.Exit(errorCode);
```

4. Start another game server process by calling ProcessReady(processParams).

Migrate to managed EC2

After you've developed your game server and you're ready to prepare for production, you can have Amazon GameLift manage your hardware. To migrate to a managed EC2 fleet, upload your build to Amazon GameLift and create a managed EC2 fleet. For more information about uploading your build and setting up a fleet, see Upload a custom server build to Amazon GameLift and Create a Amazon GameLift managed fleet.

Manage your Amazon GameLift fleets

Use the Amazon GameLift console or the AWS CLI to update your fleet settings, change remote locations, or delete a fleet.

Update a fleet configuration

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

Amazon GameLift console

1. In the Amazon GameLift console, in the navigation pane, choose **Fleets**.
2. Choose the fleet you want to update. A fleet must be in `ACTIVE` status before you can edit it.
3. On the Fleet detail page, in any of the following sections, choose **Edit**.
   - **Fleet settings**
     - Change the fleet attributes such as **Name** and **Description**.
     - Add or remove **Metric groups**, that Amazon CloudWatch uses to track aggregated Amazon GameLift metrics for multiple fleets.
     - Update **Resource creation limit** settings.
     - Turn game session protection on or off.
   - **Runtime configuration** – You can change any of the following settings of your runtime configurations and add or remove runtime configurations.
     - Change the **Launch path** of your game server.
     - Add, remove, or change optional **Launch parameters**.
     - Change the number of **Concurrent processes** that your game servers run.
   - **Game session activation** – Change how you want server processes to run and host game sessions by updating **Max concurrent game session activations** and **New activation timeout**.
   - **EC2 port settings** – Update the IP addresses and port ranges that allow inbound access to the fleet.
4. Choose **Confirm** to save changes.

AWS CLI

Use the following AWS CLI commands to update a fleet:

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

Update fleet locations

You can add or remove a fleet's remote locations using the Amazon GameLift console or the AWS CLI. You can't change a fleet's home Region.

Amazon GameLift console

1. In the Amazon GameLift console, in the navigation pane, choose **Fleets**.
2. Choose the fleet you want to update. A fleet must be in `ACTIVE` status before you can edit it.
3. On the Fleet detail page, choose the **Locations** tab to view the fleet's locations.
4. To add new remote locations, choose **Add** and select the locations you want to deploy instances to. This list doesn't include instances where the fleet's instance type isn't available.
5. With new locations selected, choose **Add**. Amazon GameLift adds the new locations to the list, with status set to `NEW`. Amazon GameLift then begins provisioning an instance in each added location and preparing it to host game sessions.
6. To remove existing remote locations from the fleet, use the check boxes to select one or more listed locations.
7. With one or more fleets selected, choose **Remove**. The removed locations remain in the list, with status set to `DELETING`. Amazon GameLift then begins the process of terminating activity in the removed location. If there are active instances that are hosting game sessions, Amazon GameLift uses the game server termination process to gracefully end game sessions, terminate game servers, and shut down instances.

AWS CLI

Use the following AWS CLI commands to update fleet locations:

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

Delete a fleet

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

> **Note**
>
> If the fleet has a VPC peering connection, first request authorization by calling CreateVpcPeeringAuthorization. Amazon GameLift deletes the VPC peering connection during fleet deletion.

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

Amazon GameLift console

1. In the Amazon GameLift console, in the navigation pane, choose **Fleets**.
2. Choose the fleet you want to delete. You can only delete fleets in `ACTIVE` or `ERROR` status.
3. Choose **Delete**.
4. In the **Delete fleet** dialog box, confirm the deletion by entering **`delete`**.
5. Choose **Delete**.

AWS CLI

Use the following AWS CLI command to delete a fleet:

- delete-fleet

Add an alias to a Amazon GameLift fleet

A Amazon 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. Use aliases instead of specific fleet IDs to 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** – Routes player traffic to a specified fleet ID. You can update the fleet ID for an alias at any time.
- **Terminal** – Passes a message back to the client. For example, you can direct players who are using an out-of-date client to a location where they can get an upgrade.

Fleets have a finite lifespan, and there are several reasons to switch out fleets during the life of a game. You can't update a fleet's game server build or change certain computing resource attributes on an existing fleet. Instead, 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 useful in games that don't 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 Amazon GameLift service. Without aliases, a fleet switch requires updates to your game code and possibly distribution of an updated game clients to players.

When updating the fleet-id an alias points to, there is a transition period of up to 2 minutes where game sessions on the alias may end up on the old fleet.

Create a new alias

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

1. In the Amazon GameLift console, in the navigation pane, choose **Aliases**.
2. On the **Aliases** page, choose **Create alias**. We recommend including the fleet type in your alias names. This makes it much easier to identify the fleet type when viewing a list of aliases.
3. On the **Create alias** page, under **Alias details**, do the following:
    a. For **Name**, enter an alias name.
    b. For description, enter a short description for identification.
    c. Choose **Simple** or **Terminal** routing type.
4. (Optional) Under **Tags**, add tags to the alias by entering **Key** and **Value** pairs.
5. Choose **Create**.

Edit an alias

You can edit an alias using the Amazon GameLift console or with the AWS CLI command update-alias.

1. In the Amazon GameLift console, in the navigation pane, choose **Aliases**.
2. On the **Aliases** page, choose the alias you want to edit.
3. On alias page choose **Edit**.
4. 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.
5. Choose **Save changes**.

Debug Amazon GameLift fleet issues

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

Fleet creation issues

When a fleet is created, the Amazon GameLift service initiates a workflow that deploys a new instance in each of the fleet's locations and prepares it to run you game servers. For a detailed description, see How Amazon GameLift fleet creation works. 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. Amazon 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 Amazon 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 Amazon 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 Amazon 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 **Fleet** detail page, Details)section 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 Amazon 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).

VPC peering connection request failed.

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

Server process issues

Server processes start but fail quickly or report poor health.

Other than issues with your game build, this outcome can happen when trying to run too many server processes simultaneously on the instance. The optimum number of concurrent processes depends on both the instance type and your game server's resource requirements. Try reducing the number of concurrent processes, which is set in the fleet's runtime configuration, to see if performance improves. You can change a fleet's runtime configuration using either the Amazon GameLift 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 Amazon GameLift. 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 Amazon 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, 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:

- 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 connect to Amazon GameLift fleet instances.

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 connect to Amazon GameLift fleet instances

You can connect to any instance in your active Amazon GameLift managed EC2 fleets. Common reasons to access an instance include:

- Troubleshoot issues with your game server integration
- Fine-tune your runtime configuration and other fleet-specific settings
- Get real-time game server activity, such as log tracking.
- Run benchmarking tools using actual player traffic.
- Investigate specific issues with a game session or server process.

When connecting to an instance, consider these potential issues:

- You can connect to instances in active fleets. Non-active fleets, those activating or are in an error state, might be accessible for a short period of time. For help with fleet activation issues, see Debug Amazon GameLift fleet issues.
- Connecting to an active instance doesn't affect the instance's hosting activity. The instance continues to start and stop server processes based on the runtime configuration. It activates and hosts game session. It might shut down in response to a scale down event or other event.
- Any changes you make to files or settings on the instance might impact the instance's active game sessions and connected players.

The following instructions describe how to remotely connect to an instance using the AWS command line interface (CLI). You can also make programmatic calls using the AWS SDK, as documented in the Amazon GameLift service API reference.

Gather instance data

Collect the following information:

- The ID of the instance you want to connect to. You can use either the instance ID or ARN.
- The Amazon GameLift server SDK version being used on the instance. The server SDK is integrated with the game build running on the instance.

To retrieve instance data

The following steps assume you have a managed EC2 fleet ID for the instance you want to connect to.

1. **Get the compute name.**

Call list-compute for the managed EC2 fleet to get a list of all active computes in the fleet. For a single-location fleet, specify the fleet ID or ARN. For a multi-location fleet, specify the fleet ID or ARN and a location. For a managed EC2 fleet, computes are EC2 instances and the returned property `ComputeName` is the instance ID. For example:

Request

```
aws gamelift list-compute \
  --fleet-id  "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa" \
  --location ""sa-east-1"
```

Response

```
{
  "ComputeList": [
    {
      "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
      "FleetArn": "arn:aws:gamelift:us-west-2::fleet/
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
      "ComputeName": "i-0abc12d3e45fa6b78",
      "IpAddress": "00.00.000.00",
      "DnsName":
 "b08444ki909kvqu6zpw3is24x5pyz4b6m05i3jbxvpk9craztu0lqrbbrbnbkks.uwp57060n1k6dnlnw49b78hg1
west-2.amazongamelift.com",
      "ComputeStatus": "Active",
      "Location": "sa-east-1",
      "CreationTime": "2023-07-09T22:51:45.931000-07:00",
      "OperatingSystem": "AMAZON_LINUX",
      "Type": "c4.large"
    }
  ]
}
```

2. **Find the server SDK version.**

   The server SDK version is an attribute of a build resource.

   a. Call describe-fleet-attributes with a fleet ID to get the fleet's build ID and ARN.

   b. Call describe-build with the build ID or ARN to get the build's server SDK version.

      For example:

Request

```
aws gamelift describe-fleet-attributes /
  --fleet-ids  "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa"
```

Response

```
{
  "FleetAttributes": [
    {
      "FleetId": "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa",
      "ComputeType": "EC2",
      "BuildId": "build-3333cccc-44dd-55ee-66ff-00001111aa22",
      . . .
    }
  ]
}
```

Request

```
aws gamelift describe-build /
  --build-id "build-3333cccc-44dd-55ee-66ff-00001111aa22"
```

Response

```
"Build": {
  "BuildId": "build-1111aaaa-22bb-33cc-44dd-5555eeee66ff",
  "Name": "My_Game_Server_Build_One",
  "OperatingSystem": "AMAZON_LINUX_2",
  "ServerSdkVersion": "5.1.1",
  . . .
}
```

Connect to an instance (server SDK 5)

If the instance you want to connect to is running a game build with server SDK version 5.x, use the following instructions to connect to the instance using Amazon EC2 Systems Manager (SSM). You can access remote instances that are running either Windows or Linux.

1. **Request access credentials for the instance.** When you have a compute name and fleet ID for the instance you want to connect to, call get-compute-access. If successful, Amazon GameLift returns a set of temporary credentials for accessing the instance. For example:

   Request

   ```
   aws gamelift get-compute-access \
   --compute-name i-11111111a222b333c \
   --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa
   --region us-west-2
   ```

   Response

   ```
   {
     "ComputeName": " i-11111111a222b333c ",
     "Credentials": {
       "AccessKeyId": " ASIAIOSFODNN7EXAMPLE ",
       "SecretAccessKey": " wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY ",
       "SessionToken": " AQoDYXdzEJr...<remainder of session token>"
     },
     "FleetArn": " arn:aws:gamelift:us-west-2::fleet/
   fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa ",
     "FleetId": " fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa "
   }
   ```

2. **Export the access credentials.** You can optionally export the credentials to environment variables and use them to configure the AWS CLI for the default user. For more details, see Environment variables to configure the AWS CLI in the AWS Command Line Interface User Guide.

   ```
   export AWS_ACCESS_KEY_ID=ASIAIOSFODNN7EXAMPLE
   export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   export AWS_SESSION_TOKEN=AQoDYXdzEJr...<remainder of session token>
   ```

3. **Connect to the fleet instance.** Start an SSM session with the instance you want to connect to. Include the AWS Region or location of the instance. For more information, see Starting a session (AWS CLI) in the *Amazon EC2 Systems Manager User Guide*. Use the credentials you acquired in Step 1. For example:

   ```
   aws ssm start-session \
   --target i-11111111a222b333c \
   ```

```
--region us-west-2
```

Connect to an instance (server SDK 4.x or earlier)

If the instance you want to connect to is running a game build with server SDK version 4 or earlier, use the following instructions. You can connect to instances that are running either Windows or Linux. Connect to a Windows instance using a remote desktop protocol (RDP) client. Connect to a Linux instance using an SSH client.

1. **Request access credentials for the instance.** When you have an instance ID, use the command get-instance-access to request access credentials. If successful, Amazon GameLift returns the instance's operating system, IP address, and a set of credentials (user name and secret key). The credentials format depends on the instance operating system. Use the following instructions to retrieve credentials for either RDP or SSH.

   - **For Windows instances** – To connect to a Windows instance, RDP requires a user name and password. The `get-instance-access` request returns these values as simple strings, so you can use the returned values as is. Example credentials:

     ```
     "Credentials": {
         "Secret": "aA1bBB2cCCd3EEE",
         "UserName": "gl-user-remote"
     }
     ```

   - **For Linux instances** – To connect to a Linux instance, SSH requires a user name and private key. Amazon GameLift issues RSA private keys and returns them as a single string, with the newline character (\n) indicating line breaks. To make the private key usable, take these steps: (1) convert the string to a `.pem` file, and (2) set permissions for the new file. Example credentials returned:

     ```
     "Credentials": {
         "Secret": "-----BEGIN RSA PRIVATE KEY-----
     nEXAMPLEKEYKCAQEAy7WZhaDsrA1W3mRlQtvhwyORRX8gnxgDAfRt/gx42kWXsT4rXE/b5CpSgie/
     \nvBoU7jLxx92pNHoFnByP+Dc21eyyz6CvjTmWA0JwfWiW5/akH7iO5dSrvC7dQkW2duV5QuUdE0QW
     \nZ/aNxMniGQE6XAgfwlnXVBwrerrQo+ZWQeqiUwwMkuEbLeJFLhMCvYURpUMSC1oehm449ilx9X1F
     \nG50TCFeOzfl8dqqCP6GzbPaIjiU19xX/azOR9V+tpUOzEL+wmXnZt3/nHPQ5xvD2OJH67km6SuPW
     \noPzev/D8V+x4+bHthfSjR9Y7DvQFjfBVwHXigBdtZcU2/wei8D/HYwIDAQABAoIBAGZ1kaEvnrqu
     \n/uler7vgIn5m7lN5LKw4hJLAIW6tUT/fzvtcHK0SkbQCQXuriHmQ2MQyJX/0kn2NfjLV/
     ufGxbL1\nmb5qwMGUnEpJaZD6QSSs3kICLwWUYUiGfc0uiSbmJoap/
     ```

```
GTLU0W5Mfcv36PaBUNy5p53V6G7hXb2\nbahyWyJNfjLe4M86yd2YK3V2CmK+X/
BOsShnJ36+hjrXPPWmV3N9zEmCdJjA+K15DYmhm/
tJWSD9\n81oGk9TopEp7CkIfatEATyyZiVqoRq6k64iuM9JkA3OzdXzMQexXVJ1TLZVEH0E7bhlY9d8O1ozR
\noQs/FiZNAx2iijCWyv0lpjE73+kCgYEA9mZtyhkHkFDpwrSM1APaL8oNAbbjwEy7Z5Mqfql
+lIp1\nYkriL0DbLXlvRAH+yHPRit2hHOjtUNZh4Axv+cpg09qbUI3+43eEy24B7G/Uh
+GTfbjsXsOxQx/x\np9otyVwc7hsQ5TA5PZb
+mvkJ5OBEKzet9XcKwONBYELGhnEPe7cCgYEA06Vgov6YHleHui9kHuws
\nayav0elc5zkxjF9nfHFJRry21R1trw2Vdpn+9g481URrpzWVOEihvm+xTtmaZlSp//lkq75XDwnU
\nWA8gkn6O3QE3fq2yN98BURsAKdJfJ5RL1HvGQvTe10HLYYXpJnEkHv+Unl2ajLivWUt5pbBrKbUC
\ngYBjbO+OZk0sCcpZ29sbzjYjpIddErySIyRX5gV2uNQwAjLdp9PfN295yQ+BxMBXiIycWVQiw0bH
\noMo7yykABY7Ozd5wQewBQ4AdSlWSX4nGDtsiFxWiI5sKuAAeOCbTosy1s8w8fxoJ5Tz1sdoxNeGs
\nArq6Wv/G16zQuAE9zK9vvwKBgF+09VI/1wJBirsDGz9whVWfFPrTkJNvJZzYt69qezxlsjgFKshy
\nWBhd4xHZtmCqpBPlAymEjr/TOlbxyARmXMnIOWIAnNXMGB4KGSyl1mzSVAoQ+fqR+cJ3d0dyPl1j
\njjb0Ed/NY8frlNDxAVHE8BSkdsx2f6ELEyBKJSRr9snRAoGAMrTwYneXzvTskF/S5Fyu0iOegLDa
\nNWUH38v/nDCgEpIXD5Hn3qAEcju1IjmbwlvtW+nY2jVhv7UGd8MjwUTNGItdb6nsYqM2asrnF3qS
\nVRkAKKKYeGjkpUfVTrW0YFjXkfcrR/V+QFL5OndHAKJXjW7a4ejJLncTzmZSpYzwApc=\n-----END
 RSA PRIVATE KEY-----",
    "UserName": "gl-user-remote"
}
```

When using the AWS CLI, you can automatically generate a `.pem` file by including the *--query* and *--output* parameters to your `get-instance-access` request.

To set permissions on the `.pem` file, run the following command:

```
$ chmod 400 MyPrivateKey.pem
```

2. **Open a port for the remote connection.** You can access instances in Amazon GameLift fleets through any port authorized in the fleet configuration. You can view a fleet's port settings using the command `describe-fleet-port-settings`.

   As a best practice, we recommend opening ports for remote access only when you need them and closing them when you're finished. You can't update port settings after creating a fleet but before it's active. If you get stuck, re-create the fleet with the port settings open.

   Use the command `update-fleet-port-settings` to add a port setting for the remote connection (such as 22 for SSH or 3389 for RDP). For the IP range value, specify the IP addresses for the devices you plan to use to connect (converted to CIDR format). Example:

```
$ AWS gamelift update-fleet-port-settings
    --fleet-id  "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa"
```

```
    --inbound-permission-authorizations
 "FromPort=22,ToPort=22,IpRange=54.186.139.221/32,Protocol=TCP"
```

The following example opens up port 3389 on a Windows fleet

```
$ AWS gamelift update-fleet-port-settings
--fleet-id  "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa"
    --inbound-permission-authorizations
 "FromPort=3389,ToPort=3389,IpRange=54.186.139.221/32,Protocol=TCP"
```

3. **Open a remote connection client.** Use Remote Desktop for Windows or SSH for Linux instances. Connect to the instance using the IP address, port setting, and access credentials.

   SSH example:

```
ssh -i MyPrivateKey.pem gl-user-remote@192.0.2.0
```

View files on remote instances

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, or disrupting game shutdown processes and causing errors in saved game data and logs.

Look for these resources on a hosting instance:

- **Game build files.** These files are the game build that you uploaded to Amazon GameLift. They include one or more game server executables, assets, and dependencies. Game build files are in a root directory called `game`:
  - On Windows: `c:\game`
  - On Linux: `/local/game`
- **Game log files.** Find the log files that your game server generates in the `game` root directory at whatever directory path you designated.
- **Amazon GameLift hosting resources.** The root directory `Whitewater` contains files used by the Amazon GameLift service to manage game hosting activity. Don't modify these files for any reason.

- **Runtime configuration.** Don't access runtime configuration for individual instances. To make changes to a runtime configuration property, update the fleet's runtime configuration (see the AWS SDK operation UpdateRuntimeConfiguration or the AWS CLI update-runtime-configuration).
- **Fleet data.** A JSON file contains information about the fleet that the instance belongs to, for use by server processes running on the instance. The JSON file is in the following location:
  - On Windows: `C:\GameMetadata\gamelift-metadata.json`
  - On Linux: `/local/gamemetadata/gamelift-metadata.json`
- **TLS certificates.** If the instance is on a fleet that has TLS certificate generation enabled, look for certificate files, including the certificate, certificate chain, private key, and root certificate in the following location:
  - On Windows: `c:\\GameMetadata\Certificates`
  - On Linux: `/local/gamemetadata/certificates/`

Scaling Amazon GameLift hosting capacity

Hosting capacity, measured in instances, represents the number of game sessions that Amazon GameLift can host concurrently and the number of concurrent players that those game sessions can accommodate. One of the most challenging tasks with game hosting is scaling capacity to meet player demand without wasting money on resources that you don't need. For more information, see Scaling fleet capacity.

Capacity is adjusted at the fleet location level. All fleets have at least one location: the fleet's home AWS Region. When viewing or scaling capacity, the information is listed by location, including the fleet's home Region and any additional remote locations.

You can manually set the number of instances to maintain, or you can set up auto scaling to dynamically adjust capacity as player demand changes. We recommend that you start by turning on the target-based auto scaling option. The goal of target-based auto scaling is to maintain enough hosting resources to accommodate current players plus a little extra to handle unexpected spikes in player demand. For most games, target-based auto scaling offers a highly effective scaling solution.

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

- Set minimum and maximum limits for capacity scaling

- Manually set capacity levels
- Use target-based auto scaling
- Manage rule-based auto scaling (advanced feature)
- Temporarily disable auto scaling

You can do most fleet scaling activities using the Amazon GameLift console. You can also use an AWS SDK or the AWS Command Line Interface (AWS CLI) with the Amazon GameLift service API.

To manage fleet capacity in the console

1. Open the Amazon GameLift console.
2. In the navigation pane, choose **Hosting**, **Fleets**.
3. On the **Fleets** page, choose the name of an active fleet to open the fleet's detail page.
4. Choose the **Scaling** tab. On this tab, you can:
   - View historical scaling metrics for the entire fleet.
   - View and update capacity settings for each fleet location, including scaling limits and current capacity settings.
   - Update target-based auto scaling, view rule-based auto scaling policies applied to the entire fleet, and suspend auto scaling activity for each location.

Topics

- Set Amazon GameLift capacity limits
- Manually set capacity for a Amazon GameLift fleet
- Auto-scale fleet capacity with Amazon GameLift

Set Amazon GameLift capacity limits

When scaling hosting capacity for a Amazon GameLift fleet location, either manually or by auto scaling, consider the location's scaling limits. All fleet locations have a minimum and maximum limit that define the allowed range for the location's capacity. By default, limits on fleet locations have a minimum of 0 instances and a maximum of 1 instance. Before you can scale a fleet location, adjust the limits.

If you're using auto scaling, the maximum limit allows Amazon GameLift to scale up a fleet location to meet player demand but prevents runaway hosting costs, such as during a DDOS attack. Set up an Amazon CloudWatch alarm to notify you when capacity approaches the maximum limit, so you can evaluate the situation and manually adjust as needed. (You can also create a billing alarm to monitor AWS costs.) The minimum limit is useful to maintain hosting availability, even when player demand is low.

You can set capacity limits for a fleet's locations in the Amazon GameLift console or by using the AWS Command Line Interface (AWS CLI).

To set capacity limits

Console

1. Open the Amazon GameLift console.
2. In the navigation pane, choose **Hosting**, **Fleets**.
3. On the **Fleets** page, choose the name of an active fleet to open the fleet's detail page.
4. On the **Scaling** tab, under **Scaling capacity**, select a fleet location, and then choose **Edit**.
5. In the **Edit scaling capacity** dialog box, set instance counts for **Min size**, **Desired instances**, and **Max size**.
6. Choose **Confirm**.

AWS CLI

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

   ```
   aws gamelift describe-fleet-location-capacity \
       --fleet-id <fleet identifier> \
       --location <location name>
   ```

2. **Update limit settings.** In a command line window, use the update-fleet-capacity command with the following parameters. You can adjust both instance limits and desired instance count with the same command.

```
--fleet-id <fleet identifier>
--location <location name>
--max-size <maximum capacity for scaling>
--min-size <minimum capacity for scaling>
--desired-instances <fleet capacity goal>
```

Example:

```
aws gamelift update-fleet-capacity \
    --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \
    --location us-west-2 \
    --max-size 10 \
    --min-size 1 \
    --desired-instances 10
```

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

Manually set capacity for a Amazon GameLift fleet

When you create a new fleet, Amazon GameLift automatically sets the desired instances to one instance in each fleet location. Then, Amazon GameLift deploys one new instance in each location. To change fleet capacity, you can add a target-based auto scaling policy, or you can manually set the number of instances that you want for a location. For more information, see Scaling fleet capacity.

Setting a fleet's capacity manually can be useful when you don't need auto scaling or when you need to hold capacity at a specified level. Manually setting capacity works only if you aren't using a target-based auto scaling policy. If you have a target-based auto scaling policy, it immediately resets the desired capacity based on its own scaling rules.

You can manually set capacity in the Amazon GameLift console or by using the AWS Command Line Interface (AWS CLI). The fleet's status must be active.

Suspend auto scaling

You can suspend all auto scaling activity for each fleet location. With auto scaling suspended, the desired number of instances in the fleet location remains the same unless manually changed. When you suspend auto scaling for a location, it affects the fleet's current policies and any policies that you may define in the future.

To manually set fleet capacity

Console

1. Open the Amazon GameLift console.
2. In the navigation pane, choose **Hosting**, **Fleets**.
3. On the **Fleets** page, choose the name of an active fleet to open the fleet's detail page.
4. On the **Scaling** tab, under **Suspended auto-scaling locations**, select each location that you want to suspend auto scaling for, and then choose **Suspend**.
5. Under **Scaling capacity**, select a location that you want to set manually, and then choose **Edit**.
6. In the **Edit scaling capacity** dialog box, set your preferred value for **Desired instances**, and then choose **Confirm**. This tells Amazon GameLift the number of instances to maintain in an active state, ready to host game sessions.

   Amazon GameLift responds to the changes by deploying additional instances or shutting down unneeded ones. As Amazon GameLift completes this process, the number of active instances in the location changes to match the updated desired instances value. This process may take a little time.

AWS CLI

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

   ```
   aws gamelift describe-fleet-location-capacity \
       --fleet-id <fleet identifier> \
       --location <location name>
   ```

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

```
--fleet-id <fleet identifier>
--location <location name>
--desired-instances <fleet capacity as an integer>
--max-size <maximum capacity>    [Optional]
--min-size <minimum capacity>    [Optional]
```

Example:

```
aws gamelift update-fleet-capacity \
    --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \
    --location us-west-2 \
    --desired-instances 5 \
    --max-size 10 \
    --min-size 1
```

If your request is successful, Amazon GameLift returns the fleet ID. If the new desired instances setting is outside the minimum and maximum limits, Amazon GameLift returns an error.

Auto-scale fleet capacity with Amazon GameLift

Use auto-scaling in Amazon GameLift to dynamically scale your fleet capacity in response to game server activity. As players arrive and start game sessions, auto scaling can add more instances; as player demand wanes, auto scaling can terminate unneeded instances. Auto scaling is an effective way to minimize your hosting resources and costs, while still providing a smooth, fast player experience.

To use auto scaling, you create scaling policies that tell Amazon GameLift when to scale up or down. There are two types of scaling policies: target-based and rule-based. The target-based approach—target tracking—is a complete solution. We recommend it as the simplest and most effective option. Rule-based scaling policies require you to define each aspect of the auto scaling decision-making process, which is useful for addressing specific issues. This solution works best as a supplement to target-based auto scaling.

You can manage target-based auto scaling using the Amazon GameLift console, the AWS Command Line Interface (AWS CLI), or an AWS SDK. You can manage rule-based auto scaling using the AWS CLI or an AWS SDK only, although you can view rule-based scaling policies in the console.

Topics

Target-based auto scaling

Target-based auto scaling for Amazon GameLift adjusts capacity levels based on the fleet metric `PercentAvailableGameSessions`. This metric represents the fleet's available buffer for sudden increases in player demand.

The primary reason for maintaining a capacity buffer is player wait time. When game session slots are ready and waiting, it takes seconds to get new players into game sessions. If no resources are available, players must wait for existing game sessions to end or for new resources to become available. It can take minutes to start up new instances and server processes.

When setting up target-based auto scaling, specify the size of the buffer that you want the fleet to maintain. Because `PercentAvailableGameSessions` measures the percentage of available resources, the actual buffer size is a percentage of the total fleet capacity. Amazon GameLift adds or removes instances to maintain the target buffer size. With a large buffer, you minimize wait time, but you also pay for extra resources that you may not use. If your players are more tolerant of wait times, you can lower costs by setting a small buffer.

To set target-based auto scaling

Console

1. Open the Amazon GameLift console.
2. In the navigation pane, choose **Hosting**, **Fleets**.
3. On the **Fleets** page, choose the name of an active fleet to open the fleet's detail page.
4. Choose the **Scaling** tab. This tab displays the fleet's historical scaling metrics and contains controls for adjusting current scaling settings.
5. Under **Scaling capacity**, check that the **Min size** and **Max size** limits are appropriate for the fleet. With auto scaling enabled, capacity adjusts between these two limits.

6. In **Target-based auto-scaling policy**, choose **Edit**.
7. In the **Edit target-based auto-scaling policy** dialog box, for **Percent available game sessions**, set the percentage that you want to maintain, and then choose **Confirm**. After you've confirmed the settings, Amazon GameLift adds a new target-based policy under **Target-based auto-scaling policy**.

AWS CLI

1. **Set capacity limits.** Set the limit values using the update-fleet-capacity command. For more information, see Set Amazon GameLift capacity limits.
2. **Create a new policy.** Open a command-line window and use the put-scaling-policy command with your policy's parameter settings. To update an existing policy, specify the policy's name and provide a complete version of the updated policy.

```
--fleet-id <unique fleet identifier>
--name "<unique policy name>"
--policy-type <target- or rule-based policy>
--metric-name <name of metric>
--target-configuration <buffer size>
```

Example:

```
aws gamelift put-scaling-policy \
    --fleet-id "fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa" \
    --name "My_Target_Policy_1" \
    --policy-type "TargetBased" \
    --metric-name "PercentAvailableGameSessions" \
    --target-configuration "TargetValue=5"
```

Auto scale with rule-based policies

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

A rule-based policy states the following: "If a fleet metric meets or crosses a threshold value for a certain length of time, then change the fleet's capacity by a specified amount." This topic describes the syntax used to construct a policy statement and provides help with creating and managing your rule-based policies.

Manage rule-based policies

Create, update, or delete rule-based policies using an AWS SDK or the AWS Command Line Interface (AWS CLI) with the Amazon GameLift service API. You can view all active policies in the Amazon GameLift console.

To temporarily stop all scaling policies for a fleet, use the AWS CLI command stop-fleet-actions.

To create or update a rule-based scaling policy (AWS CLI):

1. **Set capacity limits.** Set either or both limit values using the update-fleet-capacity command. For more information, see Set Amazon GameLift capacity limits.
2. **Create a new policy.** Open a command line window and use the put-scaling-policy command with your policy's parameter settings. To update an existing policy, specify the policy's name and provide a complete version of the updated policy.

```
--fleet-id <unique fleet identifier>
--name "<unique policy name>"
--policy-type <target- or rule-based policy>
--metric-name <name of metric>
--comparison-operator <comparison operator>
--threshold <threshold integer value>
--evaluation-periods <number of minutes>
--scaling-adjustment-type <adjustment type>
--scaling-adjustment <adjustment amount>
```

Example:

```
aws gamelift put-scaling-policy \
    --fleet-id fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa \
    --name "Scale up when AGS<50" \
    --policy-type RuleBased \
    --metric-name AvailableGameSessions \
    --comparison-operator LessThanThreshold \
    --threshold 50 \
    --evaluation-periods 10 \
```

```
    --scaling-adjustment-type ChangeInCapacity \
    --scaling-adjustment 1
```

To delete a rule-based scaling policy using the AWS CLI:

- Open a command line window and use the delete-scaling-policy command with the fleet ID and policy name.

  Example:

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

Syntax for auto scaling rules

To construct a rule-based scaling policy statement, specify six variables:

If *<metric name>* remains *<comparison operator>* *<threshold value>* for *<evaluation period>*, then change fleet capacity using *<adjustment type>* to/by *<adjustment value>*.

For example, this policy statement starts a scale-up event whenever a fleet's extra capacity is less than what's needed to handle 50 new game sessions:

If `AvailableGameSessions` remains at `less than 50` for `10 minutes`, then change fleet capacity using `ChangeInCapacity` by `1 instances`.

Metric name

To start a scaling event, link an auto scaling policy to one of the following fleet-specific metrics. For complete metric descriptions, see Amazon GameLift metrics for fleets.

- Activating game sessions
- Active game sessions
- Available game sessions
- Percent available game sessions
- Active instances
- Available player sessions

- Current player sessions
- Idle instances
- Percent idle instances

If the fleet is in a game session queue, you can use the following metrics:

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

Comparison operator

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

Threshold value

When the specified metric value meets or crosses the threshold value, it starts a scaling event. This value is always a positive integer.

Evaluation period

The metric must meet or cross the threshold value for the full length of the evaluation period before starting a scaling event. The evaluation period length is consecutive; if the metric retreats from the threshold, the evaluation period starts over again.

Adjustment type and value

This set of variables works together to specify how Amazon GameLift should adjust the fleet's capacity when a scaling event starts. Choose from three possible adjustment types:

- **Change in capacity** – Increase or decrease the current capacity by a specified number of instances. Set the adjustment value to the number of instances to add or remove from the fleet. Positive values add instances, while negative values remove instances. For example, a value of "-10" scales down the fleet by 10 instances, regardless of the fleet's total size.
- **Percent change in capacity** – Increase or decrease the current capacity by a specified percentage. Set the adjustment value to the percentage that you want to increase or decrease the fleet capacity by. Positive values add instances, while negative values remove instances. For example, for a fleet with 50 instances, a percentage change of "20" adds 10 instances to the fleet.

- **Exact capacity** – Increase or decrease the current capacity to a specific value. Set the adjustment value to the exact number of instances that you want to maintain in the fleet.

Tips for rule-based auto scaling

The following suggestions can help you get the most out of auto scaling with rule-based policies.

Use multiple policies

You can have multiple auto scaling policies for a fleet at the same time. The most common scenario is to have a target-based policy manage most scaling needs and use rule-based policies to handle edge cases. There are no limits on using multiple policies.

With multiple policies, each policy behaves independently. There is no way to control the sequence of scaling events. For example, if you have multiple policies driving scaling up, it's possible that player activity could start multiple scaling events simultaneously. Avoid policies that start each other. For example, you could create an infinite loop if you create scale-up and scale-down policies that set capacity beyond the threshold of each other.

Set maximum and minimum capacity

Each fleet has a maximum and minimum capacity limit. This feature is important when using auto scaling. Auto scaling never sets capacity to a value outside of this range. By default, newly created fleets have a minimum of 0 and a maximum of 1. For your auto scaling policy to affect capacity as intended, increase the maximum value.

Fleet capacity is also constrained by limits on the fleet's instance type and by service quotas in your AWS account. You can't set a minimum and maximum outside these limits and account quotas.

Track metrics after a change in capacity

After changing capacity in response to an auto scaling policy, Amazon GameLift waits 10 minutes before responding to triggers from the same policy. This wait gives Amazon GameLift time to add the new instances, launch the game servers, connect players, and start collecting data from the new instances. During this time, Amazon GameLift evaluates the policy against the metric and tracks the policy's evaluation period, which restarts after a scaling event occurs. This means that a scaling policy could start another scaling event immediately after the wait time is over.

There is no wait time between scaling events that different auto scaling policies start.

Setting up Amazon GameLift queues for game session placement

A game session queue is the primary mechanism for processing new game session requests and locating available game servers to host them. Queues offer significant benefits for game developers and players. These include:

- **Queues provide best possible placement.** When processing game session placement requests, a queue uses Amazon GameLift algorithms to prioritize queue locations based on a set of defined preferences.
- **Host games on lower-priced Spot fleets.** Use queues to optimize use of AWS Spot fleets, which offer significantly lower hosting costs. By default, queues always try to place new game sessions in Spot fleets.
- **Place new games faster during high demand.** Queues use multiple possible locations for placements. This means that there is always fallback capacity if the preferred placement location is unavailable.
- **Make game availability more resilient.** Outages can happen. With a multi-location queue, a slowdown or outage doesn't have to affect player access to your game.
- **Use extra fleet capacity more efficiently.** To handle unexpected surges in player demand, queues provide quick access to extra hosting capacity. The fleet locations in a queue provide back-up capacity for each other. Locations scale up or down based on player demand.
- **Get metrics on game session placements and queue performance.** Amazon GameLift emits queue metrics, including statistics on placement successes and failures, the number of requests in the queue, and average time that requests spend in the queue. You can view these metrics in the Amazon GameLift console or in CloudWatch.

To get started with queues, see Design a game session queue.

Design a game session queue

This topic describes how to design a queue that delivers a player experience with minimal latency and that efficiently uses hosting resources. For more information about game session queues and how they work, see Setting up Amazon GameLift queues for game session placement.

These Amazon GameLift features require queues:

- Matchmaking with FlexMatch
- Use Spot Instances with Amazon GameLift

Define your queue's scope

Your game's player population might have groups of players who shouldn't play together. For example, if you publish your game in two languages each language should have it's own game servers.

To set up game session placement for your player population, create a separate queue for each player segment. Scope each queue to place players into the correct game servers. Some common ways to scope queues include:

- **By geographic locations.** When deploying your game servers in multiple geographic areas, you might build queues for players in each location to reduce player latency.
- **By build or script variations.** If you have more than one variation of your game server, you might be supporting player groups that can't play in the same game sessions. For example, game server builds or scripts might support different languages or device types.
- **By event types.** You might create a special queue to manage games for participants in tournaments or other special events.

Create a player latency policy

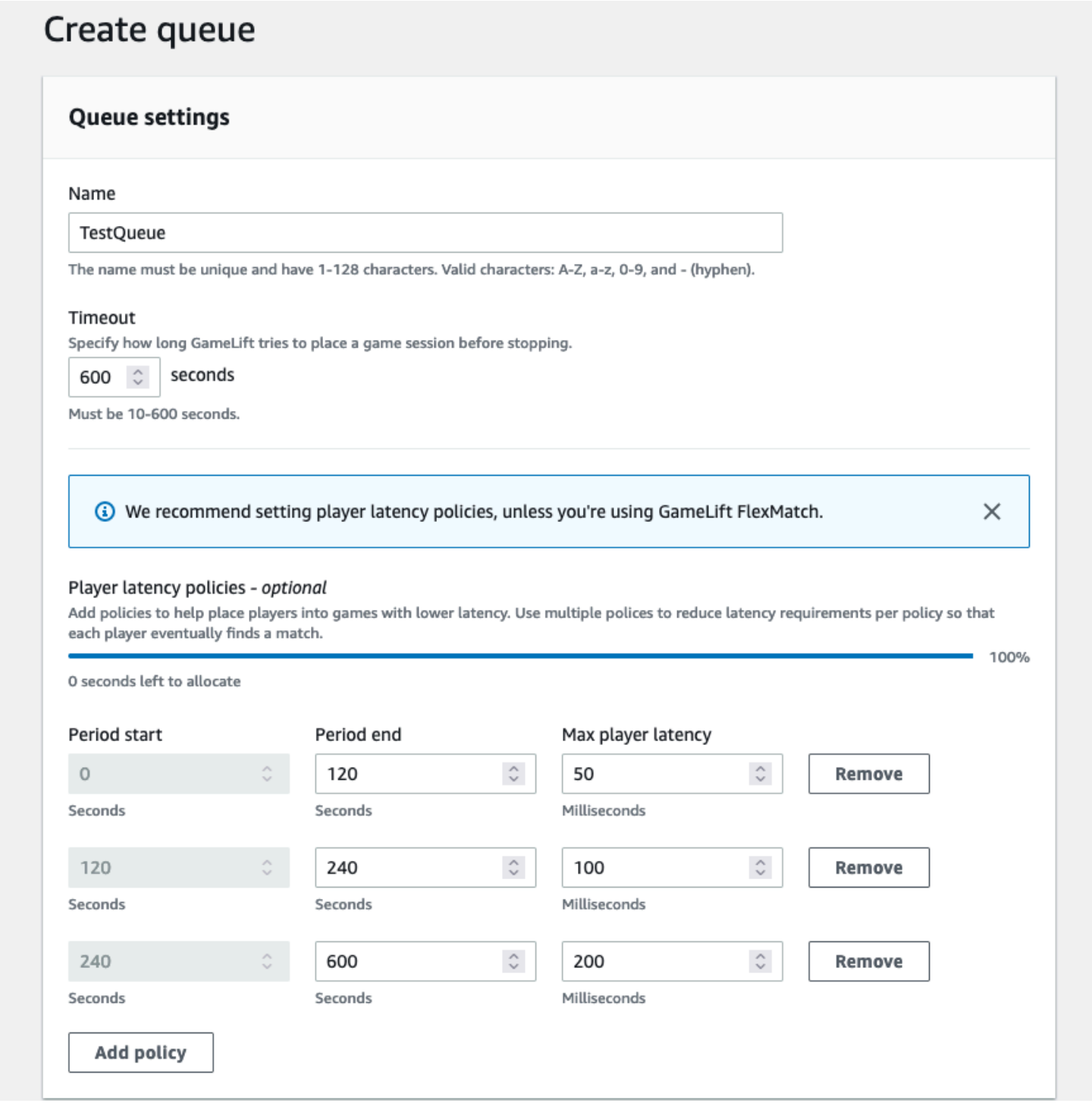
If your placement requests include player latency data, the algorithm finds game sessions in locations with the lowest average latency for all players. Placing game sessions based on average player latency prevents Amazon GameLift from placing most players in games with high latency. However, Amazon GameLift still places players with extreme latency. To accommodate these players, create player latency policies.

A player latency policy prevents Amazon GameLift from placing a requested game session anywhere that players in the request would experience latency over the maximum value. Player latency policies can also prevent Amazon GameLift from matching game session requests with higher latency players.

> **Tip**
>
> To manage latency specific rules, such as requiring similar latency across all players in a group, you can use Amazon GameLift FlexMatch to create latency-based matchmaking rules.

For example, consider this queue with a 5-minute timeout and the following player latency policies:

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



Build a multi-location queue

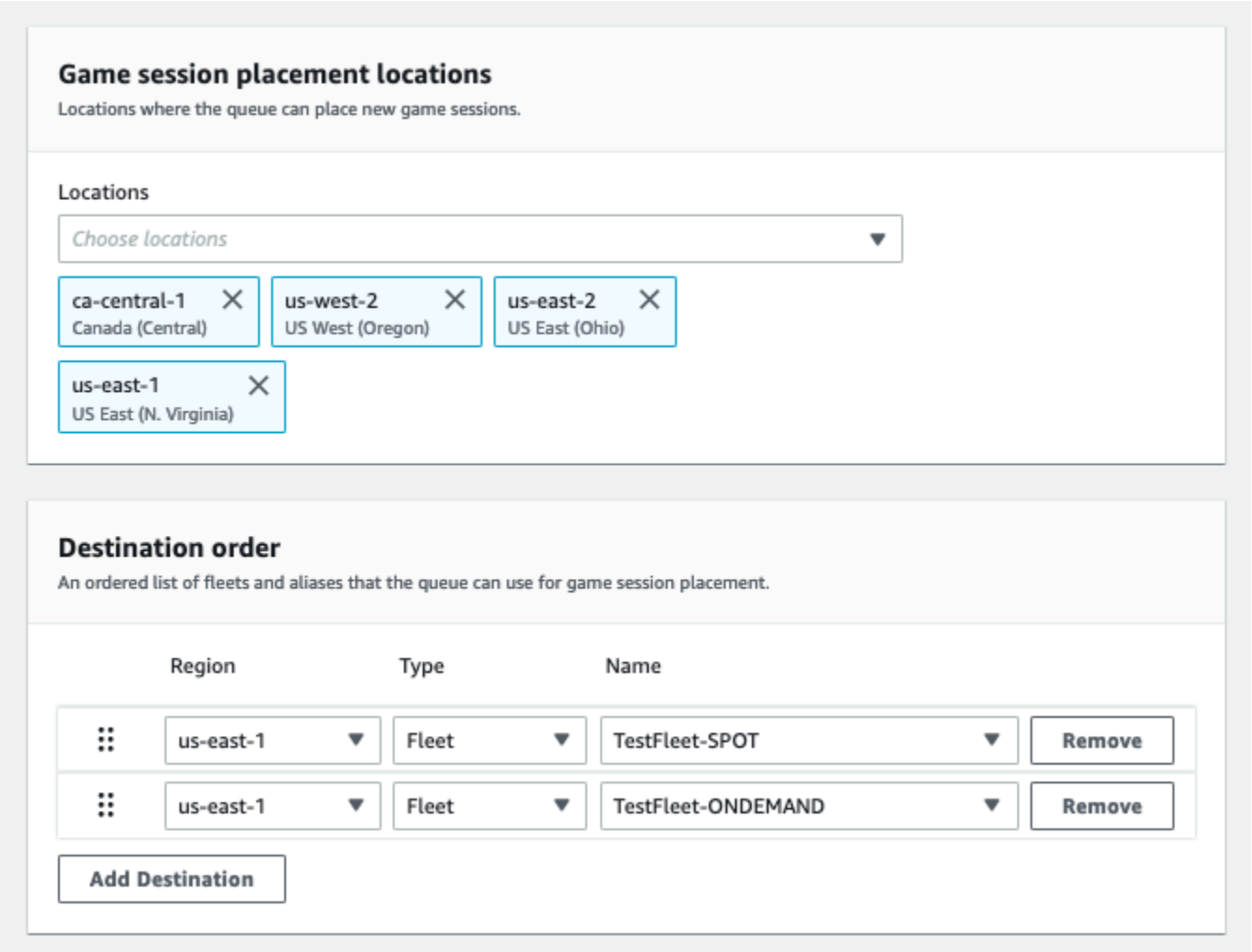
We recommend a multi-location design for all queues. This design can improve placement speed and hosting resiliency. A multi-location design is required to use player latency data to put players into game sessions with minimal latency. If you're building multi-location queues that use Spot Instance fleets, follow the instructions in Tutorial: Set up a game session queue for Spot Instances.

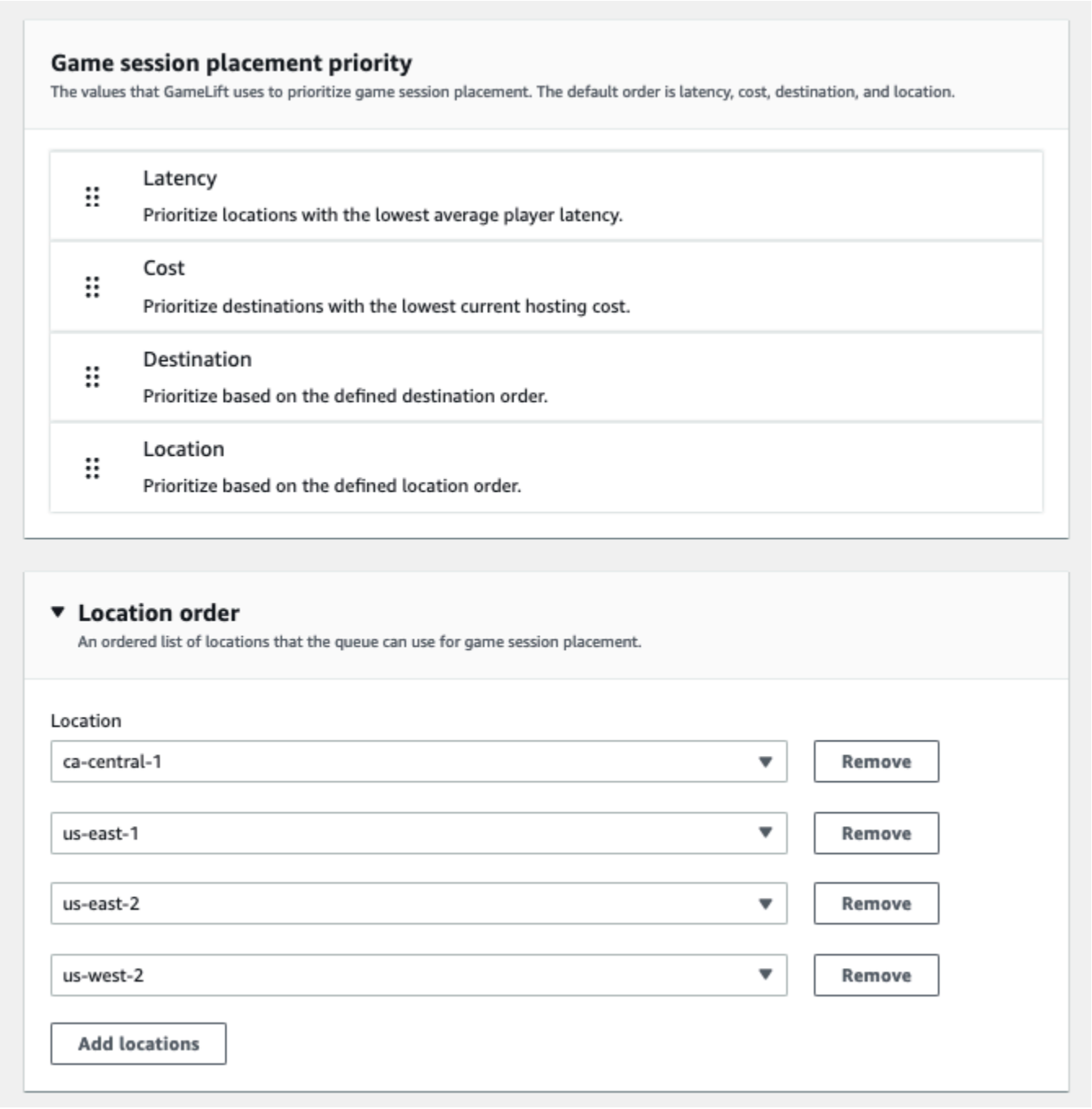
One way to create a multi-location queue is to add a multi-location fleet to a queue. That way, the queue can place game sessions in any of the fleet's locations. You can also add other fleets with different configurations or home locations for redundancy. If you're using a multi-location Spot Instance fleet, follow best practices and include an On-Demand Instance fleet with the same locations.

The following example outlines the process of designing a basic multi-location queue. In this example, we use two fleets: one Spot Instance fleet and one On-Demand Instance fleet. Each fleet has the following AWS Regions for placement locations: `us-east-1`, `us-east-2`, `ca-central-1`, and `us-west-2`.

To create a basic multi-location queue with multi-location fleets

1. Choose a location to create the queue in. You can minimize request latency by placing the queue in a location near where you deployed the client service. In this example, we create the queue in `us-east-1`.
2. Create a new queue and add your multi-location fleets as queue destinations. The destination order determines how Amazon GameLift places game sessions. In this example, we list the Spot Instance fleet first and the On-Demand Instance fleet second.
3. Define the queue's game session placement priority order. This order determines where the queue searches first for an available game server. In this example, we use the default priority order.
4. Define the location order. If you don't define the location order, Amazon GameLift uses the locations in alphabetical order.


Game session placement locations
Locations where the queue can place new game sessions.
Locations
Choose locations
ca-central-1
Canada (Central)
us-west-2
US West (Oregon)
us-east-2
US East (Ohio)
us-east-1
US East (N. Virginia)
Destination order
An ordered list of fleets and aliases that the queue can use for game session placement.
Region
Type
Name
us-east-1
Fleet
TestFleet-SPOT
Remove
us-east-1
Fleet
TestFleet-ONDEMAND
Remove
Add Destination



Prioritize game session placement

Amazon GameLift uses the FleetIQ algorithm to determine where to place a new game session based on an ordered set of criteria. You can use the default priority order, or you can customize the order.

Default priority order

For placement requests that include player latency data, FleetIQ prioritizes the game session placement criteria in the following default order:

1. **Latency** – Lowest average latency for all players in the request.
2. **Cost** – Lowest hosting cost, if latency is equal in multiple locations. Hosting cost is primarily based on a combination of the instance type and location.
3. **Destination** – Destination order, if latency and cost are equal in multiple locations. FleetIQ prioritizes destinations based on the order listed in the queue configuration.
4. **Location** – Location order, if latency, cost, and destination are equal in multiple locations. FleetIQ prioritizes locations based on the order listed in the queue configuration.

Custom priority order

To customize a queue's priority order in the Amazon GameLift console, drag the priority value to the position that you want it in. To customize a queue's priority order using the AWS Command Line Interface (AWS CLI), use the **create-game-session-queue** command with the **--priority-configuration** option. You can use this command to create a new queue or to update an existing queue.

The FleetIQ algorithm appends any criteria not explicitly mentioned to the end of your list, based on the default order. If you include the location criterion in your priority configuration, you must also provide an ordered list of locations.

Design multiple queues as needed

Depending on your game and players, you might want to create more than one game session queue. When your game client service requests a new game session, it specifies which game session queue to use. To help you determine whether to use multiple queues, consider:

- Variations of your game server. You can create a separate queue for each variation of your game server. All fleets in a queue must deploy compatible game servers. This is because players who use the queue to join games must be able to play on any of the queue's game servers.
- Different player groups. You can customize how Amazon GameLift places game sessions based on player group. For example, you might need queues customized for certain game modes that require a special instance type or runtime configuration. Or, you might want a special queue to manage placements for a tournament or other event.
- Game session queue metrics. You can set up queues based on how you want to collect game session placement metrics. For more information, see Amazon GameLift metrics for queues.

Evaluate queue metrics

Use metrics to evaluate how well your queues are performing. You can view metrics related to queues in the Amazon GameLift console or in Amazon CloudWatch. For a list and descriptions of queue metrics, see Amazon GameLift metrics for queues.

Queue metrics can provide insight about the following:

- **Overall queue performance** – Queue metrics indicate how successfully a queue responds to placement requests. These metrics can also help you identify when and why placements fail. For queues with manually scaled fleets, the `AverageWaitTime` and `QueueDepth` metrics can indicate when you should adjust capacity for a queue.
- **FleetIQ algorithm performance** – For placement requests using the FleetIQ algorithm, metrics show how often the algorithm finds ideal game session placement. The placement may prioritize using resources with the lowest player latency or resources with the lowest cost. There are also error metrics that identify common reasons why Amazon GameLift can't find an ideal placement. For more information about metrics, see Monitor Amazon GameLift with Amazon CloudWatch.
- **Location specific placements** – For multi-location queues, metrics show successful placements by location. For queues that use the FleetIQ algorithm, this data provides useful insight into where player activity occurs.

When evaluating metrics for FleetIQ algorithm performance, consider the following tips:

- To track the queue's rate of finding an ideal placement, use the `PlacementsSucceeded` metric in combination with the FleetIQ metrics for lowest latency and lowest price.
- To boost a queue's rate of finding an ideal placement, review the following error metrics:
  - If the `FirstChoiceOutOfCapacity` is high, adjust capacity scaling for the queue's fleets.
  - If the `FirstChoiceNotViable` error metric is high, look at your Spot Instance fleets. Spot Instance fleets are considered not viable when the interruption rate for a particular instance type is too high. To resolve this issue, change the queue to use Spot Instance fleets with different instance types. We recommend that you include Spot Instance fleets with different instance types in each location.

Best practices for Amazon 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:

- **Add fleets in locations that cover your players.** You can add fleets and aliases in any available location. Location is important if you're making placements based on reported player latency.
- **Use aliases for all fleets.** Assign an alias to each fleet in a queue, and use the alias names when setting destinations in your queue.
- **Use the same or a similar game build or script for all fleets.** The queue might put players into game sessions on any fleet in the queue. Players must be able to play in any game session on any fleet.
- **Create fleets in at least two locations.** By having game servers hosted in at least one other location, you mitigate the impact of Regional outages on your players. You can keep your backup fleets scaled down, and use auto scaling to increase capacity if usage increases.
- **Prioritize your game session placement.** A queue prioritizes placement choices based on several elements, including destination list order.
- **Create your queue in the same location as your client service.** By putting your queue in a location near your client service, you can minimize communication latency.
- **Use fleets with multiple locations.** Use the queue filter configuration to prevent the queue from placing game sessions in specified locations. You can use at least two multi-location fleets with different home locations to mitigate the impact of game placements during a Regional outage.
- **Use the same TLS certificate setting for all fleets.** Game clients that connect to game sessions in your fleets must have compatible communication protocols.

Best practices for queues with Spot fleets

If your queue includes Spot fleets, set up a resilient queue. This takes advantage of cost savings with Spot fleets while minimizing the effect of game session interruptions. For help with correctly building fleets and game session queues for use with Spot fleets, see Tutorial: Set up a game session queue for Spot Instances. For more information about Spot instances, see Use Spot Instances with Amazon GameLift.

In addition to the general best practices in the previous section, consider these Spot-specific best practices:

- **Create at least one On-Demand fleet in each location.** On-Demand fleets provide backup 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 location.** If one Spot Instance type becomes temporarily unavailable, the interruption affects only one Spot fleet in the location. 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 Amazon GameLift console to view historical pricing data for instance types.

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.

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.

When updating the queue destination in a queue, there is a short transition period (up to 30 seconds) during which game sessions placed on the queue destinations may still end up on the old fleet.

Console

1. In the Amazon GameLift console, in the navigation page, choose **Queues**.
2. On the **Queues** page, choose **Create queue**.
3. On the **Create queue** page, under **Queue settings** do the following:

a. For **Name**, enter a queue name.

b. For **Timeout**, enter long you want Amazon GameLift to try to place a game session before stopping. Amazon GameLift searches for available resources on any fleet until the request times out.

c. (Optional) For **Player latency policies**, enter how long Amazon GameLift should look for resources within the defined maximum latency. Add additional policies to gradual relax the maximum latency. To add additional policies, choose **Add policy**.

4. Under **Game session placement locations**, select locations to include in the queue. By default **All locations** are included. All fleets in the queue must have the same certificate configuration. All fleets should be running game builds that are compatible with the game clients using the queue.

5. Under **Destination order**, add one or more destinations to the queue.

a. Choose **Add destination**.

b. Select the **Location** that the destination is in.

c. Select the type for your destination.

d. From the resulting list of fleet or alias names, select the one you want to add.

e. If you have multiple destinations, set the default order by dragging the six dots icon to the left of the destination. Amazon GameLift uses this order when searching destinations for available resources to place a new game session.

6. For **Game session placement priority**, add and drag the **Latency**, **Cost**, **Destination**, and **Location** values to define how Amazon GameLift prioritizes fleets in your queue. For more information about prioritizing fleets, see Prioritize game session placement.

7. Add locations to your **Location order** and drag them to the priority that the queue should use. If **Location** is the last priority for game session placement, Amazon GameLift uses it as a tiebreaker.

8. (Optional) Under **Event notification settings** do the following:

a. Select or create an SNS topic to receive placement-related event notifications. For more information about event notifications, see Set up event notification for game session placement.

b. Add **Custom event data** to append to events created by this queue.

9. (Optional) Add **Tags**. For more information about tagging, see Tagging AWS resources.

10. Choose **Create**.

AWS CLI

Example Create a queue

The following example creates a game session queue with these configurations:

- A five minute timeout
- Two fleet destinations
- Filters to only allow locations in the `us-east-1`, `us-east-2`. `us-west-2`, and `ca-central-1`
- Prioritizes destinations based on cost and then locations in the defined order.

```
aws gamelift create-game-session-queue \
    --name "sample-test-queue" \
    --timeout-in-seconds 300 \
    --destinations DestinationArn="arn:aws:gamelift:us-east-1:111122223333:fleet/
fleet-772266ba-8c82-4a6e-b620-a74a62a93ff8" DestinationArn="arn:aws:gamelift:us-
east-1:111122223333:fleet/fleet-33f28fb6-aa8b-4867-85b4-ceb217bf5994" \
    --filter-configuration "AllowedLocations=us-east-1, ca-central-1, us-east-2, us-
west-2" \
    --priority-configuration
 PriorityOrder="LOCATION","DESTINATION",LocationOrder="us-east-1","us-east-2","ca-
central-1","us-west-2" \
    --notification-target "arn:aws:sns:us-east-1:111122223333:gamelift-test.fifo"
```

> **Note**
>
> You can get fleet and alias ARN values by calling either describe-fleet-attributes or describe-alias with the fleet or alias ID.

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

Example Create a queue with player latency policies

The following example creates a game session queue with these configurations:

- A ten minute timeout
- Three fleet destinations
- A set of player latency policies

```
aws gamelift create-game-session-queue \
    --name "matchmaker-queue" \
    --timeout-in-seconds 600 \
    --destinations DestinationArn=arn:aws:gamelift:us-east-1::alias/alias-a1234567-
b8c9-0d1e-2fa3-b45c6d7e8910 \
                DestinationArn=arn:aws:gamelift:us-west-2::alias/alias-b0234567-
c8d9-0e1f-2ab3-c45d6e7f8901 \
                DestinationArn=arn:aws:gamelift:us-west-2::fleet/fleet-f1234567-
b8c9-0d1e-2fa3-b45c6d7e8912 \
    --player-latency-policies
 "MaximumIndividualPlayerLatencyMilliseconds=50,PolicyDurationSeconds=120" \

 "MaximumIndividualPlayerLatencyMilliseconds=100,PolicyDurationSeconds=120" \
                "MaximumIndividualPlayerLatencyMilliseconds=150" \
```

If the `create-game-session-queue` request is successful, Amazon GameLift returns a GameSessionQueue object with the new queue configuration.

Set up event notification for game session placement

You can use event notifications to monitor the status of individual placement requests. We recommend setting up event notifications for all games with high-volume placement activity.

There are two options for setting up event notifications.

- Have Amazon GameLift publish event notifications to an Amazon Simple Notification Service (Amazon SNS) topic using a queue.
- Use automatically published Amazon EventBridge events and its suite of tools for managing events.

For a list of game session placement events emitted by Amazon GameLift, see Game session placement events.

Set up an SNS topic

For Amazon GameLift to publish all events generated by a game session queue to a topic, set the notification target field to a topic.

To set up an SNS topic for Amazon GameLift event notification

1. Sign in to the AWS Management Console and open the Amazon SNS console at https://console.aws.amazon.com/sns/v3/home.
2. From the SNS **Topics** page, choose **Create topic** and follow the instructions to create your topic.
3. Under **Access policy**, do the following:

   a. Choose the **Advanced** method.

   b. Add the following bolded section of the JSON object to the existing policy.

```
{
  "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"
      ],
      "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"
        }
      }
    }
  ]
}
```

c. (Optional) Add additional access control to the topic by adding conditions to the resource policy.

4. Choose **Create topic**.
5. After you've created your SNS topic, add it to queues during queue creation, or edit an existing queue to add it.

Set up an SNS topic with server-side encryption

With server-side encryption (SSE), you can store sensitive data in encrypted topics. SSE protects the contents of messages in Amazon SNS topics using keys that are managed in AWS Key Management Service (AWS KMS). For more information about server-side encryption with Amazon SNS, see Encryption at rest in the *Amazon Simple Notification Service Developer Guide*.

To set up an SNS topic with server-side encryption, review the following topics:

- Creating key in the *AWS Key Management Service Developer Guide*
- Enabling SSE for a topic in the *Amazon Simple Notification Service Developer Guide*

When creating your KMS key, use the following KMS key policy:

```
{
    "Effect": "Allow",
    "Principal": {
        "Service": "gamelift.amazonaws.com"
     },
    "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": "*",
    "Condition": {
        "ArnLike": {
            "aws:SourceArn":
 "arn:aws:gamelift:your_region:your_account:gamesessionqueue/your_queue_name"
        },
        "StringEquals": {
            "kms:EncryptionContext:aws:sns:topicArn":
 "arn:aws:sns:your_region:your_account:your_sns_topic_name"
        }
    }
}
```

Set up EventBridge

Amazon GameLift automatically posts all game session placement events to EventBridge. With EventBridge you can set up rules to have events routed to targets for processing. For example, you can set a rule to route the event `PlacementFulfilled` to an AWS Lambda function that handles tasks that precede connecting to a game session. For more information about EventBridge, see What is Amazon EventBridge? in the *Amazon EventBridge User Guide*.

The following are some examples of EventBridge rules to use with Amazon GameLift queues:

Matches events from all Amazon GameLift queues

```
{
    "source": [
        "aws.gamelift"
    ],
    "detail-type": [
        "GameLift Queue Placement Event"
    ]
}
```

Matches events from a specific queue

```
{
    "source": [
        "aws.gamelift"
    ],
    "detail-type": [
        "GameLift Queue Placement Event"
    ],
    "resources": [
        "arn:aws:gamelift:your_region:your_account:gamesessionqueue/your_queue_name"
    ]
}
```

Tutorial: Set up a game session queue for Spot Instances

Introduction

This tutorial describes how to set up game session placement for games deployed on low-cost Spot fleets. Spot fleets require additional steps to maintain continual game server availability for your players.

Intended audience

This tutorial is for game developers who want to use Spot fleets to host custom game servers or Realtime Servers.

What you'll learn

- Define the group of players who your game session queue serves.
- 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 Amazon GameLift places game sessions.
- Add player latency policies to help minimize latency issues.

Prerequisites

Before creating fleets and queues for game session placement, complete the following tasks:

- Review How Amazon GameLift works.
- Integrate you game server with Amazon GameLift.
- Upload your game server build or Realtime script to Amazon GameLift.

- Plan your fleet configuration.

Step 1: Define the scope of your queue

In this tutorial, we design a queue for a game that has one game server build variation. At launch, we're releasing the game in two locations: Asia Pacific (Seoul) and Asia Pacific (Singapore). Because these locations are close to each other, latency isn't an issue for our players.

For this example, there's one player segment, which means we create one queue. In the future, when we release the game in North America, we can create a second queue that's scoped for North American players.

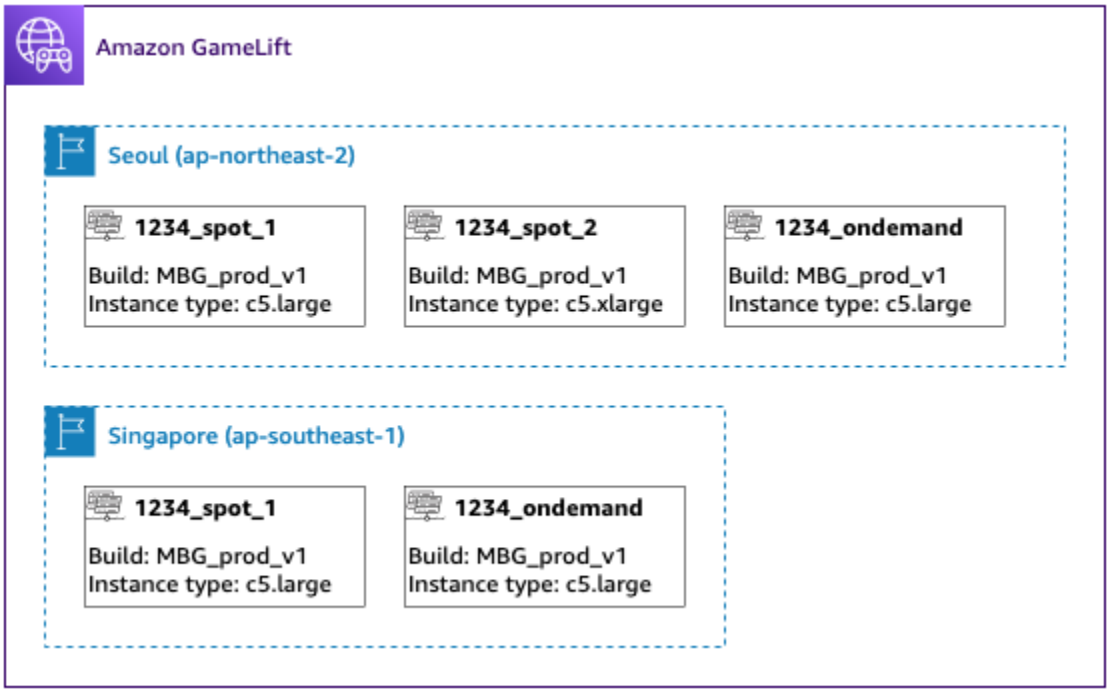
For more information, see Define your queue's scope.

Step 2: Create Spot fleet infrastructure

Create fleets in locations and with game server builds or scripts that fit the scope that you defined in Step 1: Define the scope of your queue.

In this tutorial, we create a two location infrastructure with at least one Spot fleet and one On-Demand fleet in each location. Every fleet deploys the same game server build. In addition, we anticipate that player traffic will be heavier in the Seoul location, so we add more Spot fleets there.

The following diagram shows the example Spot fleet infrastructure, with 3 fleets in the ap-northeast-2 (Seoul) location and 2 fleets in the ap-southeast-1 (Singapore) location. All instances in both fleets are using the build MBG_prod_V1. The fleet in ap-northeast-2 contains the following fleet configurations: fleet 1234_spot_1 with an instance type of c5.large, fleet 1234_spot_2 with an instance type of c5.xlarge, and fleet 1234_ondemand with an instance type of c5.large. The fleet in ap-southeast-1 contains the following fleet configurations: fleet 1234_spot_1 with an instance type of c5.large and fleet 1234_ondemand with an instance type of c5.large.

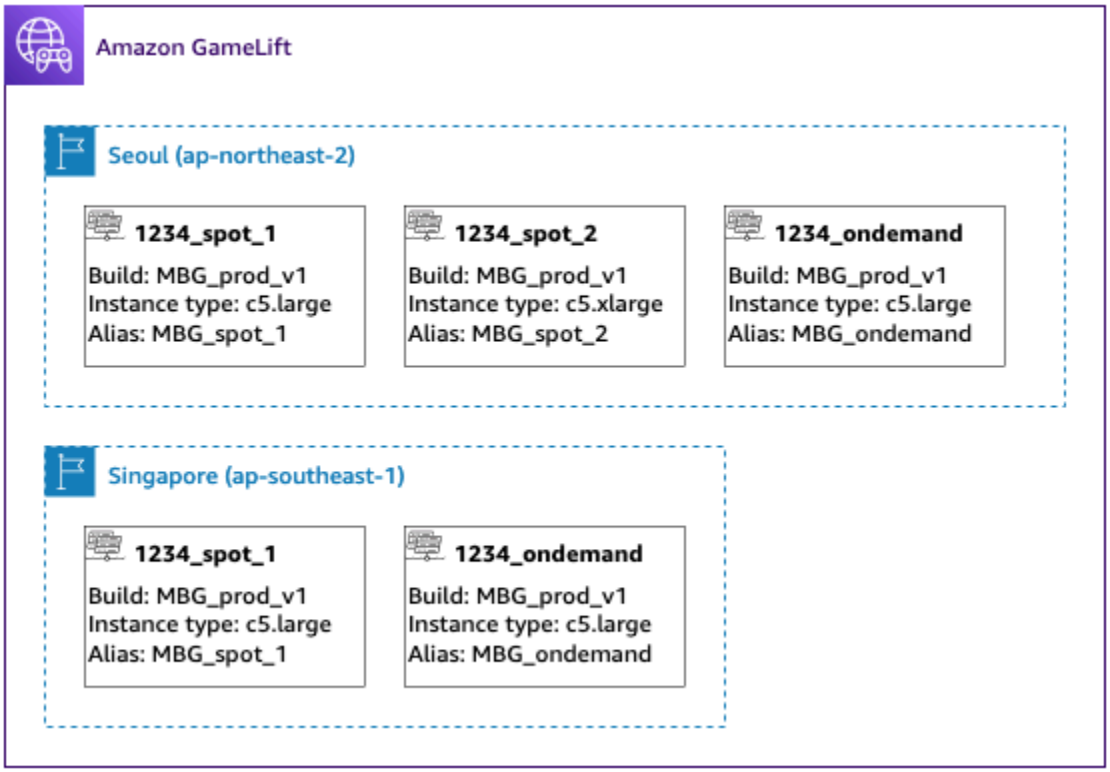


Step 3: Assign aliases for each fleet

Create a new alias for each fleet in your infrastructure. Aliases abstract fleet identities, making periodic fleet replacement efficient. For more information about creating aliases, see Add an alias to a Amazon GameLift fleet.

Our fleet infrastructure has five fleets, so we create five aliases using the routing strategy. We need three aliases in the Asia Pacific (Seoul) location, and two aliases in the Asia Pacific (Singapore) location.

The following diagram shows the Spot fleet infrastructure described in step two with aliases added to each fleet. Fleet 1234_spot_1 has the alias MBG_spot_1, Fleet 1234_spot_2 has the alias MBG_spot_2, and fleet 1234_ondemand has the alias MBG_ondemand.



For more information, see Build a multi-location queue.

Step 4: Create a queue with destinations

Create the game session queue and add your fleet destinations. For more information about creating a queue, see Create a game session queue.

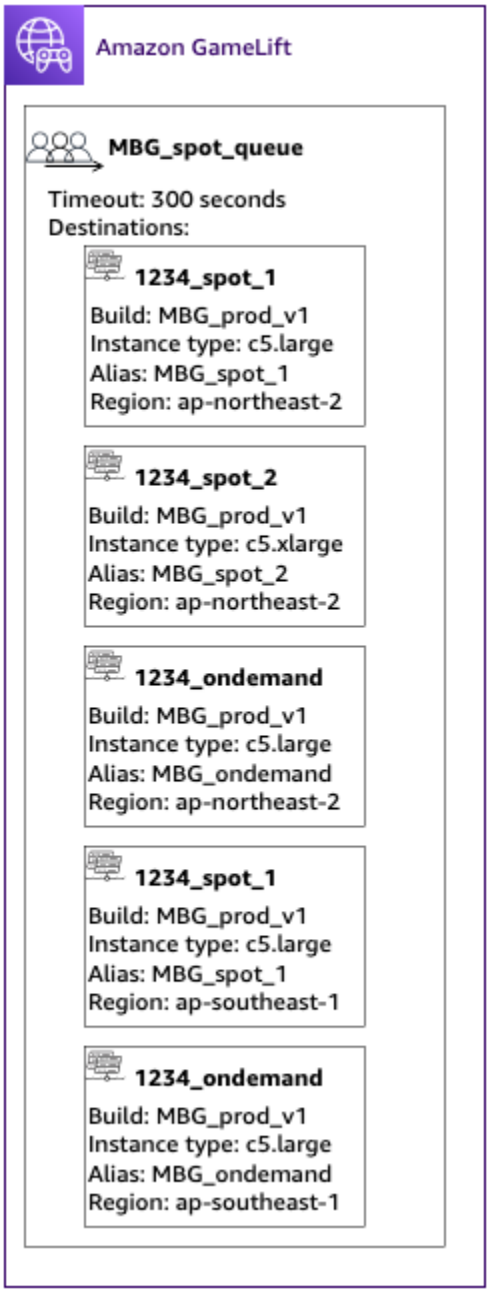
When creating your queue:

- Set the default timeout to 10 minutes. Later, you can test how the queue timeout affects your players' wait times for getting into games.
- Skip the section on player latency policies for now. We'll cover this in the next step.
- Prioritize the fleets in your queue. When working with Spot fleets, we recommend either of the following approaches:
  - If your infrastructure uses a primary location with fleets in a second location for backup, prioritize fleets first by location then by fleet type.
  - If your infrastructure uses multiple locations equally, prioritize fleets by fleet type, placing Spot fleets at the top of the queue.

For this tutorial, we create a new queue with the name **MBG_spot_queue**, and add the aliases of all five of our fleets. We then prioritize placements first by location and second by fleet type.

Based on this configuration, this queue always attempts to place new game sessions into a Spot fleet in Seoul. When those fleets are full, the queue uses available capacity on the Seoul On-Demand fleet as a backup. If all three Seoul fleets are unavailable, Amazon GameLift places game sessions on the Singapore fleets.

The following diagram shows a queue with a timeout of 300 seconds and prioritized destinations. The destinations are in the following order: 1234_spot_1 in ap-northeast-2, 1234_spot_2 in ap-northeast-2, 1234_ondemand in ap-northeast-2, 1234_spot_1 in ap-southeast-1, and 1234_ondemand in ap-southeast-1.



Step 5: Add latency limits to the queue

Our game includes latency information in game session placement requests. We also have a player party feature that creates a game session for a group of players. We can have players wait a little longer to get into games with the ideal gameplay experience. Our game tests show the following observations:

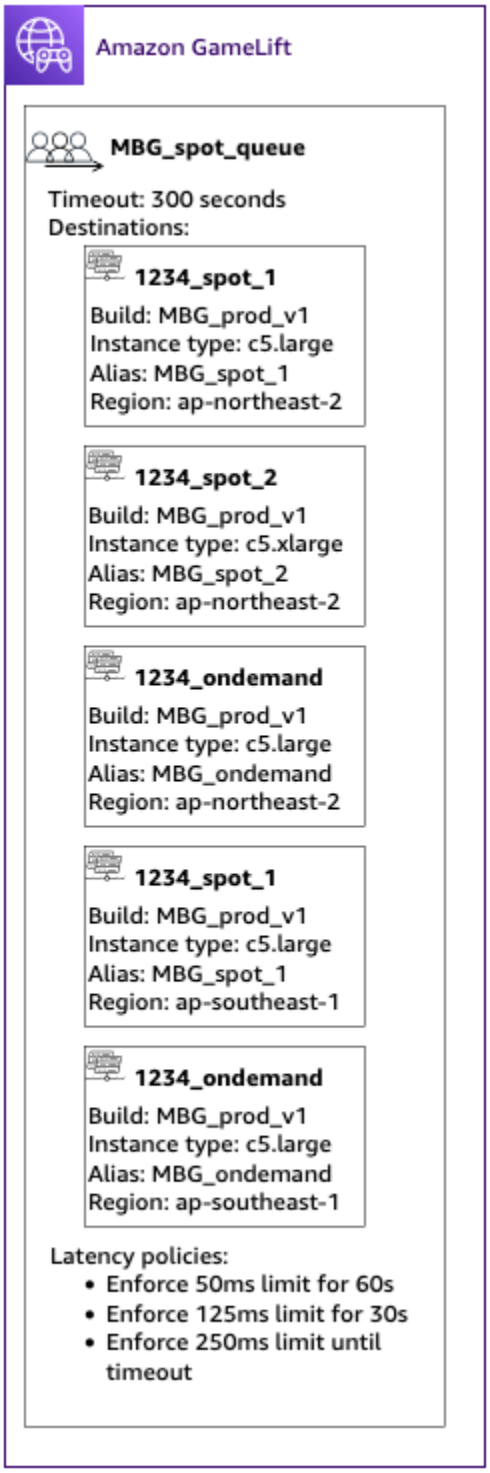
- Latency under 50 milliseconds is ideal.

- The game is unplayable at latencies over 250 milliseconds.
- Players become impatient at about one minute.

For our queue, with a 300-second timeout, we add policy statements limiting the allowable latency. The policy statements gradually allow larger latency values up to 250-millisecond latency.

With this policy, our queue looks for placements with ideal latency (under 50 milliseconds) for the first minute, and then relaxes the limit. The queue doesn't make placements where player latency is 250 milliseconds or higher.

The following diagram shows the queue from step four with player latency policies added. The player latency policies state, enforce 50ms limit for 60s, enforce 125ms limit for 30s, and enforce 250ms limit until timeout.



Summary

Congratulations! Here are the things you accomplished:

- You have a game session queue scoped for a segment of your player population.

- Your queue uses Spot fleets effectively and is resilient when Spot interruptions happen.
- Your queue prioritizes the fleets for the top player experience.
- The queue has latency limits to protect players from bad gameplay experiences.

You can now use the queue to place game sessions for the players it serves. When making game session placement requests for these players, reference this game session queue name in the request. For more information about making game session placement requests, see Create game sessions, or Integrating a game client for Realtime Servers.

Next steps:

- Design your own queue.
- Create a queue.
- Use a queue with your game client.

Manage resources using AWS CloudFormation

You can use AWS CloudFormation to manage your Amazon GameLift resources. 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 Amazon 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 Amazon GameLift resources, see the Amazon GameLift resource type reference.

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 Amazon GameLift.

- **Consistently manage your resources through AWS CloudFormation.** If you change your resources outside of AWS CloudFormation your resources will get out of sync with your resource templates.
- **Use AWS CloudFormation stacks and stack sets to efficiently manage multiple resources.**
  - Use stacks to manage groups of connected resources. For example, a stack that contains a build, a fleet that references the build, and an alias that references the fleet. If you update your template to replace a build, AWS CloudFormation replaces the fleets connected to the 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.
- **Group your location-specific resources and global resources into separate stacks when you are managing resources in multiple locations.**
- **Place your global resources close to the services that use it.** Resources like queues and matchmaking configurations tend to receive a high volume of requests from specific sources. By placing your resources close to the source of those requests, you minimize the request travel time and can improve overall performance.
- **Place your matchmaking configuration in the same Region as the game session queue that it uses.**
- **Create a separate alias for each fleet in the stack.**

Using AWS CloudFormation stacks

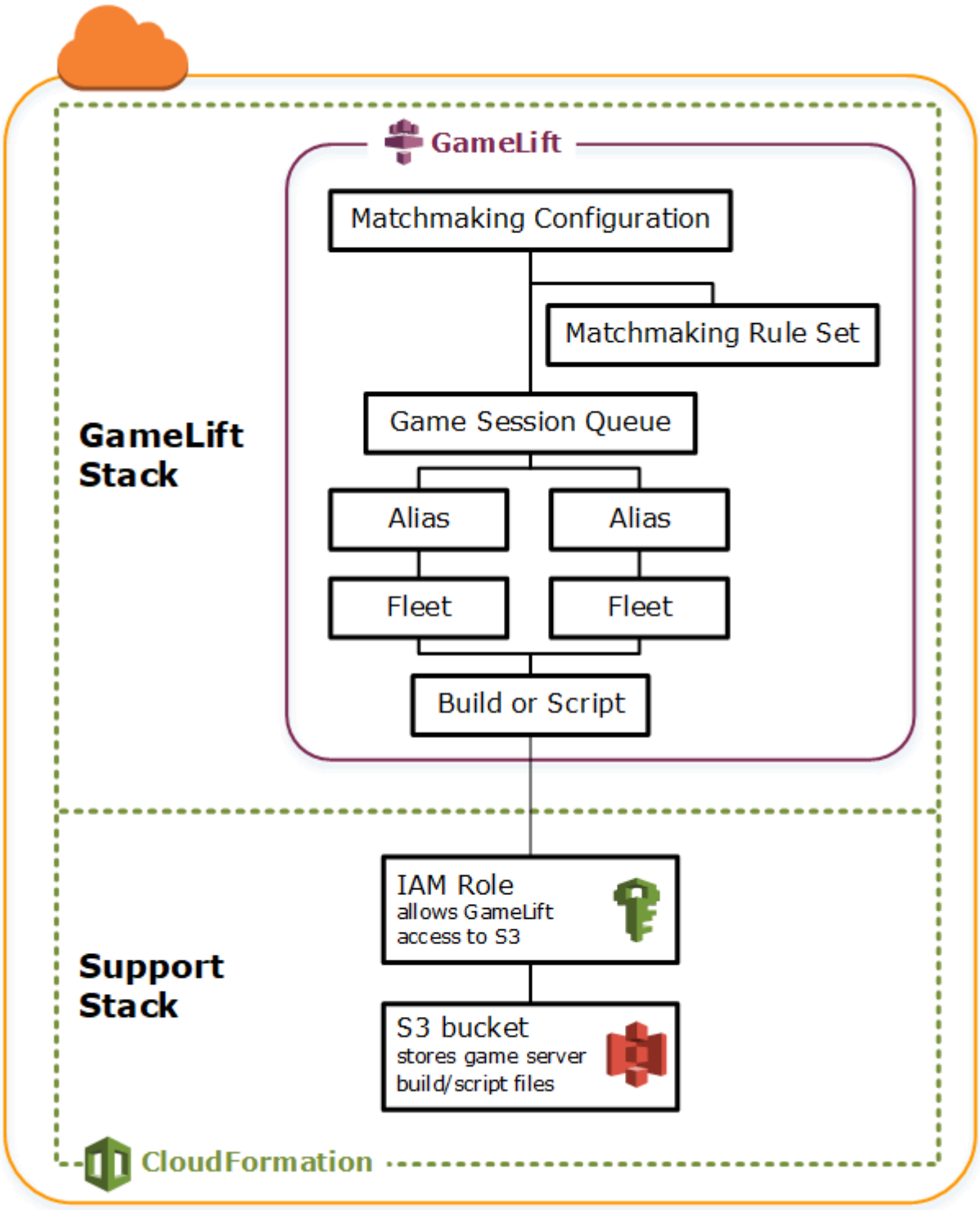
We recommend the following structures to use when setting up AWS CloudFormation stacks for Amazon GameLift resources. Your optimal stack structure varies depending on if you are deploying your game in one location or multiple locations.

Stacks for a single location

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

- **Support stack** – This stack contains resources that your Amazon 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 Amazon GameLift permission to retrieve your files from the S3 bucket when creating a Amazon 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.
- **Amazon GameLift stack** – This stack contains all of your Amazon 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.



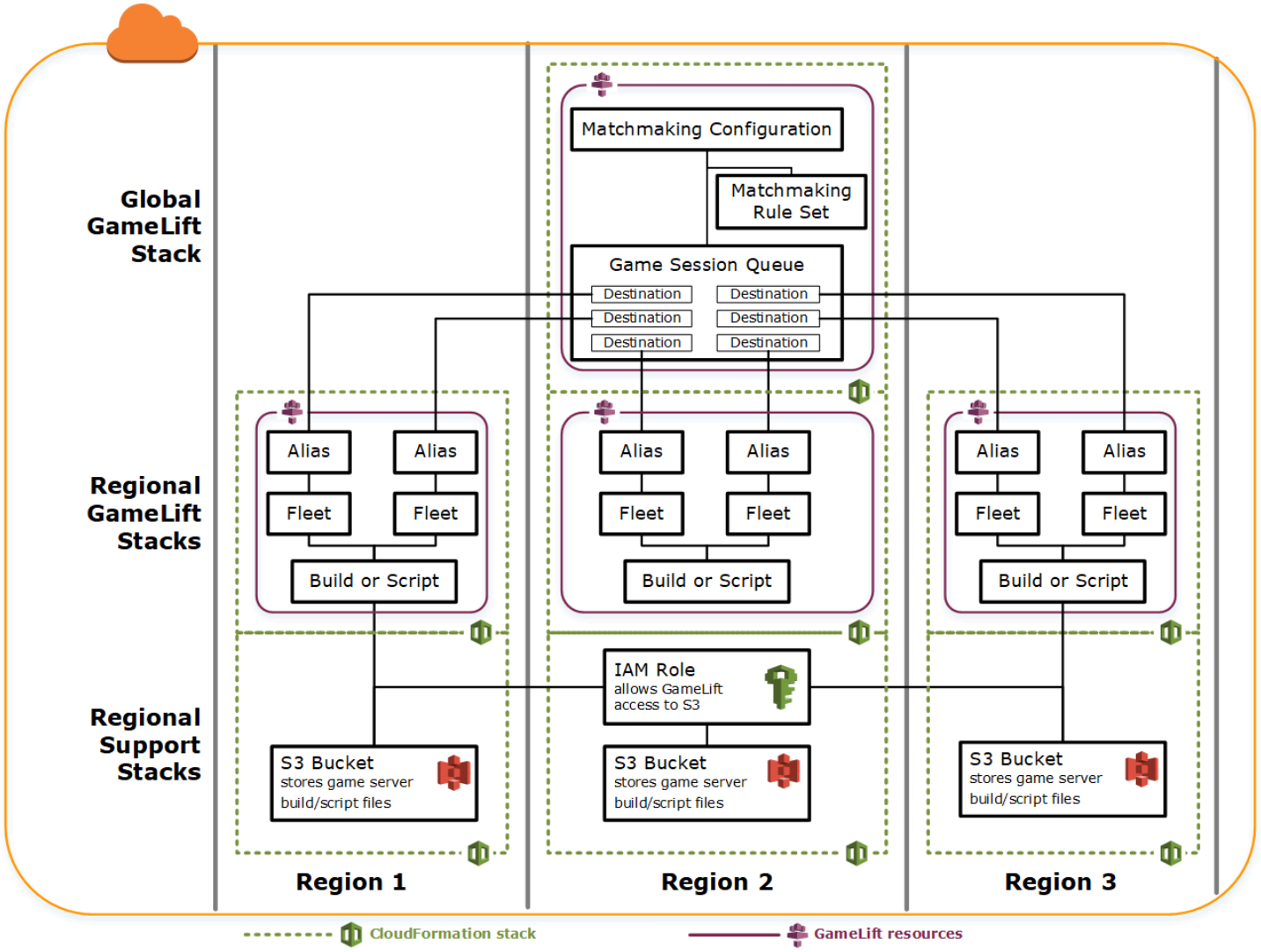
Stacks for multiple regions

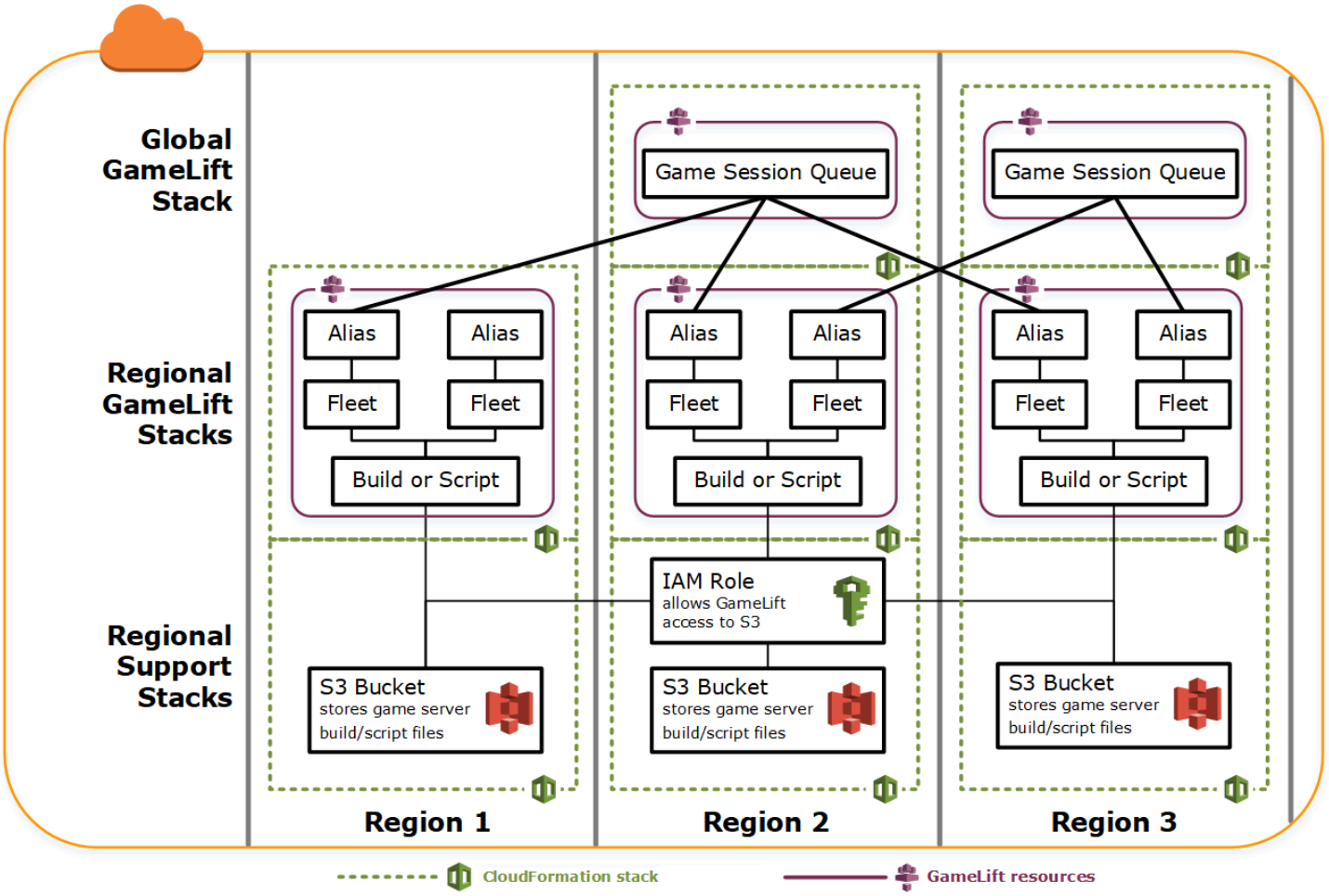
When deploying your game in more than one Region, keep in mind how resources can interact across Regions. Some resources, such as Amazon GameLift fleets, can only reference other resources in the same Region. Other resources, such as a Amazon GameLift queue, are Region

agnostic. To manage Amazon GameLift resources in multiple Regions, we recommend the following structure.

- **Regional support stacks** – These stacks contain resources that your Amazon 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. Amazon 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 Amazon GameLift stacks** –This stack contains the Amazon 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 Amazon 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 illustrates a multistack structure for deploying resources in several AWS Regions. The first diagram shows a structure for a single game session queue. The second diagram shows a structure with multiple queues.


Global GameLift Stack
Regional GameLift Stacks
Regional Support Stacks
Matchmaking Configuration
Matchmaking Rule Set
Game Session Queue
Destination
Alias
Fleet
Build or Script
IAM Role
allows GameLift access to S3
S3 Bucket
stores game server build/script files
Region 1
Region 2
Region 3
CloudFormation stack
GameLift resources



Updating builds

Amazon 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 Amazon 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.

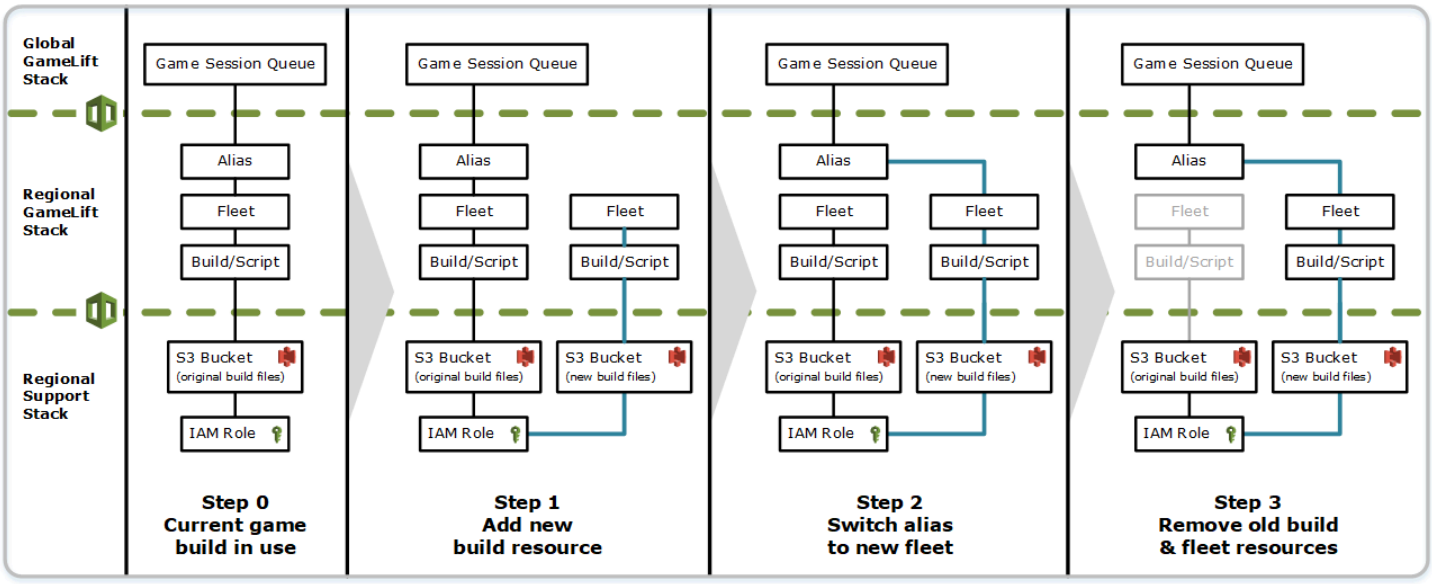
You can also have AWS CloudFormation retain resources instead of deleting them. For more information, see RetainResources in the *AWS CloudFormation API Reference*.

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 Amazon 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 Amazon 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*.

VPC peering for Amazon GameLift

This topic provides guidance on how to set up a VPC peering connection between your Amazon GameLift-hosted game servers and your other non-Amazon 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.

If you're already familiar with Amazon VPCs and VPC peering, understand that setting up peering with Amazon GameLift game servers is somewhat different. You don't have access to the VPC that contains your game servers—it is controlled by the Amazon GameLift service—so you can't directly request VPC peering for it. Instead, you first pre-authorize the VPC with your non-Amazon GameLift resources to accept a peering request from the Amazon GameLift service. Then you trigger Amazon GameLift to request the VPC peering that you just authorized. Amazon 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 Amazon GameLift game servers.
   - AWS account that you use to manage your non-Amazon GameLift resources.

   If you're using the same account for Amazon GameLift and non-Amazon 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 Amazon GameLift game servers – This is your Amazon GameLift fleet ID. Your game servers are deployed in Amazon GameLift on a fleet of EC2 instances. A fleet is automatically placed in its own VPC, which is managed by the Amazon GameLift service. You don't have direct access to the VPC, so it is identified by the fleet ID.

- VPC for your non-Amazon GameLift AWS resources – You can establish a VPC peering with any resources that run on AWS and are managed by an AWS account that you have access to. If you haven't already created a VPC for these resources, see Getting started with Amazon VPC. Once you have created a VPC, you can find the VPC ID by signing into the AWS Management Console for Amazon VPC and viewing your VPCs.

> **Note**
>
> When setting up a peering, both VPCs must exist in the same region. The VPC for your Amazon GameLift fleet game servers is in the same region as the fleet.

3. **Authorize a VPC peering.**

   In this step, you are pre-authorizing a future request from Amazon GameLift to peer the VPC with your game servers with your VPC for non-Amazon GameLift resources. This action updates the security group for your VPC.

   To authorize the VPC peering, call the Amazon GameLift service API CreateVpcPeeringAuthorization() or use the AWS CLI command `create-vpc-peering-authorization`. Make this call using the account that manages your non-Amazon GameLift resources. Identify the following information:

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

   Once you've authorized a VPC peering, the authorization remains valid for 24 hours unless revoked. You can manage your VPC peering authorizations using the following operations:

   - DescribeVpcPeeringAuthorizations() (AWS CLI `describe-vpc-peering-authorizations`).
   - DeleteVpcPeeringAuthorization() (AWS CLI `delete-vpc-peering-authorization`).

4. **Request a peering connection.**

   With a valid authorization, you can request that Amazon GameLift establish a peering connection.

To request a VPC peering, call the Amazon GameLift service API CreateVpcPeeringConnection() or use the AWS CLI command `create-vpc-peering-connection`. Make this call using the account that manages your Amazon 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-Amazon 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 Amazon 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 Amazon 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 Amazon GameLift game servers.
- AWS account that you use to manage your non-Amazon GameLift resources.

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

2. **Get the VPC ID for your non-Amazon 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-Amazon GameLift resources are managed under a different AWS account or user/user group than the one you use with Amazon 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-Amazon GameLift resources.**

   When Amazon GameLift creates the new fleet and a corresponding VPC, it also sends a request to peer with the VPC for your non-Amazon 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-Amazon GameLift resources, call the Amazon 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-Amazon GameLift resources.
   - Amazon GameLift AWS account ID – ID of the account that you use to manage your Amazon GameLift fleet.

   Once you've authorized a VPC peering, the authorization remains valid for 24 hours unless revoked. You can manage your VPC peering authorizations using the following operations:

   - DescribeVpcPeeringAuthorizations() (AWS CLI `describe-vpc-peering-authorizations`).
   - DeleteVpcPeeringAuthorization() (AWS CLI `delete-vpc-peering-authorization`).

4. Follow the instructions for creating a new fleet using the AWS CLI. Include the following additional parameters:

   - *peer-vpc-aws-account-id* – ID for the account that you use to manage the VPC with your non-Amazon GameLift 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-Amazon GameLift AWS account ID and the VPC ID.

```
$ AWS gamelift create-fleet
    --name "My_Fleet_1"
    --description "The sample test fleet"
    --ec2-instance-type "c5.large"
    --fleet-type "ON_DEMAND"
    --build-id "build-1111aaaa-22bb-33cc-44dd-5555eeee66ff"
    --runtime-configuration "GameSessionActivationTimeoutSeconds=300,
                             MaxConcurrentGameSessionActivations=2,
                             ServerProcesses=[{LaunchPath=C:\game\Bin64.dedicated
\MultiplayerSampleProjectLauncher_Server.exe,
                                               Parameters=+sv_port 33435 +start_lobby,
```

```
                                                  ConcurrentExecutions=10}]"
    --new-game-session-protection-policy "FullProtection"
    --resource-creation-limit-policy "NewGameSessionsPerCreator=3,
                                      PolicyPeriodInMinutes=15"
    --ec2-inbound-permissions
 "FromPort=33435,ToPort=33435,IpRange=0.0.0.0/0,Protocol=UDP"

 "FromPort=33235,ToPort=33235,IpRange=0.0.0.0/0,Protocol=UDP"
    --metric-groups  "EMEAfleets"
    --peer-vpc-aws-account-id "111122223333"
    --peer-vpc-id "vpc-a11a11a"
```

Copyable version:

```
AWS gamelift create-fleet --name "My_Fleet_1" --description "The
 sample test fleet" --fleet-type "ON_DEMAND" --metric-groups
 "EMEAfleets" --build-id "build-1111aaaa-22bb-33cc-44dd-5555eeee66ff"
 --ec2-instance-type "c5.large" --runtime-configuration
 "GameSessionActivationTimeoutSeconds=300,MaxConcurrentGameSessionActivations=2,ServerProcesses
\game\Bin64.dedicated\MultiplayerSampleProjectLauncher_Server.exe,Parameters=
+sv_port 33435 +start_lobby,ConcurrentExecutions=10}]" --new-game-session-
protection-policy "FullProtection" --resource-creation-limit-policy
 "NewGameSessionsPerCreator=3,PolicyPeriodInMinutes=15" --ec2-inbound-
permissions "FromPort=33435,ToPort=33435,IpRange=0.0.0.0/0,Protocol=UDP"
 "FromPort=33235,ToPort=33235,IpRange=0.0.0.0/0,Protocol=UDP" --peer-vpc-aws-account-id
 "111122223333" --peer-vpc-id "vpc-a11a11a"
```

Troubleshooting VPC peering issues

If you're having trouble establishing a VPC peering connection for your Amazon GameLift game servers, consider these common root causes:

- An authorization for the requested connection was not found:
  - Check the status of a VPC authorization for the non-Amazon GameLift VPC. It might not exist or it might have expired.
  - Check the regions of the two VPCs you're trying to peer. If they're not in the same region, they can't be peered.
- The CIDR blocks (see Invalid VPC peering connection configurations) of your two VPCs are overlapping. The IPv4 CIDR blocks that are assigned to peered VPCs cannot overlap. The CIDR block of the VPC for your Amazon GameLift fleet is automatically assigned and can't be changed,

so you'll need to change the CIDR block for of the VPC for your non-Amazon GameLift resources. To resolve this issue:

- Look up this CIDR block for your Amazon GameLift fleet by calling `DescribeVpcPeeringConnections()`.
- Go to the Amazon VPC console, find the VPC for your non-Amazon 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 Amazon 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

The **Amazon GameLift dashboard** provides a view of the following:

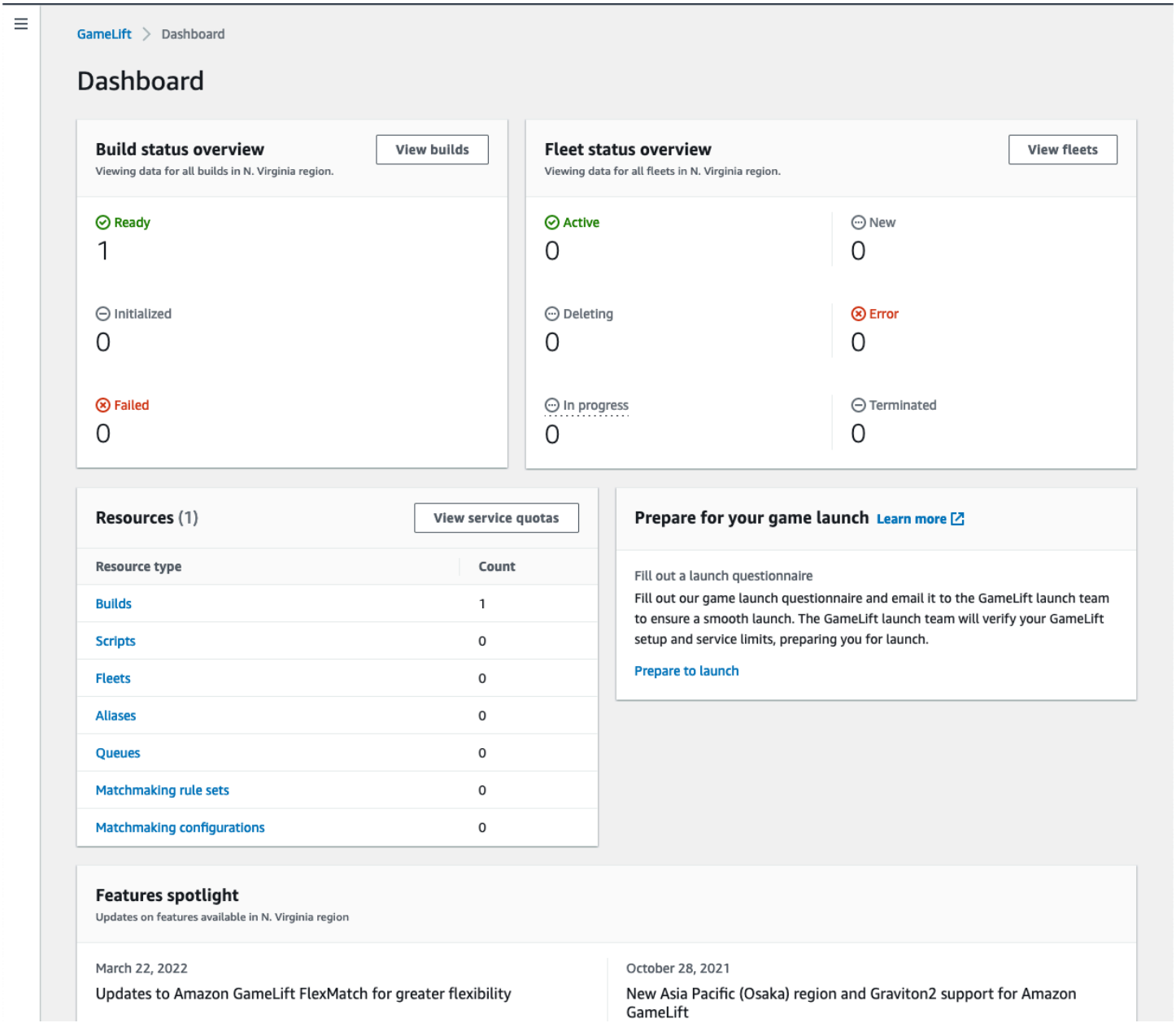
- The number of builds in **Ready**, **Initialized**, and **Failed** statuses. Choose **View builds** for details about builds in your current Region.
- The number of fleets in all statuses. Choose **View fleets** for details about fleets in your current Region.
- Your current resources.
- The overall health of your service, according to AWS Health. For more information, see Concepts for AWS Health in the *AWS Health User Guide*.
- New feature and service announcements.

To open the Amazon GameLift dashboard

- In the Amazon GameLift console, in the navigation pane, choose **Dashboard**.

From the dashboard, you can:

- Prepare your game for launch by choosing **Prepare for launch** and filling out the corresponding launch questionnaire.
- Request service quota increases in preparation for launches or in response to launches by choosing **View service quotas**.
- View blog posts and detailed information about new features by choosing the link in the **Features spotlight**.



View your builds

On the **Builds** page of the Amazon GameLift console, you can view information about and manage all the game server builds that you've uploaded to Amazon GameLift. In the navigation pane, choose **Hosting**, **Builds**.

The **Builds** page shows the following information for each build:

> **Note**
>
> The **Builds** page shows builds in your current AWS Region only.

- **Name** – The name associated with the uploaded build.
- **Status** – The status of the build. Displays one of three status messages:
  - **Initialized** – The upload hasn't started or is still in progress.
  - **Ready** – The build is ready for fleet creation.
  - **Failed** – The build timed out before Amazon GameLift received the binaries.
- **Creation time** – The date and time that you uploaded the build to Amazon GameLift.
- **Build ID** – The unique ID assigned to the build on upload.
- **Version** – The version label associated with the uploaded build.
- **Operating system** – The OS that the build runs on. The build OS determines which operating system Amazon GameLift installs on a fleet's instances.
- **Size** – The size, in megabytes (MB), of the build file uploaded to Amazon GameLift.
- **Fleets** – The number of fleets deployed with the build.

From this page you can do any of the following:

- View build details. Choose a build's name to open its build details page.
- Create a new fleet from a build. Select a build, and then choose **Create fleet**.
- Filter and sort the build list. Use the controls at the top of the table.
- Delete a build. Select a build, and then choose **Delete**.

Build details

On the **Builds** page, choose a build's name to open its details page. The **Overview** section of the details page displays the same build summary information as the **Builds** page. The **Fleets** section shows a list of fleets created with the build, including the same summary information as the **Fleets** page.

View your scripts

On the **Scripts** page of the Amazon GameLift console, you can view information about and manage all the Realtime Servers scripts that you've uploaded to Amazon GameLift. In the navigation pane, choose **Hosting**, **Scripts**.

The **Scripts** page shows the following information for each script:

> **Note**
>
> The **Scripts** page shows scripts in your current AWS Region only.

- **Name** – The name associated with the uploaded script.
- **ID** – The unique ID assigned to the script on upload.
- **Version** – The version label associated with the uploaded script.
- **Size** – The size, in megabytes (MB), of the script file uploaded to Amazon GameLift.
- **Creation time** – The date and time that you uploaded the script to Amazon GameLift.
- **Fleets** – The number of fleets deployed with the script.

From this page you can do any of the following:

- View script details. Choose a build's name to open its script details page.
- Create a new fleet from a script. Select a script, and then choose **Create fleet**.
- Filter and sort the script list. Use the controls at the top of the table.
- Delete a script. Select a script, and then choose **Delete**.

Script details

On the **Scripts** page, choose a script's name to open its details page. The **Overview** section of the details page displays the same script summary information as the **Builds** page. The **Fleets** section shows a list of fleets created with the script, including the same summary information as the **Fleets** page.

View your fleets

You can view information on all the fleets created to host your games on Amazon GameLift under your AWS account. The list shows fleets created your current Region. From the **Fleets** page, you can create a new fleet or view additional detail on a fleet. A fleet's detail page contains usage information, metrics, game session data, and player session data. You can also edit a fleet record or delete a fleet.

To view the **Fleets** page, choose **Fleets** from the navigation pane.

The **Fleets** page displays the following summary information by default. You can customize the information shown by choosing the **Settings** (gear) button.

- **Name** – Friendly name given to the fleet.
- **Status** – The status of the fleet, which can be one of these states: **New**, **Downloading**, **Building**, and **Active**.
- **Creation time** – The date and time the fleet was created.
- **Compute type** – The type of compute used to host your games. A fleet can be a **Managed EC2** fleet or a **Anywhere** fleet.
- **Instance type** – The Amazon EC2 instance type, which determines the computing capacity of fleet's instances.
- **Active instances** – The number of EC2 instances in use for the fleet.
- **Desired instances** – The number of EC2 instances to keep active.
- **Game sessions** – The number of active game sessions running in the fleet. The data is delayed by five minutes.

View fleet details

Access a **Fleet** detail page from the dashboard or the **Fleets** page by choosing the fleet name.

On the fleet details page you can take the following actions:

- Update a fleet's attributes, port settings, and runtime configuration.
- Add or remove fleet locations.
- Change fleet capacity settings.
- Set or change target-tracking auto-scaling.
- Delete a fleet.

Details

Fleet settings

- **Fleet ID** – Unique identifier assigned to the fleet.
- **Name** – The name of the fleet.
- **ARN** – The identifier assigned to this fleet. A fleet's ARN identifies it as an Amazon GameLift resource and specifies the region and AWS account.
- **Description** – A short identifiable description of the fleet.
- **Status** – Current status of the fleet, which may be **New**, **Downloading**, **Building**, and **Active**.
- **Creation time** – The date and time when the fleet was created.
- **Termination time** – The date and time the fleet was terminated. This is blank if the fleet is still active.
- **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.
- **Instance role** – An AWS IAM role that manages access to your other AWS resources, if one was provided during fleet creation.
- **TLS certificate** – Whether the fleet is enabled or disabled to use a TLS certificate for authenticating a game server and encrypting all client/server communication.
- **Metric group** – The group used to aggregate metrics for multiple fleets.
- **Game scaling protection policy** – Current setting for game session protection for the fleet.
- **Maximum game sessions per player** – The maximum number of sessions a player can create during the **Policy period**.
- **Policy period** – How long to wait until resetting the number of sessions a player has created.

Build details

The **Build details** section displays the build hosted on the fleet. Select the build name to see the full build detail page.

Runtime configuration

The **Runtime configuration** section displays the server processes to launch on each instance. It includes the path for the game server executable and optional launch parameters.

Game session activation

The **Game session activation** section displays the number of server processes that launch at the same time and how long to wait for the process to activate before terminating it.

EC2 port settings

The **Ports** section displays the fleet's connection permissions, including IP address and port setting ranges.

Metrics

The **Metrics** tab displays a graphical representation of fleet metrics over time. For more information about using metrics in Amazon GameLift, see Monitor Amazon GameLift with Amazon CloudWatch.

Events

The **Events** tab provides a log of all events that have occurred on the fleet, including the event code, message, and time stamp. See Event descriptions in the Amazon GameLift API Reference.

Scaling

The **Scaling** tab contains information about fleet capacity, including the current status and capacity changes over time. It also provides tools to update capacity limits and manage auto-scaling.

Scaling capacity

View current fleet capacity settings for each fleet location. For more information about changing limits and capacity, see Scaling Amazon GameLift hosting capacity.

- **AWS Location** – Name of a location where fleet instances are deployed.
- **Status** – Hosting status of the fleet location. Location status must be `ACTIVE` to be able to host games.
- **Min size** – The smallest number of instances that must be deployed in the location.
- **Desired instances** – The target number of active instances to maintain the location. When active instances and desired instances aren' the same, a scaling event is started to start or shut down instances as needed until active instances equals desired instances.
- **Max size** – The most instances that can be deployed in the location.
- **Available** – The service limit on instances minus the number of instances in use. This value tells you the maximum number of instances that you can add to the location.

Auto-scaling policies

This section covers information about auto-scaling policies that are applied to the fleet. You can set up or update a target-based policy. The fleet's rule-based policies, which must be defined using the AWS SDK or CLI, are displayed here. For more information about scaling, see Auto-scale fleet capacity with Amazon GameLift.

Scaling history

View graphs of capacity changes over time.

Locations

The **Locations** tab lists all locations where fleet instances are deployed. Locations include the fleet's home Region and any remote locations that have been added. You can add or remove locations directly in this tab.

- **Location** – Name of a location where fleet instances are deployed.
- **Status** – Hosting status of the fleet location. Location status tracks the process of activating the first instances in the location. Location status must be `ACTIVE` to be able to host games.
- **Active instances** – The number of instances with server processes running on the fleet location.
- **Active servers** – The number of game server processes able to host game sessions in the fleet location.
- **Game sessions** – The number of game sessions active on instances in the fleet location.

- **Player sessions** – The number of player sessions, which represent individual players, that are participating in game sessions that are active in the fleet location.

Game sessions

The **Game sessions** tab lists past and present game sessions hosted on the fleet, including some detail information. Choose a game session ID to access additional game session information, including player sessions. For more information about player sessions, see View data on game and player sessions.

View data on game and player sessions

You can view information about the games sessions and individual players. For more information about game sessions and player sessions, see How players connect to games.

To view game session and player data

1. In the Amazon GameLift console, in the navigation pane, choose **Fleets**.
2. Choose the fleet from the **Fleets** list that hosted your game sessions.
3. Choose the **Game sessions** tab. This tab lists all game sessions hosted on the fleet along with summary information.
4. Choose a game session to view additional information about the game session and a list of players that connected to the game.

Details

Overview

This section displays a summary of your game session information.

- **Status** – Game session status.
  - **Activating** – The instance is initiating a game session.
  - **Active** – A game session is running and available to receive players, depending on the session's player creation policy.
  - **Terminated** – the game session has ended.
- **ARN** – The Amazon Resource Name of the game session.

- **Name** – Name generated for the game session.
- **Location** – The location that Amazon GameLift hosted the game session in.
- **Creation time** – Date and time that Amazon GameLift created the stream session.
- **Ending time** – Date and time that the game session ended.
- **DNS name** – The host name of the game session.
- **IP address** – IP address specified for the game session.
- **Port** – Port number used to connect to the game session.
- **Creator ID** – A unique identifier of the player that initiated the game session.
- **Player session creation policy** – Indicates if the game session is accepting new players.
- **Game scaling protection policy** – The type of game session protection to set on all new instances that Amazon GameLift starts in the fleet.

Game data

Well-formatted data to send to your game session on start.

Game properties

Key and value pair properties that influence your game session.

Matchmaking data

The FlexMatch matchmaker JSON. To review and edit the matchmaker choose **View matchmaking configuration**. For more information about FlexMatch matchmaking, see Build a matchmaker.

Player sessions

The following player session data is collected for each game session:

- **Player session ID** – The identifier assigned to the player session.
- **Player ID** – A unique identifier for the player. Choose this ID to get additional player information.
- **Status** – The status of the player session. The following are possible statuses:
  - **Reserved** – Player session has been reserved, but the players isn't connected.
  - **Active** – Player session is connected to the game server.
  - **Completed** – Player session has ended; player is no longer connected.
  - **Timed Out** – Player failed to connect.

- **Creation time** – The time the player connected to the game session.
- **Ending time** – The time the player disconnected from the game session.
- **Player data** – Information about the player provided during player session creation.

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 times, end times, and total connected time for each player session. You can choose to view data for the relevant game sessions and fleets.

View your aliases

The **Alias** page displays information about the fleet aliases you created in your current Region. To view the aliases page, choose **Aliases** in the navigation pane.

You can do the following on the aliases page:

- Create a new alias. Choose **Create alias**.
- Filter and sort the aliases table. Use the controls at the top of the table.
- View alias details. Choose an alias name to open the alias detail page.
- Delete an alias. Choose an alias and then choose **Delete**.

Alias details

The alias details page displays information about the alias.

From this page you can:

- Edit an alias. Choose **Edit**.
- View the fleets you associated with the alias.
- Delete an alias. Choose **Delete**.

Alias detail information includes:

- **ID** – The unique number used to identify the alias.

- **Description** – The description of the alias.
- **ARN** – The Amazon Resource Name of the alias.
- **Creation** – The date and time the alias was created.
- **Last updated** – The date and time that the alias was last updated.
- **Routing type** – The routing type for the alias, which can be one of these:
  - **Simple** – Routes player traffic to a specified fleet ID. You can update the fleet ID for an alias at any time.
  - **Terminal** – Passes a message back to the client. For example, you can direct players who are using an out-of-date client to a location where they can get an upgrade.
- **Tags** – Key and value pairs used to identify the alias.

View your queues

You can view information on all existing game session placement queues. The queues page shows queues created in your current Region. From the **Queues** page, you can create a new queue, delete existing queues, or open a details page for a selected queue. Each queue details page contains the queue's configuration and metrics data. For more information about queues, see Setting up Amazon GameLift queues for game session placement.

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. Amazon GameLift places new game sessions on any fleet in the queue.

View queue details

You can access detailed information on any queue, including the queue configuration and metrics. To open a queue details page, go to the **Queues** page and choose a queue name.

The queue detail page displays a summary table and tabs containing additional information. On this page you can do the following:

- Update the queue's configuration, list of destinations and player latency policies. Choose **Edit**.
- Delete a queue. After you delete a queue, all requests for new game sessions that reference that queue name will fail. Choose **Delete**.

> **Note**
>
> To restore a deleted queue, create a new queue with the deleted queue's name.

Details

Overview

The **Overview** section displays the queue's Amazon Resource Name (**ARN**) and the **Timeout**. You can use the ARN when referencing the queue in other actions or areas of Amazon GameLift. The timeout is the maximum length of time, in seconds, that a game session placement request remains in the queue before timing out.

Event notification

The **Event notification** section lists the **SNS topic** Amazon GameLift publishes event notifications to and the **Event data** that is added to all events created by this queue.

Tags

The **Tags** table displays the keys and values used to tag the resource. For more information about tagging, see Tagging AWS resources.

Metrics

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

Queue metrics include a range of information describing placement activity across the queue, including successful placements organized by Region. You can use Region data to understand where you are hosting your games. Regional placement metrics can help to detect issues with the overall queue design.

Queue metrics are also available in Amazon CloudWatch. For descriptions of available metrics, see Amazon GameLift metrics for queues.

Destinations

The **Destinations** tab shows all fleets or aliases listed for the queue.

When Amazon GameLift searches the destinations for available resources to host a new game session, it searches the default order listed here. As long as there is capacity on the first destination listed, Amazon GameLift places new game sessions there. You can have individual game session placement requests override the default order by providing player latency data. This data tells Amazon GameLift to search for an available destination with the lowest average player latency. For more information about designing your queues, see Design a game session queue.

Session placement

Player latency policies

The **Player latency policies** section shows all policies that the queue uses. The tables lists the policies in the order they're enforced.

Locations

The **Locations** section shows the locations that this queue can put a game session in.

Priority

The **Priority** section shows the order that the queue evaluates a game sessions details.

Location order

The **Location order** section shows the default order that the queue uses when placing game sessions. The queue uses this order if you haven't defined other types of priority.

Monitoring Amazon GameLift

If you're using Amazon GameLift FleetIQ as a standalone feature with Amazon EC2, see 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.
- *Game session logs* – You can output custom server messages for your game sessions to log files that are stored in Amazon S3.

Topics

Monitor Amazon GameLift with Amazon CloudWatch

You can monitor Amazon GameLift using Amazon CloudWatch, an AWS service that collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months

to provide a historical perspective on how your game server hosting with Amazon 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 Amazon GameLift. All metrics that are available in CloudWatch are also available in the Amazon GameLift console, which provides the data as a set of customizable graphs. To access CloudWatch metrics for your games, use the AWS Management Console, the AWS CLI, or the CloudWatch API.

If a metric does not have a location, it uses the home location.

Dimensions for Amazon GameLift metrics

Amazon GameLift supports filtering metrics by the following dimensions.

Dimension	Description
`Location`	Filter metrics for a fleet deployment location. If a metric does not have a location, it uses the home location.
`FleetId`	Filter metrics for a single fleet. This dimension can be used with all fleet metrics for instances, server processes, game sessions, and player sessions.
`MetricGroup`	Filter metrics for a collection of fleets. Add a fleet to a metric group by adding the metric group name to the fleet's attributes (see UpdateFleetAttributes()). This dimension can be used with all fleet metrics for instances, server processes, game sessions, and player sessions.
`QueueName`	Filter metrics for a single queue. This dimension is used with metrics for game session queues only.
`ConfigurationName`	Filter metrics for a single matchmaking configuration. This dimension is used with metrics for matchmaking configurations.

Dimension	Description
ConfigurationName-RuleName	Filter metrics for an intersect of a matchmaki ng configuration and matchmaking rule. This dimension is used with metrics for matchmaking rules only.
InstanceType	Filter metrics for an EC2 instance type designati on, such as "c4.large". This dimension is used with metrics for spot instances.
OperatingSystem	Filter metrics for an instance's operating system This dimension is used with metrics for spot instances.
GameServerGroup	Filter FleetIQ metrics for a game server group.

Amazon 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 Amazon GameLift solution. The Amazon GameLift service sends metrics to CloudWatch every minute.

Instances

Metric	Description
ActiveInstances	Instances with ACTIVE status, which means they are running active server processes. The count includes idle instances and those that are hosting one or more game sessions. This metric measures current total instance capacity. This metric can be used with automatic scaling. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum

Metric	Description
	Dimensions: Location
DesiredInstances	Target number of active instances that Amazon GameLift is working to maintain in the fleet. With automatic scaling, this value is determined based on the scaling policies currently in force. Without automatic scaling, this value is set manually. This metric is not available when viewing data for fleet metric groups. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
IdleInstances	Active instances that are currently hosting zero (0) game sessions. This metric measures capacity that is available but unused. This metric can be used with automatic scaling. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location

Metric	Description
`MaxInstances`	Maximum number of instances that are allowed for the fleet. A fleet's instance maximum determine s the capacity ceiling during manual or automatic scaling up. This metric is not available when viewing data for fleet metric groups. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
`MinInstances`	Minimum number of instances allowed for the fleet. A fleet's instance minimum determines the capacity floor during manual or automatic scaling down. This metric is not available when viewing data for fleet metric groups. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
`PercentIdleInstances`	Percentage of all active instances that are idle (calculated as `IdleInstances / ActiveIns tances`). This metric can be used for automatic scaling. Units: Percent Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location

Metric	Description
RecycledInstances	Number of spot instances that have been recycled and replaced. Amazon GameLift recycles spot instances that are not currently hosting game sessions and have a high probability of interruption. Units: Count Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: Location
InstanceInterruptions	Number of spot instances that have been interrupt ed. Units: Count Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: Location
CPUUtilization	*EC2 metric. For Amazon GameLift this metric represents hardware performance across all active instances in a fleet location.* The percentage of physical CPU time that Amazon EC2 uses to run the instance, which includes time spent to run both the user code and Amazon EC2 code. Tools in your operating system can show a different percentag e than CloudWatch due to factors such as legacy device simulation, configuration of non-legacy devices, interrupt-heavy workloads, live migration, and live update. Units: Percent

Metric	Description
NetworkIn	*EC2 metric. For Amazon GameLift this metric represents hardware performance across all active instances in a fleet location.* The number of bytes received on all network interfaces by the instance. This metric identifies the volume of incoming network traffic to an application on a single instance. Units: Bytes
NetworkOut	*EC2 metric. For Amazon GameLift this metric represents hardware performance across all active instances in a fleet location.* The number of bytes sent out on all network interfaces by the instance. This metric identifies the volume of outgoing network traffic to an application on a single instance. Units: Bytes
DiskReadBytes	*EC2 metric. For Amazon GameLift this metric represents hardware performance across all active instances in a fleet location.* Bytes read from all instance store volumes available to the instance. This metric is used to determine the volume of the data the application reads from the hard disk of the instance. You can use it to determine the speed of the application. Units: Bytes

Metric	Description
DiskWriteBytes	*EC2 metric. For Amazon GameLift this metric represents hardware performance across all active instances in a fleet location.* Bytes written to all instance store volumes available to the instance. This metric is used to determine the volume of the data the application writes onto the hard disk of the instance. You can use it to determine the speed of the application. Units: Bytes
DiskReadOps	*EC2 metric. For Amazon GameLift this metric represents hardware performance across all active instances in a fleet location.* Completed read operations from all instance store volumes available to the instance in a specified period of time. To calculate the average I/O operations per second (IOPS) for the period, divide the total operations in the period by the number of seconds in that period. Units: Count
DiskWriteOps	*EC2 metric. For Amazon GameLift this metric represents hardware performance across all active instances in a fleet location.* Completed write operations to all instance store volumes available to the instance in a specified period of time. To calculate the average I/O operations per second (IOPS) for the period, divide the total operations in the period by the number of seconds in that period. Units: Count

Server processes

Metric	Description
`ActiveServerProcesses`	Server processes with ACTIVE status, which means they are running and able to host game sessions. The count includes idle server processes and those that are hosting game sessions. This metric measures current total server process capacity. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
`HealthyServerProcesses`	Active server processes that are reporting healthy. This metric is useful for tracking the overall health of the fleet's game servers. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
`PercentHealthyServerProcesses`	Percentage of all active server processes that are reporting healthy (calculated as `HealthyServerProcesses / ActiveServerProcesses`). Units: Percent Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location

Metric	Description
ServerProcessAbnormalTerminations	Server processes that were shut down due to abnormal circumstances since the last report. This metric includes terminations that were initiated by the Amazon GameLift 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
ServerProcessActivations	Server processes that successfully transitioned from ACTIVATING to ACTIVE status since the last report. Server processes cannot host game sessions until they are active. Units: Count Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: Location

Metric	Description
ServerProcessTerminations	Server processes that were shut down since the last report. This includes all server processes that transitioned to TERMINATED status for any reason, including normal and abnormal process terminations. Units: Count Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: Location

Game sessions

Metric	Description
ActivatingGameSessions	Game sessions with ACTIVATING status, which means they are in the process of starting up. Game sessions cannot host players until they are active. High numbers for a sustained period of time may indicate that game sessions are not transitioning from ACTIVATING to ACTIVE status. This metric can be used with automatic scaling. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
ActiveGameSessions	Game sessions with ACTIVE status, which means they are able to host players, and are hosting zero or more players. This metric measures the total

Metric	Description
	number of game sessions currently being hosted. This metric can be used with automatic scaling. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
`AvailableGameSessions`	Active, healthy server processes that are not currently being used to host a game session and can start a new game session without a delay to spin up new server processes or instances. This metric can be used with automatic scaling. **Note** For fleets that limit concurrent game session activations, use the metric `ConcurrentActivatableGameSessions`. That metric more accurately represents the number of new game sessions that can start without any type of delay. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location

Metric	Description
ConcurrentActivatableGameSessions	Active, healthy server processes that are not currently being used to host a game session and can immediately start a new game session. This metric differs from 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. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: Location
PercentAvailableGameSessions	Percentage of game session slots on all active server processes (healthy or unhealthy) that are not currently being used (calculated as AvailableGameSessions / [ActiveGameSessions + AvailableGameSessions + unhealthy server processes]). This metric can be used with automatic scaling. Units: Percent Relevant CloudWatch statistics: Average Dimensions: Location

Metric	Description
GameSessionInterruptions	Number of game sessions on spot instances that have been interrupted. Units: Count Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum Dimensions: Location

Player sessions

Metric	Description
CurrentPlayerSessions	Player sessions with either ACTIVE status (player is connected to an active game session) or RESERVED status (player has been given a slot in a game session but hasn't yet connected). This metric can be used with automatic scaling. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum
PlayerSessionActivations	Player sessions that transitioned from RESERVED status to ACTIVE since the last report. This occurs when a player successfully connects to an active game session. Units: Count Relevant CloudWatch statistics: Sum, Average, Minimum, Maximum

Amazon GameLift metrics for queues

The `Amazon GameLift` namespace includes the following metrics related to activity across a game session placement queue. Queues are used with a managed Amazon GameLift solution. The Amazon GameLift service sends metrics to CloudWatch every minute.

Metric	Description
`AverageWaitTime`	Average amount of time that game session placement requests in the queue with status PENDING have been waiting to be fulfilled. Units: Seconds Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum Dimensions: Location
`FirstChoiceNotViable`	Game sessions that were successfully placed but NOT in the first-choice fleet, because that fleet was considered not viable (such as a spot fleet with a high interruption rate). This metric is based on cost, not latency. The first-choice fleet is either the first fleet listed in the queue or—when a placement request includes player latency data—it is the first fleet chosen by FleetIQ prioritization. If there are no viable spot fleets, any fleet in that region may be selected. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum
`FirstChoiceOutOfCapacity`	Game sessions that were successfully placed but NOT in the first-choice fleet, because that fleet had no available resources. The first-choice fleet is either the first fleet listed in the queue or—when a

Metric	Description
	placement request includes player latency data — it is the first fleet chosen by your defined FleetIQ prioritization. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum
`LowestLatencyPlacement`	Game sessions that were successfully placed in a region that offers the queue's lowest possible latency for the players. This metric is emitted only when player latency data is included in the placement request. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum
`LowestPricePlacement`	Game sessions that were successfully placed in a fleet with the queue's lowest possible price for the chosen region. This fleet can be either a spot fleet or an on-demand instance if the queue has no spot instances. This metric is emitted only when player latency data is included in the placement request. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum

Metric	Description
`Placement <region name>`	Game sessions that are successfully placed in fleets located in the specified region. This metric breaks down the `PlacementsSucceeded` metric by region. Units: Count Relevant CloudWatch statistics: Sum
`PlacementsCanceled`	Game session placement requests that were canceled before timing out since the last report. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum
`PlacementsFailed`	Game session placement requests that failed for any reason since the last report. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum
`PlacementsStarted`	New game session placement requests that were added to the queue since the last report. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum

Metric	Description
PlacementsSucceeded	Game session placement requests that resulted in a new game session since the last report. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum
PlacementsTimedOut	Game session placement requests that reached the queue's timeout limit without being fulfilled since the last report. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum
QueueDepth	Number of game session placement requests in the queue with status PENDING. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum Dimensions: Location

Amazon GameLift metrics for matchmaking

The Amazon GameLift namespace includes metrics on FlexMatch activity for matchmaking configurations and matchmaking rules. FlexMatch matchmaking is used with a managed Amazon GameLift solution. The Amazon GameLift service sends metrics to CloudWatch every minute.

For more information on the sequence of matchmaking activity, see How Amazon GameLift FlexMatch works.

Matchmaking configurations

Metric	Description
CurrentTickets	Matchmaking requests currently being processed or waiting to be processed. Units: Count Relevant CloudWatch statistics: Average, Minimum, Maximum, Sum
MatchAcceptancesTimedOut	For matchmaking configurations that require acceptance, the potential matches that timed out during acceptance since the last report. Units: Count Relevant CloudWatch statistics: Sum
MatchesAccepted	For matchmaking configurations that require acceptance, the potential matches that were accepted since the last report. Units: Count Relevant CloudWatch statistics: Sum
MatchesCreated	Potential matches that were created since the last report. Units: Count Relevant CloudWatch statistics: Sum
MatchesPlaced	Matches that were successfully placed into a game session since the last report. Units: Count Relevant CloudWatch statistics: Sum

Metric	Description
MatchesRejected	For matchmaking configurations that require acceptance, the potential matches that were rejected by at least one player since the last report. Units: Count Relevant CloudWatch statistics: Sum
PlayersStarted	Players in matchmaking tickets that were added since the last report. Units: Count Relevant CloudWatch statistics: Sum
TicketsFailed	Matchmaking requests that resulted in failure since the last report. Units: Count Relevant CloudWatch statistics: Sum
TicketsStarted	New matchmaking requests that were created since the last report. Units: Count Relevant CloudWatch statistics: Sum
TicketsTimedOut	Matchmaking requests that reached the timeout limit since the last report. Units: Count Relevant CloudWatch statistics: Sum

Metric	Description
TimeToMatch	For matchmaking requests that were put into a potential match before the last report, the amount of time between ticket creation and potential match creation. Units: Seconds Relevant CloudWatch statistics: Data Samples, Average, Minimum, Maximum
TimeToTicketCancel	For matchmaking requests that were canceled before the last report, the amount of time between ticket creation and cancellation. Units: Seconds Relevant CloudWatch statistics: Data Samples, Average, Minimum, Maximum
TimeToTicketSuccess	For matchmaking requests that succeeded before the last report, the amount of time between ticket creation and successful match placement. Units: Seconds Relevant CloudWatch statistics: Data Samples, Average, Minimum, Maximum

Matchmaking rules

Metric	Description
RuleEvaluationsPassed	Rule evaluations during the matchmaking process that passed since the last report. This metric is limited to the top 50 rules. Units: Count

Metric	Description
	Relevant CloudWatch statistics: Sum
`RuleEvaluationsFailed`	Rule evaluations during matchmaking that failed since the last report. This metric is limited to the top 50 rules. Units: Count Relevant CloudWatch statistics: Sum

Amazon GameLift metrics for FleetIQ

The `Amazon GameLift` namespace includes metrics for FleetIQ game server group and game server activity as part of a FleetIQ standalone solution for game hosting. The Amazon GameLift service sends metrics to CloudWatch every minute. Also see Monitoring your Auto Scaling groups and instances using amazon CloudWatch in the *Amazon EC2 Auto Scaling User Guide*.

Metric	Description
`AvailableGameServers`	Game servers that are available to run a game execution and are not currently occupied with gameplay. This number includes game servers that have been claimed but are still in AVAILABLE status. Units: Count Relevant CloudWatch statistics: Sum Dimensions: GameServerGroup
`UtilizedGameServers`	Game servers that are currently occupied with gameplay. This number includes game servers that are in UTILIZED status. Units: Count Relevant CloudWatch statistics: Sum

Metric	Description
	Dimensions: GameServerGroup
DrainingAvailableGameServers	Game servers on instances scheduled for terminati on that are currently not supporting gameplay. These game servers are the lowest priority to be claimed in response to a new claim request. Units: Count Relevant CloudWatch statistics: Sum Dimensions: GameServerGroup
DrainingUtilizedGameServers	Game servers on instances scheduled for terminati on that are currently supporting gameplay. Units: Count Relevant CloudWatch statistics: Sum Dimensions: GameServerGroup
PercentUtilizedGameServers	Portion of game servers that are currently supportin g game executions. This metric indicates the amount of game server capacity that is currently in use. It is useful for driving an Auto Scaling policy that can dynamically add and remove instances to match with player demand. Units: Percent Relevant CloudWatch statistics: Average, Minimum, Maximum Dimensions: GameServerGroup

Metric	Description
GameServerInterruptions	Game servers on Spot Instances that were interrupted due to limited Spot availability. Units: Count Relevant CloudWatch statistics: Sum Dimensions: GameServerGroup, InstanceType
InstanceInterruptions	Spot Instances that were interrupted due to limited availability. Units: Count Relevant CloudWatch statistics: Sum Dimensions: GameServerGroup, InstanceType

Logging Amazon GameLift API calls with AWS CloudTrail

Amazon GameLift is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon GameLift. CloudTrail captures all API calls for Amazon GameLift as events. The calls captured include calls from the Amazon GameLift console and code calls to the Amazon GameLift API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Amazon GameLift. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon GameLift, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide.

Amazon GameLift information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Amazon GameLift, that activity is recorded in a CloudTrail event along with other AWS service

events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail Event history.

For an ongoing record of events in your AWS account, including events for Amazon GameLift, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring Amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple regions and Receiving CloudTrail log files from multiple accounts

All Amazon GameLift actions are logged by CloudTrail and are documented in the *Amazon GameLift API Reference*. For example, calls to `CreateGameSession`, `CreatePlayerSession` and `UpdateGameSession` actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Understanding Amazon GameLift log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `CreateFleet` and `DescribeFleetAttributes` actions.


```
{
    "Records": [
        {
            "eventVersion": "1.04",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "AIDACKCEVSQ6C2EXAMPLE",
                "arn": "arn:aws:iam::111122223333:user/myUserName",
                "accountId": "111122223333",
                "accessKeyId": AKIAIOSFODNN7EXAMPLE",
                "userName": "myUserName"
            },
            "eventTime": "2015-12-29T23:40:15Z",
            "eventSource": "gamelift.amazonaws.com",
            "eventName": "CreateFleet",
            "awsRegion": "us-west-2",
            "sourceIPAddress": "192.0.2.0",
            "userAgent": "[]",
            "requestParameters": {
                "buildId": "build-92b6e8af-37a2-4c10-93bd-4698ea23de8d",
                "eC2InboundPermissions": [
                    {
                        "ipRange": "10.24.34.0/23",
                        "fromPort": 1935,
                        "protocol": "TCP",
                        "toPort": 1935
                    }
                ],
                "logPaths": [
                    "C:\\game\\serverErr.log",
                    "C:\\game\\serverOut.log"
                ],
                "eC2InstanceType": "c5.large",
                "serverLaunchPath": "C:\\game\\MyServer.exe",
                "description": "Test fleet",
                "serverLaunchParameters": "-paramX=baz",
                "name": "My_Test_Server_Fleet"
            },
            "responseElements": {
                "fleetAttributes": {
```

```
                "fleetId": "fleet-0bb84136-4f69-4bb2-bfec-a9b9a7c3d52e",
                "serverLaunchPath": "C:\\game\\MyServer.exe",
                "status": "NEW",
                "logPaths": [
                    "C:\\game\\serverErr.log",
                    "C:\\game\\serverOut.log"
                ],
                "description": "Test fleet",
                "serverLaunchParameters": "-paramX=baz",
                "creationTime": "Dec 29, 2015 11:40:14 PM",
                "name": "My_Test_Server_Fleet",
                "buildId": "build-92b6e8af-37a2-4c10-93bd-4698ea23de8d"
            }
        },
        "requestID": "824a2a4b-ae85-11e5-a8d6-61d5cafb25f2",
        "eventID": "c8fbea01-fbf9-4c4e-a0fe-ad7dc205ce11",
        "eventType": "AwsApiCall",
        "recipientAccountId": "111122223333"
    },
    {
        "eventVersion": "1.04",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "AIDACKCEVSQ6C2EXAMPLE",
            "arn": "arn:aws:iam::111122223333:user/myUserName",
            "accountId": "111122223333",
            "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
            "userName": "myUserName"
        },
        "eventTime": "2015-12-29T23:40:15Z",
        "eventSource": "gamelift.amazonaws.com",
        "eventName": "DescribeFleetAttributes",
        "awsRegion": "us-west-2",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "[]",
        "requestParameters": {
            "fleetIds": [
                "fleet-0bb84136-4f69-4bb2-bfec-a9b9a7c3d52e"
            ]
        },
        "responseElements": null,
        "requestID": "82e7f0ec-ae85-11e5-a8d6-61d5cafb25f2",
        "eventID": "11daabcb-0094-49f2-8b3d-3a63c8bad86f",
        "eventType": "AwsApiCall",
```

```
            "recipientAccountId": "111122223333"
        },
    ]
}
```

Logging server messages in Amazon GameLift

You can capture custom server messages from your Amazon GameLift servers in log files. The way you configure logging depends on whether you use custom servers or Realtime Servers (see the appropriate subsections in this chapter).

Topics

Logging server messages (custom servers)

You can capture custom server messages from your Amazon GameLift custom servers in log files. To learn about logging for Realtime Servers, see Logging server messages (Realtime Servers).

> **Important**
>
> There is a limit on the size of a log file per game session (see Amazon GameLift endpoints and quotas in the *AWS General Reference*). When a game session ends, Amazon GameLift uploads the server logs to Amazon Simple Storage Service (Amazon S3). Amazon GameLift will not upload logs that exceed the limit. Logs can grow very quickly and exceed the size limit. You should monitor your logs and limit the log output to necessary messages only.

Configuring logging for custom servers

With Amazon GameLift custom servers, you write your own code to perform logging, which you configure as part of your server process configuration. Amazon GameLift uses your logging configuration to identify the files that it must upload to Amazon S3 at the end of each game session.

The following instructions show how to configure logging using simplified code examples:

C++

To configure logging (C++)

1. Create a vector of strings that are directory paths to game server log files.

```
std::string serverLog("serverOut.log");        // Example server log file
std::vector<std::string> logPaths;
logPaths.push_back(serverLog);
```

2. Provide your vector as the LogParameters of your ProcessParameters object.

```
Aws::GameLift::Server::ProcessParameters processReadyParameter =
 Aws::GameLift::Server::ProcessParameters(
    std::bind(&Server::onStartGameSession, this, std::placeholders::_1),
    std::bind(&Server::onProcessTerminate, this),
    std::bind(&Server::OnHealthCheck, this),
    std::bind(&Server::OnUpdateGameSession, this),
    listenPort,
    Aws::GameLift::Server::LogParameters(logPaths));
```

3. Provide the ProcessParameters object when you call ProcessReady().

```
Aws::GameLift::GenericOutcome outcome =
   Aws::GameLift::Server::ProcessReady(processReadyParameter);
```

For a more complete example, see ProcessReady().

C#

To configure logging (C#)

1. Create a list of strings that are directory paths to game server log files.

```
List<string> logPaths = new List<string>();
logPaths.Add("C:\\game\\serverOut.txt");     // Example of a log file that the
 game server writes
```

2. Provide your list as the LogParameters of your ProcessParameters object.

```
var processReadyParameter = new ProcessParameters(
    this.OnGameSession,
```

```
    this.OnProcessTerminate,
    this.OnHealthCheck,
    this.OnGameSessionUpdate,
    port,
    new LogParameters(logPaths));
```

3. Provide the ProcessParameters object when you call ProcessReady().

```
var processReadyOutcome =
   GameLiftServerAPI.ProcessReady(processReadyParameter);
```

For a more complete example, see ProcessReady().

Writing to logs

Your log files exist after your server process has started. You can write to the logs using any method to write to files. To capture all of your server's standard output and error output, remap the output streams to log files, as in the following examples:

C++

```
std::freopen("serverOut.log", "w+", stdout);
std::freopen("serverErr.log", "w+", stderr);
```

C#

```
Console.SetOut(new StreamWriter("serverOut.txt"));
Console.SetError(new StreamWriter("serverErr.txt"));
```

Accessing server logs

When a game session ends, Amazon GameLift automatically stores the logs in an Amazon S3 bucket and retains them for 14 days. To get the location of the logs for a game session, you can use the GetGameSessionLogUrl API operation. To download the logs, use the URL that the operation returns.

Logging server messages (Realtime Servers)

You can capture custom server messages from your Realtime Servers in log files. To learn about logging for custom servers, see Logging server messages (custom servers).

There are different types of messages that you can ouptput to your log files (see Logging messages in your server script). In addition to your custom messages, your Realtime Servers output system messages using the same message types and write to the same log files. You can adjust the logging level for your fleet to reduce the amount of logging messages that your servers generate (see Adjusting the logging level).

> **Important**
>
> There is a limit on the size of a log file per game session (see Amazon GameLift endpoints and quotas in the *AWS General Reference*). When a game session ends, Amazon GameLift uploads the server logs to Amazon Simple Storage Service (Amazon S3). Amazon GameLift will not upload logs that exceed the limit. Logs can grow very quickly and exceed the size limit. You should monitor your logs and limit the log output to necessary messages only.

Logging messages in your server script

You can output custom messages in the script for your Realtime Servers. Use the following steps to send server messages to a log file:

1. Create a varaible to hold the reference to the logger object.

   ```
   var logger;
   ```

2. In the `init()` function, get the logger from the session object and assign it to your logger variable.

   ```
   function init(rtSession) {
       session = rtSession;
       logger = session.getLogger();
   }
   ```

3. Call the appropriate function on the logger to output a message.

   Debug messages

```
logger.debug("This is my debug message...");
```

Informational messages

```
logger.info("This is my info message...");
```

Warning messages

```
logger.warn("This is my warn message...");
```

Error messages

```
logger.error("This is my error message...");
```

Fatal error messages

```
logger.fatal("This is my fatal error message...");
```

Customer experience fatal error messages

```
logger.cxfatal("This is my customer experience fatal error message...");
```

For an example of the logging statements in a script, see Realtime Servers script example.

The output in the log files indicates the type of message (DEBUG, INFO, WARN, ERROR, FATAL, CXFATAL), as shown in the following lines from an example log:

```
09 Sep 2021 11:46:32,970 [INFO] (gamelift.js) 215: Calling GameLiftServerAPI.InitSDK...
09 Sep 2021 11:46:32,993 [INFO] (gamelift.js) 220: GameLiftServerAPI.InitSDK succeeded
09 Sep 2021 11:46:32,993 [INFO] (gamelift.js) 223: Waiting for Realtime server to
 start...
09 Sep 2021 11:46:33,15 [WARN] (index.js) 204: Connection is INSECURE. Messages will be
 sent/received as plaintext.
```

Accessing server logs

When a game session ends, Amazon GameLift automatically stores the logs in Amazon S3 and retains them for 14 days. You can use the GetGameSessionLogUrl API call to get the location of the logs for a game session. Use URL returned by the API call to download the logs.

Adjusting the logging level

Logs can grow very quickly and exceed the size limit. You should monitor your logs and limit the log output to necessary messages only. For Realtime Servers, you can adjust the logging level by providing a parameter in your fleet's runtime configuration in the form `loggingLevel:`*`LOGGING_LEVEL`*, where *`LOGGING_LEVEL`* is one of the following values:

1. `debug`
2. `info` *(default)*
3. `warn`
4. `error`
5. `fatal`
6. `cxfatal`

This list is ordered from least severe (`debug`) to most severe (`cxfatal`). You set a single `loggingLevel` and the server will only log messages at that severity level or a higher severity level. For example, setting `loggingLevel:error` will make all of the servers in your fleet only write `error`, `fatal`, and `cxfatal` messages to the log.

You can set the logging level for your fleet when you create it or after it is running. Changing your fleet's logging level after it is running will only affect logs for game sessions created after the update. Logs for any existing game sessions won't be affected. If you don't set a logging level when you create your fleet, your servers will set the logging level to `info` by default. Refer to the following sections for instructions to set the logging level.

Setting the logging level when creating a Realtime Servers fleet (Console)

Follow the instructions at Create a Amazon GameLift managed fleet to create your fleet, with the following addition:

- In the **Server process allocation** substep of the **Process management** step, provide the logging level key-value pair (such as `loggingLevel:error`) as a value for **Launch parameters**. Use a

non-alphanumeric character (except comma) to separate the logging level from any additional parameters (for example, `loggingLevel:error +map Winter444`).

Setting the logging level when creating a Realtime Servers fleet (AWS CLI)

Follow the instructions at Create a Amazon GameLift managed fleet to create your fleet, with the following addition:

- In the argument to the `--runtime-configuration` parameter for `create-fleet`, provide the logging level key-value pair (such as `loggingLevel:error`) as a value for `Parameters`. Use a non-alphanumeric character (except comma) to separate the logging level from any additional parameters. See the following example:

```
--runtime-configuration "GameSessionActivationTimeoutSeconds=60,
                    MaxConcurrentGameSessionActivations=2,
                    ServerProcesses=[{LaunchPath=/local/game/myRealtimeLaunchScript.js,
                                      Parameters=loggingLevel:error +map Winter444,
                                      ConcurrentExecutions=10}]"
```

Setting the logging level for a running Realtime Servers fleet (Console)

Follow the instructions at Update a fleet configuration to update your fleet using the Amazon GameLift Console, with the following addition:

- On the **Edit fleet** page, under **Server process allocation**, provide the logging level key-value pair (such as `loggingLevel:error`) as a value for **Launch parameters**. Use a non-alphanumeric character (except comma) to separate the logging level from any additional parameters (for example, `loggingLevel:error +map Winter444`).

Setting the logging level for a running Realtime Servers fleet (AWS CLI)

Follow the instructions at Update a fleet configuration to update your fleet using the AWS CLI, with the following addition:

- In the argument to the `--runtime-configuration` parameter for `update-runtime-configuration`, provide the logging level key-value pair (such as `loggingLevel:error`) as a value for `Parameters`. Use a non-alphanumeric character (except comma) to separate the logging level from any additional parameters. See the following example:

```
--runtime-configuration "GameSessionActivationTimeoutSeconds=60,
                   MaxConcurrentGameSessionActivations=2,
                   ServerProcesses=[{LaunchPath=/local/game/myRealtimeLaunchScript.js,
                                   Parameters=loggingLevel:error +map Winter444,
                                   ConcurrentExecutions=10}]"
```

Security in Amazon GameLift

If you're using Amazon GameLift FleetIQ as a standalone feature with Amazon EC2, see 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 Amazon GameLift. The following topics show you how to configure Amazon 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 Amazon GameLift resources.

Topics

- Security best practices for Amazon GameLift

Data protection in Amazon GameLift

If you're using Amazon GameLift FleetIQ as a standalone feature with Amazon EC2, see 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. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see 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 users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-2 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 text fields such as a **Name** field. This includes when you work with Amazon GameLift or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Amazon GameLift-specific data is handled as follows:

- Game server builds and scripts that you upload to Amazon 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 Amazon 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 Amazon GameLift console or by calls to the service API.
- Metric and event data is stored in Amazon GameLift and can be accessed through the Amazon 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 Amazon 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 Amazon 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 Amazon GameLift in an encrypted format.

Encryption in transit

Connections to the Amazon 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 automatically). Authentication is managed using the IAM-defined access policies for the security credentials that are used to make the connection.

Direct communication between game clients and game servers is as follows:

- For custom game servers being hosted on Amazon GameLift resources, communication does not involve the Amazon GameLift service. Encryption of this communication is the responsibility of the customer. You can use TLS-enabled fleets to have your game clients authenticate the game server on connection and to encrypt all communication between your game client and game server.
- For Realtime Servers with TLS certificate generation enabled, traffic between game client and Realtime servers using the Realtime Client SDK is encrypted in flight. TCP traffic is encrypted using TLS 1.2, and UDP traffic is encrypted using DTLS 1.2.

Internetwork traffic privacy

You can remotely access your Amazon 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 Amazon 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 Amazon GameLift instance, see Remotely connect to Amazon GameLift fleet instances.

Identity and access management for Amazon 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 Amazon GameLift resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience
- Authenticating with identities
- Managing access using policies
- How Amazon GameLift works with IAM

- Identity-based policy examples for Amazon GameLift
- Troubleshooting Amazon GameLift identity and access

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in Amazon GameLift.

Service user – If you use the Amazon GameLift service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more Amazon 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 Amazon GameLift, see Troubleshooting Amazon GameLift identity and access.

Service administrator – If you're in charge of Amazon GameLift resources at your company, you probably have full access to Amazon GameLift. It's your job to determine which Amazon GameLift features and resources your service users 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 Amazon GameLift, see How Amazon GameLift works with IAM.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to Amazon GameLift. To view example Amazon GameLift identity-based policies that you can use in IAM, see Identity-based policy examples for Amazon GameLift.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be *authenticated* (signed in to AWS) as the AWS account root user, as an IAM user, or by assuming an IAM role.

You can sign in to AWS as a federated identity by using credentials provided through an identity source. AWS IAM Identity Center (IAM Identity Center) users, your company's single sign-on authentication, and your Google or Facebook credentials are examples of federated identities. When you sign in as a federated identity, your administrator previously set up identity federation using IAM roles. When you access AWS by using federation, you are indirectly assuming a role.

Depending on the type of user you are, you can sign in to the AWS Management Console or the AWS access portal. For more information about signing in to AWS, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

If you access AWS programmatically, AWS provides a software development kit (SDK) and a command line interface (CLI) to cryptographically sign your requests by using your credentials. If you don't use AWS tools, you must sign requests yourself. For more information about using the recommended method to sign requests yourself, see Signing AWS API requests in the *IAM User Guide*.

Regardless of the authentication method that you use, you might 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 Multi-factor authentication in the *AWS IAM Identity Center User Guide* and Using multi-factor authentication (MFA) in AWS in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one 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 don't use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users, including users that require administrator access, to use federation with an identity provider to access AWS services by using temporary credentials.

A *federated identity* is a user from your enterprise user directory, a web identity provider, the AWS Directory Service, the Identity Center directory, or any user that accesses AWS services by using credentials provided through an identity source. When federated identities access AWS accounts, they assume roles, and the roles provide temporary credentials.

For centralized access management, we recommend that you use AWS IAM Identity Center. You can create users and groups in IAM Identity Center, or you can connect and synchronize to a set of users and groups in your own identity source for use across all your AWS accounts and applications. For

information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. Where possible, we recommend relying on temporary credentials instead of creating IAM users who have long-term credentials such as passwords and access keys. However, if you have specific use cases that require long-term credentials with IAM users, we recommend that you rotate access keys. For more information, see Rotate access keys regularly for use cases that require long-term credentials in the *IAM User Guide*.

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:

- **Federated user access** – To assign permissions to a federated identity, you create a role and define permissions for the role. When a federated identity authenticates, the identity is associated with the role and is granted the permissions that are defined by the role. For information about roles for federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. If you use IAM Identity Center, you configure a permission set. To control what your identities can access after they authenticate, IAM Identity Center correlates the permission set to a role in IAM. For information about permissions sets, see Permission sets in the *AWS IAM Identity Center User Guide*.

- **Temporary IAM user permissions** – An IAM user or role can assume an IAM role to temporarily take on different permissions for a specific task.
- **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.
  - **Forward access sessions (FAS)** – When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.
  - **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 AWS 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 AWS identities or resources. A policy is an object in AWS that, when associated with an identity or resource, defines their permissions. AWS evaluates these policies when a principal (user, root user, or role session) makes a request. 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**.

By default, users and roles have no permissions. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

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 an 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 Amazon GameLift works with IAM

Before you use IAM to manage access to Amazon GameLift, learn what IAM features are available to use with Amazon GameLift.

IAM features you can use with Amazon GameLift

IAM feature	Amazon GameLift support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Yes

IAM feature	Amazon GameLift support
Temporary credentials	Yes
Principal permissions	Yes
Service roles	Yes
Service-linked roles	No

To get a high-level view of how Amazon GameLift and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for Amazon GameLift

Supports identity-based policies	Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Creating IAM policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. You can't specify the principal in an identity-based policy because it applies to the user or role to which it is attached. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Amazon GameLift

To view examples of Amazon GameLift identity-based policies, see Identity-based policy examples for Amazon GameLift.

Resource-based policies within Amazon GameLift

Supports resource-based policies	No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. Adding a cross-account principal to a resource-based policy is only half of establishing the trust relationship. When the principal and the resource are in different AWS accounts, an IAM administrator in the trusted account must also grant the principal entity (user or role) permission to access the resource. They grant permission by attaching an identity-based policy to the entity. However, if a resource-based policy grants access to a principal in the same account, no additional identity-based policy is required. For more information, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Policy actions for Amazon GameLift

Supports policy actions	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. 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.

For a list of Amazon GameLift actions, see Actions defined by Amazon GameLift in the *Service Authorization Reference*.

Policy actions in Amazon GameLift use the following prefix before the action:

```
gamelift
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "gamelift:action1",
      "gamelift:action2"
         ]
```

You can specify multiple actions using wildcards (*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "gamelift:Describe*"
```

To view examples of Amazon GameLift identity-based policies, see Identity-based policy examples for Amazon GameLift.

Policy resources for Amazon GameLift

Supports policy resources	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. 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": "*"
```

For a list of Amazon GameLift resource types and their ARNs, see Resources defined by Amazon GameLift in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by Amazon GameLift.

Some Amazon GameLift resources have ARN values, which allows the resources to have their access managed using IAM policies. The Amazon 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) in the *AWS General Reference*.

For example, to specify the `fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa` fleet in your statement, use the following ARN:

```
"Resource": "arn:aws:gamelift:us-west-2:123456789012:fleet/
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa"
```

To specify all fleets that belong to a specific account, use a wildcard (*):

```
"Resource": "arn:aws:gamelift:us-west-2:123456789012:fleet/*"
```

To view examples of Amazon GameLift identity-based policies, see Identity-based policy examples for Amazon GameLift.

Policy condition keys for Amazon GameLift

Supports service-specific policy condition keys	Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element (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*.

For a list of Amazon GameLift condition keys, see Condition keys for Amazon GameLift in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by Amazon GameLift.

To view examples of Amazon GameLift identity-based policies, see Identity-based policy examples for Amazon GameLift.

ACLs in Amazon GameLift

Supports ACLs	No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with Amazon GameLift

Supports ABAC (tags in policies)	Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. In AWS, these attributes are called *tags*. You can attach tags to IAM entities (users or roles) and to many AWS resources. Tagging entities and resources is the first step of ABAC. Then you design ABAC policies to allow operations when the principal's tag matches the tag on the resource that they are trying to access.

ABAC is helpful in environments that are growing rapidly and helps with situations where policy management becomes cumbersome.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see What is ABAC? in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

For an example identity-based policy that limits access to a resource based on the tags on that resource, see View Amazon GameLift fleets based on tags.

Using temporary credentials with Amazon GameLift

Supports temporary credentials	Yes

Some AWS services don't work when you sign in using temporary credentials. For additional information, including which AWS services work with temporary credentials, see AWS services that work with IAM in the *IAM User Guide*.

You are using temporary credentials if you sign in to the AWS Management Console using any method except a user name and password. For example, when you access AWS using your company's single sign-on (SSO) link, that process automatically creates temporary credentials. You also automatically create temporary credentials when you sign in to the console as a user and then switch roles. For more information about switching roles, see Switching to a role (console) in the *IAM User Guide*.

You can manually create temporary credentials using the AWS CLI or AWS API. You can then use those temporary credentials to access AWS. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM.

Cross-service principal permissions for Amazon GameLift

Supports forward access sessions (FAS)	Yes

When you use an IAM user or role to perform actions in AWS, you are considered a principal. When you use some services, you might perform an action that then initiates another action in a different service. FAS uses the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. FAS requests are only made when a service receives a request that requires interactions with other AWS services or resources to complete. In this case, you must have permissions to perform both actions. For policy details when making FAS requests, see Forward access sessions.

Service roles for Amazon GameLift

Supports service roles	Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break Amazon GameLift functionality. Edit service roles only when Amazon GameLift provides guidance to do so.

Allow your Amazon 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 Amazon GameLift manages, you need a service role that gives Amazon GameLift limited access to your other AWS resources. For more information, see Communicate with other AWS resources from your fleets.

Service-linked roles for Amazon GameLift

Supports service-linked roles	No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM in the *IAM User Guide*. Find a service in the table that includes a `Yes` in the **Service-linked roles** column. Choose **Yes** to view the service-linked role documentation for that service.

Identity-based policy examples for Amazon GameLift

By default, users and roles don't have permission to create or modify Amazon GameLift resources. They also can't perform tasks by using the AWS Management Console, AWS Command Line Interface (AWS CLI), or AWS API. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies. The administrator can then add the IAM policies to roles, and users can assume the roles.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Creating IAM policies in the *IAM User Guide*.

For details about actions and resource types defined by Amazon GameLift, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for Amazon GameLift in the *Service Authorization Reference*.

Topics

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon 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 with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We

recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.

- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as AWS CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see IAM Access Analyzer policy validation in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Configuring MFA-protected API access in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the Amazon GameLift console

To access the Amazon GameLift console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon 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 (users or roles) with that policy.

To ensure that those entities can still use the Amazon GameLift console, add permissions to users and groups with the syntax in the following examples and in Administrator permission examples. For more information, see Manage user permissions for Amazon GameLift.

Users that work with Amazon GameLift through AWS CLI or AWS API operations don't require minimum console permissions. Instead, you can limit access to only the operations the user needs to perform. For example, a player user, acting on behalf of game clients, requires access to request game sessions, place players into games, and other tasks.

For information about the permissions required to use all Amazon GameLift console features, see permissions syntax for administrators in Administrator permission examples.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
```

```
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Allow player access for game sessions

To place players into game sessions, game clients and backend services need permissions. For policy examples for these scenarios, see Player user permission examples.

Allow access to one Amazon GameLift queue

The following example provides a user with access to a specific Amazon GameLift queues.

This policy grants the user permissions to add, update, and delete queue destinations with the following actions: `gamelift:UpdateGameSessionQueue`, `gamelift:DeleteGameSessionQueue`, and `gamelift:DescribeGameSessionQueues`. As shown, this policy uses the `Resource` element to limit access to a single queue: `gamesessionqueue/examplequeue123`.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ViewSpecificQueueInfo",
         "Effect":"Allow",
         "Action":[
            "gamelift:DescribeGameSessionQueues"
         ],
         "Resource":"arn:aws:gamelift:::gamesessionqueue/examplequeue123"
      },
      {
         "Sid":"ManageSpecificQueue",
         "Effect":"Allow",
         "Action":[
            "gamelift:UpdateGameSessionQueue",
            "gamelift:DeleteGameSessionQueue"
         ],
```

```
         "Resource":"arn:aws:gamelift:::gamesessionqueue/examplequeue123"
      }
   ]
}
```

View Amazon GameLift fleets based on tags

You can use conditions in your identity-based policy to control access to Amazon GameLift resources based on tags. This example shows how you can create a policy that allows viewing a fleet if the `Owner` tag matches the user's user name. This policy also grants the permissions necessary to complete this operation in the console.

```
{
    "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}"}
            }
        }
    ]
}
```

Access a game build file in Amazon S3

After you integrate your game server with Amazon GameLift, upload the build files to Amazon S3. For Amazon GameLift to access the build files, use the following policy.

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

```
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::bucket-name/object-name"
        }
    ]
}
```

For more information about uploading Amazon GameLift game files, see Upload a custom server build to Amazon GameLift.

Troubleshooting Amazon GameLift identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon GameLift and AWS Identity and Access Management (IAM).

Topics

- I am not authorized to perform an action in Amazon GameLift
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my Amazon GameLift resources

I am not authorized to perform an action in Amazon GameLift

If the AWS Management Console tells you that you're not authorized to perform an action, contact your AWS account administrator for assistance. Your administrator is the person that provided you with your sign-in credentials.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a queue but doesn't have `gamelift:DescribeGameSessionQueues` permissions:

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 gamelift:DescribeGameSessionQueues on resource: examplequeue123
```

In this case, Mateo asks his administrator to update his policies to allow him read access for the `examplequeue123` resource using the `gamelift:DescribeGameSessionQueues` action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to Amazon GameLift.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon GameLift. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my Amazon GameLift resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon GameLift supports these features, see How Amazon GameLift works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.

- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see How IAM roles differ from resource-based policies in the *IAM User Guide*.

Logging and monitoring with Amazon GameLift

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon 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 Amazon 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 number of periods, not by being in a particular state. For more information, see Monitor Amazon GameLift with Amazon CloudWatch.

AWS CloudTrail Logs

CloudTrail provides a record of actions taken by a user, role, or an AWS service in Amazon GameLift. Using the information collected by CloudTrail, you can determine the request that was made to Amazon GameLift, the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, see Logging Amazon GameLift API calls with AWS CloudTrail.

Compliance validation for Amazon GameLift

Amazon GameLift is not in scope of any AWS compliance programs.

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. 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 on Amazon Web Services – This whitepaper describes how companies can use AWS to create HIPAA-eligible applications.

  > **Note**
  >
  > Not all AWS services are HIPAA eligible. For more information, see the HIPAA Eligible Services Reference.

- AWS Compliance Resources – This collection of workbooks and guides might apply to your industry and location.
- AWS Customer Compliance Guides – Understand the shared responsibility model through the lens of compliance. The guides summarize the best practices for securing AWS services and map the guidance to security controls across multiple frameworks (including National Institute of Standards and Technology (NIST), Payment Card Industry Security Standards Council (PCI), and International Organization for Standardization (ISO)).
- 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. Security Hub uses security controls to evaluate your AWS resources and to check your compliance against security industry standards and best practices. For a list of supported services and controls, see Security Hub controls reference.
- 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 Amazon GameLift

If you're using Amazon GameLift FleetIQ as a standalone feature with Amazon EC2, see 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, Amazon GameLift offers the following features to help support your data resiliency needs:

- **Multi-region queues** – Amazon 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.
- **Automatic capacity scaling** – Maintain the health and availability of your hosting resources by using Amazon 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 Amazon GameLift hosting capacity.
- **Distribution across instances** – Amazon 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 Amazon 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 Amazon GameLift

If you're using Amazon GameLift FleetIQ as a standalone feature with Amazon EC2, see Security in Amazon EC2 in the *Amazon EC2 User Guide for Linux Instances*.

As a managed service, Amazon 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 Amazon GameLift through the network. Clients must support Transport Layer Security (TLS) 1.2 or later. We recommend TLS 1.3 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

The Amazon GameLift service places all fleets into Amazon virtual private clouds (VPCs) so that each fleet exists in a logically isolated area in the AWS Cloud. You can use Amazon GameLift policies to control access from specific VPC endpoints or specific VPCs. Effectively, this isolates network access to a given Amazon 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 Amazon GameLift

If you're using Amazon GameLift FleetIQ as a standalone feature with Amazon EC2, see 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 Amazon GameLift:

- Customers are responsible for the management of software that is deployed to Amazon 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 Amazon 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 Amazon GameLift Server SDK, and the Amazon GameLift Client SDK for Realtime Servers.

Security best practices for Amazon GameLift

If you're using Amazon GameLift FleetIQ as a standalone feature with Amazon EC2, see Security in Amazon EC2 in the *Amazon EC2 User Guide for Linux Instances*.

Amazon 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.

For more information about how you can make your use of Amazon GameLift more secure, see the AWS Well-Architected Tool Security pillar..

Amazon GameLift reference guides

This section contains reference documentation for using Amazon GameLift.

Topics

Amazon GameLift service API reference (AWS SDK)

This topic provides a task-based list of API operations for use with Amazon 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 Amazon GameLift hosting resources
- Start game sessions and join players

The Amazon GameLift Service API also contains operations for use with other Amazon 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 Amazon 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 Amazon GameLift, including Realtime Servers. You can use the Amazon 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 Amazon 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 Amazon GameLift region.
- DescribeBuild – Retrieve information associated with a build.
- UpdateBuild – Change build metadata, including build name and version.
- DeleteBuild – Remove a build from Amazon GameLift.

Manage Realtime Servers configuration scripts

- CreateScript – Upload JavaScript files and create a new Amazon GameLift script resource.
- ListScripts – Get a list of all Realtime scripts uploaded to a Amazon 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 Amazon 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 Amazon 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 Amazon 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 Amazon 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 Amazon 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 Amazon 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 Amazon GameLift fleet and one of your VPCs.
- DescribeVpcPeeringConnections – Retrieve information on active or pending VPC peering connections with a Amazon GameLift fleet.
- DeleteVpcPeeringConnection – Delete a VPC peering connection with a Amazon 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 Amazon GameLift. If you're using Realtime Servers, manage game sessions using the Realtime Servers client API (C#) reference.

- **Start new game sessions for one or more players.**
  - StartGameSessionPlacement – Ask Amazon 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.
- **Get players into existing game sessions.** Find running game sessions with available player slots and reserve them for new players.
  - CreatePlayerSession – Reserve an open slot for a player to join a game session.
  - CreatePlayerSessions – Reserve open slots for multiple players to join a game session.
- **Work with game session and player session data.** Manage information on game sessions and player sessions.
  - SearchGameSessions – Request a list of active game sessions based on a set of search criteria.
  - DescribeGameSessions – Retrieve metadata for specific game sessions, including length of time active and current player count.
  - DescribeGameSessionDetails – Retrieve metadata, including the game session protection setting, for one or more game sessions.
  - DescribePlayerSessions – Get details on player activity, including status, playing time, and player data.
  - UpdateGameSession – Change game session settings, such as maximum player count and join policy.
  - GetGameSessionLogUrl – Get the location of saved logs for a game session.

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
- Amazon GameLift Realtime Servers script reference

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 Prepare your Realtime server. 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
- Asynchronous Callbacks
- Data Types

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 `GameLiftRealtimeClientSdkNet45.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 Prepare your Realtime server.

- Synchronous Actions
- Asynchronous Callbacks
- Data Types

Client()

Initializes a new client to communicate with the Realtime server and identifies the type of connection to use.

Syntax

```
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

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

```
public ConnectionStatus Connect(string endpoint, int remoteTcpPort, int listenPort,
 ConnectionToken token)
```

Parameters

endpoint

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.

> **Note**
>
> If the Realtime server is running on a fleet with a TLS certificate, you must use the DNS name.

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

Required: Yes

Return value

Returns a ConnectionStatus 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 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

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().

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

```
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().

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() callback.

Syntax

```
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 Prepare your Realtime server.

- Synchronous Actions
- Asynchronous Callbacks
- Data Types

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.

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

```
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

```
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

```
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

```
public void OnDataReceived(DataReceivedEventArgs dataReceivedEventArgs)
```

Parameters

dataReceivedEventArgs

Information related to message activity.

Type: DataReceivedEventArgs

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

```
public void OnGroupMembershipUpdated(GroupMembershipEventArgs groupMembershipEventArgs)
```

Parameters

groupMembershipEventArgs

Information related to group membership activity.

Type: GroupMembershipEventArgs

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 Prepare your Realtime server.

- Synchronous Actions
- Asynchronous Callbacks
- 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 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 Amazon 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: Yes

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.

Type: Integer

Required: Yes

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: No

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().

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() 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.

Type: Integer

Required: Yes

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() 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()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
- Realtime Servers interface

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

```
init(rtsession)
```

onMessage

Invoked when a received message is sent to the server.

Syntax

```
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

```
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

```
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

```
onSendToGroup(gameMessage))
```

onPlayerJoinGroup

Invoked when a player sends a request to join a group.

Syntax

```
onPlayerJoinGroup(groupId, peerId)
```

onPlayerLeaveGroup

Invoked when a player sends a request to leave a group.

Syntax

```
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.

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 for an example of user-defined group IDs.

Syntax

```
rtSession.broadcastGroupMembershipUpdate(groupIdToBroadcast, targetGroupId)
```

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()
```

getGameSessionId()

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: `logger.info("<string>")`.

Syntax

```
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

```
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 for an example of user-defined group IDs.

Syntax

```
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

```
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 for an example of user-defined group IDs.

Syntax

```
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). Returns a `gameMessage` object.

Syntax

```
rtSession.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). Returns a `gameMessage` object.

Syntax

```
rtSession.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 structure.

getPayloadAsText()

Gets the game message payload as text.

Syntax

```
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 servers to communicate with the Amazon GameLift service.

Topics

- Amazon GameLift server SDK reference for C++
- Amazon GameLift server SDK reference for C#
- Amazon GameLift server SDK reference for Go
- Amazon GameLift server SDK reference for Unreal Engine

Amazon GameLift server SDK reference for C++

You can use this Amazon GameLift C++ server SDK reference to help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server.

Topics

- Amazon GameLift server SDK 5.x reference for C++
- Amazon GameLift C++ server SDK 3.x reference

Amazon GameLift server SDK 5.x reference for C++

This Amazon GameLift C++ Server SDK 5.x reference can help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server.

> **Note**
>
> This topic describes the Amazon GameLift C++ API that you can use when you build with the C++ Standard Library (`std`). Specifically, this documentation applies to code that you compile with the `-DGAMELIFT_USE_STD=1` option.

Topics

- Amazon GameLift server SDK (C++) 5.x reference: Actions
- Amazon GameLift server SDK (C++) reference: Data types

Amazon GameLift server SDK (C++) 5.x reference: Actions

You can use this Amazon GameLift C++ server SDK reference to help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server.

> **Note**
>
> This topic describes the Amazon GameLift C++ API that you can use when you build with the C++ Standard Library (`std`). Specifically, this documentation applies to code that you compile with the `-DGAMELIFT_USE_STD=1` option.

Actions

- GetSdkVersion()
- InitSDK()
- InitSDK()
- ProcessReady()
- ProcessReadyAsync()

- ProcessEnding()
- ActivateGameSession()
- UpdatePlayerSessionCreationPolicy()
- GetGameSessionId()
- GetTerminationTime()
- AcceptPlayerSession()
- RemovePlayerSession()
- DescribePlayerSessions()
- StartMatchBackfill()
- StopMatchBackfill()
- GetComputeCertificate()
- GetFleetRoleCredentials()
- Destroy()

GetSdkVersion()

Returns the current version number of the SDK built into the server process.

Syntax

```
Aws::GameLift::AwsStringOutcome Server::GetSdkVersion();
```

Return value

If successful, returns the current SDK version as an the section called "AwsStringOutcome" object. The returned object includes the version number (example `5.0.0`). If not successful, returns an error message.

Example

```
Aws::GameLift::AwsStringOutcome SdkVersionOutcome =
 Aws::GameLift::Server::GetSdkVersion();
```

InitSDK()

Initializes the Amazon GameLift SDK for a managed EC2 fleet. Call this method on launch, before any other initialization related to Amazon GameLift occurs. This method reads server parameters

from the host environment to set up communication between the server and the Amazon GameLift service.

Syntax

```
Server::InitSDKOutcome Server::initSdkOutcome = InitSDK();
```

Return value

Returns an the section called "InitSDKOutcome" object that indicates whether the server process is ready to call ProcessReady().

Example

```
//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
Aws::GameLift::Server::InitSDKOutcome initSdkOutcome =
  Aws::GameLift::Server::InitSDK();
```

InitSDK()

Initializes the Amazon GameLift SDK for an Anywhere fleet. Call this method on launch, before any other initialization related to Amazon GameLift occurs. This method requires explicit server parameters to set up communication between the server and the Amazon GameLift service.

Syntax

```
Server::InitSDKOutcome Server::initSdkOutcome = InitSDK(serverParameters);
```

Parameters

ServerParameters

To initialize a game server on an Amazon GameLift Anywhere fleet, construct a `ServerParameters` object with the following information:

- The URL of the WebSocket used to connect to your game server.
- The ID of the process used to host your game server.
- The ID of the compute hosting your game server processes.
- The ID of the Amazon GameLift fleet containing your Amazon GameLift Anywhere compute.

- The authorization token generated by the Amazon GameLift operation.

Return value

Returns an the section called “InitSDKOutcome” object that indicates whether the server process is ready to call ProcessReady().

> **Note**
>
> If calls to `InitSDK()` are failing for game builds deployed to Anywhere fleets, check the `ServerSdkVersion` parameter used when creating the build resource. You must explicitly set this value to the server SDK version in use. The default value for this parameter is 4.x, which is not compatible. To resolve this issue, create a new build and deploy it to a new fleet.

Example

Amazon GameLift Anywhere example

```
//Define the server parameters
std::string websocketUrl = "wss://us-west-1.api.amazongamelift.com";
std::string processId = "PID1234";
std::string fleetId = "arn:aws:gamelift:us-west-1:111122223333:fleet/
fleet-9999ffff-88ee-77dd-66cc-5555bbbb44aa";
std::string hostId = "HardwareAnywhere";
std::string authToken = "1111aaaa-22bb-33cc-44dd-5555eeee66ff";
Aws::GameLift::Server::Model::ServerParameters serverParameters =
  Aws::GameLift::Server::Model::ServerParameters(webSocketUrl, authToken, fleetId,
 hostId, processId);

//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
Aws::GameLift::Server::InitSDKOutcome initSdkOutcome =
 Aws::GameLift::Server::InitSDK(serverParameters);
```

ProcessReady()

Notifies Amazon GameLift that the server process is ready to host game sessions. Call this method after invoking InitSDK(). This method should be called only once per process.

Syntax

```
GenericOutcome ProcessReady(const Aws::GameLift::Server::ProcessParameters
&processParameters);
```

Parameters

processParameters

A ProcessParameters object communicating the following information about the server process:

- Names of callback methods implemented in the game server code that the Amazon GameLift service invokes to communicate with the server process.
- Port number that the server process is listening on.
- Path to any game session-specific files that you want Amazon GameLift to capture and store.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example illustrates both the ProcessReady() call and delegate function implementations.

```
// Set parameters and call ProcessReady
std::string serverLog("serverOut.log");        // Example of a log file written by the
 game server
std::vector<std::string> logPaths;
logPaths.push_back(serverLog);
int listenPort = 9339;

Aws::GameLift::Server::ProcessParameters processReadyParameter =
  Aws::GameLift::Server::ProcessParameters(
    std::bind(&Server::onStartGameSession, this, std::placeholders::_1),
    std::bind(&Server::onProcessTerminate, this),
    std::bind(&Server::OnHealthCheck, this),
    std::bind(&Server::OnUpdateGameSession, this),
    listenPort,
    Aws::GameLift::Server::LogParameters(logPaths)
  );

Aws::GameLift::GenericOutcome outcome =
  Aws::GameLift::Server::ProcessReady(processReadyParameter);
```

```
// Implement callback functions
void Server::onStartGameSession(Aws::GameLift::Model::GameSession myGameSession)
{
  // game-specific tasks when starting a new game session, such as loading map
  GenericOutcome outcome =
    Aws::GameLift::Server::ActivateGameSession (maxPlayers);
}

void Server::onProcessTerminate()
{
  // game-specific tasks required to gracefully shut down a game session,
  // such as notifying players, preserving game state data, and other cleanup
  GenericOutcome outcome = Aws::GameLift::Server::ProcessEnding();
}

bool Server::onHealthCheck()
{
  bool health;
  // complete health evaluation within 60 seconds and set health
  return health;
}
```

ProcessReadyAsync()

Notifies the Amazon GameLift service that the server process is ready to host game sessions. This method should be called after the server process is ready to host a game session. The parameters specify the callback function names for Amazon GameLift to call in certain circumstances. Game server code must implement these functions.

This call is asynchronous. To make a synchronous call, use ProcessReady(). See Initialize the server process for more details.

Syntax

```
GenericOutcomeCallable ProcessReadyAsync(
    const Aws::GameLift::Server::ProcessParameters &processParameters);
```

Parameters

processParameters

A ProcessParameters object communicating the following information about the server process:

- Names of callback methods implemented in the game server code that the Amazon GameLift service invokes to communicate with the server process.
- Port number that the server process is listening on.
- Path to any game session-specific files that you want Amazon GameLift to capture and store.

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
// Set parameters and call ProcessReady
std::string serverLog("serverOut.log");        // This is an example of a log file
 written by the game server
std::vector<std::string> logPaths;
logPaths.push_back(serverLog);
int listenPort = 9339;

Aws::GameLift::Server::ProcessParameters processReadyParameter =
  Aws::GameLift::Server::ProcessParameters(std::bind(&Server::onStartGameSession, this,
 std::placeholders::_1),
  std::bind(&Server::onProcessTerminate, this), std::bind(&Server::OnHealthCheck,
 this),
  std::bind(&Server::OnUpdateGameSession, this), listenPort,
 Aws::GameLift::Server::LogParameters(logPaths));

Aws::GameLift::GenericOutcomeCallable outcome =
  Aws::GameLift::Server::ProcessReadyAsync(processReadyParameter);

// Implement callback functions
void onStartGameSession(Aws::GameLift::Model::GameSession myGameSession)
{
  // game-specific tasks when starting a new game session, such as loading map
  GenericOutcome outcome = Aws::GameLift::Server::ActivateGameSession (maxPlayers);
}

void onProcessTerminate()
{
  // game-specific tasks required to gracefully shut down a game session,
  // such as notifying players, preserving game state data, and other cleanup
```

```
  GenericOutcome outcome = Aws::GameLift::Server::ProcessEnding();
}

bool onHealthCheck()
{
  // perform health evaluation and complete within 60 seconds
  return health;
}
```

ProcessEnding()

Notifies Amazon GameLift that the server process is terminating. Call this method after all other cleanup tasks (including shutting down the active game session) and before terminating the process. Depending on the result of `ProcessEnding()`, the process exits with success (0) or error (-1) and generates a fleet event. If the process terminates with an error, the fleet event generated is `SERVER_PROCESS_TERMINATED_UNHEALTHY`.

Syntax

```
Aws::GameLift::GenericOutcome processEndingOutcome =
 Aws::GameLift::Server::ProcessEnding();
```

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example calls `ProcessEnding()` and `Destroy()` before terminating the server process with a success or error exit code.

```
Aws::GameLift::GenericOutcome processEndingOutcome =
 Aws::GameLift::Server::ProcessEnding();
Aws::GameLift::Server::Destroy();

// Exit the process with success or failure
if (processEndingOutcome.IsSuccess()) {
  exit(0);
}
else {
  cout << "ProcessEnding() failed. Error: " <<
 processEndingOutcome.GetError().GetErrorMessage();
```

```
  exit(-1);
}
```

ActivateGameSession()

Notifies Amazon GameLift that the server process has activated a game session and is now ready to receive player connections. This action should be called as part of the onStartGameSession() callback function, after all game session initialization.

Syntax

```
Aws::GameLift::GenericOutcome activateGameSessionOutcome =
 Aws::GameLift::Server::ActivateGameSession();
```

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example shows ActivateGameSession() called as part of the onStartGameSession() delegate function.

```
void onStartGameSession(Aws::GameLift::Model::GameSession myGameSession)
{
  // game-specific tasks when starting a new game session, such as loading map
  GenericOutcome outcome = Aws::GameLift::Server::ActivateGameSession();
}
```

UpdatePlayerSessionCreationPolicy()

Updates the current game session's ability to accept new player sessions. A game session can be set to either accept or deny all new player sessions.

Syntax

```
GenericOutcome
 UpdatePlayerSessionCreationPolicy(Aws::GameLift::Model::PlayerSessionCreationPolicy
 newPlayerSessionPolicy);
```

Parameters

playerCreationSessionPolicy

Type: `PlayerSessionCreationPolicy` enum value.

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example sets the current game session's join policy to accept all players.

```
Aws::GameLift::GenericOutcome outcome =

 Aws::GameLift::Server::UpdatePlayerSessionCreationPolicy(Aws::GameLift::Model::PlayerSessionCr
```

GetGameSessionId()

Retrieves the ID of the game session hosted by the active server process.

For idle processes that aren't activated with a game session, the call returns a the section called “GameLiftError”.

Syntax

```
AwsStringOutcome GetGameSessionId()
```

Parameters

This action has no parameters.

Return value

If successful, returns the game session ID as an the section called “AwsStringOutcome” object. If not successful, returns an error message.

For idle processes that aren't activated with a game session, the call returns `Success=True` and `GameSessionId=""`.

Example

```
Aws::GameLift::AwsStringOutcome sessionIdOutcome =
 Aws::GameLift::Server::GetGameSessionId();
```

GetTerminationTime()

Returns the time that a server process is scheduled to be shut down, if a termination time is available. A server process takes action after receiving an onProcessTerminate() callback from Amazon GameLift. Amazon GameLift calls onProcessTerminate() for the following reasons:

- When the server process has reported poor health or has not responded to Amazon GameLift.
- When terminating the instance during a scale-down event.
- When an instance is terminated due to a spot-instance interruption.

Syntax

```
AwsDateTimeOutcome GetTerminationTime()
```

Return value

If successful, returns the termination time as an AwsDateTimeOutcome object. The value is the termination time, expressed in elapsed ticks since 0001 00:00:00. For example, the date time value 2020-09-13 12:26:40 -000Z is equal to 637355968000000000 ticks. If no termination time is available, returns an error message.

If the process hasn't received a ProcessParameters.OnProcessTerminate() callback, an error message is returned. For more information about shutting down a server process, see Respond to a server process shutdown notification.

Example

```
Aws::GameLift::AwsLongOutcome TermTimeOutcome =
 Aws::GameLift::Server::GetTerminationTime();
```

AcceptPlayerSession()

Notifies Amazon GameLift that a player with the specified player session ID has connected to the server process and needs validation. Amazon GameLift verifies that the player session ID is valid.

After the player session is validated, Amazon GameLift changes the status of the player slot from RESERVED to ACTIVE.

Syntax

```
GenericOutcome AcceptPlayerSession(String playerSessionId)
```

Parameters

playerSessionId

Unique ID issued by Amazon GameLift when a new player session is created.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example handles a connection request that includes validating and rejecting non-valid player session IDs.

```
void ReceiveConnectingPlayerSessionID (Connection& connection, const std::string&
 playerSessionId)
{
  Aws::GameLift::GenericOutcome connectOutcome =
 Aws::GameLift::Server::AcceptPlayerSession(playerSessionId);
  if(connectOutcome.IsSuccess())
  {
    connectionToSessionMap.emplace(connection, playerSessionId);
    connection.Accept();
  }
  else
  {
    connection.Reject(connectOutcome.GetError().GetMessage();
  }
}
```

RemovePlayerSession()

Notifies Amazon GameLift that a player has disconnected from the server process. In response, Amazon GameLift changes the player slot to available.

Syntax

```
GenericOutcome RemovePlayerSession(String playerSessionId)
```

Parameters

`playerSessionId`

Unique ID issued by Amazon GameLift when a new player session is created.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
Aws::GameLift::GenericOutcome disconnectOutcome =
 Aws::GameLift::Server::RemovePlayerSession(playerSessionId);
```

DescribePlayerSessions()

Retrieves player session data which includes settings, session metadata, and player data. Use this method to get information about the following:

- A single player session
- All player sessions in a game session
- All player sessions associated with a single player ID

Syntax

```
DescribePlayerSessionsOutcome DescribePlayerSessions(DescribePlayerSessionsRequest
 describePlayerSessionsRequest)
```

Parameters

DescribePlayerSessionsRequest

A the section called “DescribePlayerSessionsRequest” object that describes which player sessions to retrieve.

Return value

If successful, returns a the section called "DescribePlayerSessionsOutcome" object containing a set of player session objects that fit the request parameters.

Example

This example requests all player sessions actively connected to a specified game session. By omitting *NextToken* and setting the *Limit* value to 10, Amazon GameLift returns the first 10 player session records matching the request.

```
// Set request parameters
Aws::GameLift::Server::Model::DescribePlayerSessionsRequest request;
request.SetPlayerSessionStatusFilter(Aws::GameLift::Server::Model::PlayerSessionStatusMapper::G
request.SetLimit(10);
request.SetGameSessionId("the game session ID");    // can use GetGameSessionId()

// Call DescribePlayerSessions
Aws::GameLift::DescribePlayerSessionsOutcome playerSessionsOutcome =
 Aws::GameLift::Server::DescribePlayerSessions(request);
```

StartMatchBackfill()

Sends a request to find new players for open slots in a game session created with FlexMatch. For more information, see FlexMatch backfill feature.

This action is asynchronous. If new players are matched, Amazon GameLift delivers updated matchmaker data using the callback function `OnUpdateGameSession()`.

A server process can have only one active match backfill request at a time. To send a new request, first call StopMatchBackfill() to cancel the original request.

Syntax

```
StartMatchBackfillOutcome StartMatchBackfill (StartMatchBackfillRequest
 startBackfillRequest);
```

Parameters

StartMatchBackfillRequest

A StartMatchBackfillRequest object that communicates the following information:

- A ticket ID to assign to the backfill request. This information is optional; if no ID is provided, Amazon GameLift will generate one.
- The matchmaker to send the request to. The full configuration ARN is required. This value is in the game session's matchmaker data.
- The ID of the game session to backfill.
- The available matchmaking data for the game session's current players.

Return value

Returns a the section called "StartMatchBackfillOutcome" object with the match backfill ticket ID, or failure with an error message.

Example

```
// Build a backfill request
std::vector<Player> players;
Aws::GameLift::Server::Model::StartMatchBackfillRequest startBackfillRequest;
startBackfillRequest.SetTicketId("1111aaaa-22bb-33cc-44dd-5555eeee66ff");  // optional,
 autogenerated if not provided
startBackfillRequest.SetMatchmakingConfigurationArn("arn:aws:gamelift:us-
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig"); //from the game
 session matchmaker data
startBackfillRequest.SetGameSessionArn("the game session ARN");        // can use
 GetGameSessionId()
startBackfillRequest.SetPlayers(players);                                  // from the
 game session matchmaker data

// Send backfill request
Aws::GameLift::StartMatchBackfillOutcome backfillOutcome =
  Aws::GameLift::Server::StartMatchBackfill(startBackfillRequest);

// Implement callback function for backfill
void Server::OnUpdateGameSession(Aws::GameLift::Server::Model::GameSession gameSession,
 Aws::GameLift::Server::Model::UpdateReason updateReason, std::string backfillTicketId)
{
  // handle status messages
  // perform game-specific tasks to prep for newly matched players
}
```

StopMatchBackfill()

Cancels an active match backfill request. For more information, see FlexMatch backfill feature.

Syntax

```
GenericOutcome StopMatchBackfill (StopMatchBackfillRequest stopBackfillRequest);
```

Parameters

StopMatchBackfillRequest

A StopMatchBackfillRequest object identifying the matchmaking ticket to cancel:

- The ticket ID assigned to the backfill request.
- The matchmaker the backfill request was sent to.
- The game session associated with the backfill request.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
// Set backfill stop request parameters

Aws::GameLift::Server::Model::StopMatchBackfillRequest stopBackfillRequest;
stopBackfillRequest.SetTicketId("1111aaaa-22bb-33cc-44dd-5555eeee66ff");
stopBackfillRequest.SetGameSessionArn("the game session ARN"); // can use
 GetGameSessionId()
stopBackfillRequest.SetMatchmakingConfigurationArn("arn:aws:gamelift:us-
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig");
// from the game session matchmaker data

Aws::GameLift::GenericOutcome stopBackfillOutcome =
  Aws::GameLift::Server::StopMatchBackfill(stopBackfillRequest);
```

GetComputeCertificate()

Retrieves the path to the TLS certificate used to encrypt the network connection between your Amazon GameLift Anywhere compute resource and Amazon GameLift. You can use the certificate

path when you register your compute device to a Amazon GameLift Anywhere fleet. For more information see, RegisterCompute.

Syntax

```
GetComputeCertificateOutcome Server::GetComputeCertificate()
```

Return value

Returns a the section called “GetComputeCertificateOutcome”.

Example

```
Aws::GameLift::GetComputeCertificateOutcome certificate =
 Aws::GameLift::Server::GetComputeCertificate();
```

GetFleetRoleCredentials()

Retrieves IAM role credentials that authorize Amazon GameLift to interact with other AWS services. For more information, see Communicate with other AWS resources from your fleets.

Syntax

```
GetFleetRoleCredentialsOutcome GetFleetRoleCredentials(GetFleetRoleCredentialsRequest
 request);
```

Parameters

GetFleetRoleCredentialsRequest

Return value

Returns a the section called “GetFleetRoleCredentialsOutcome” object.

Example

```
// form the fleet credentials request
Aws::GameLift::Server::Model::GetFleetRoleCredentialsRequest
 getFleetRoleCredentialsRequest;
getFleetRoleCredentialsRequest.SetRoleArn("arn:aws:iam::123456789012:role/service-role/
exampleGameLiftAction");
```

```
Aws::GameLift::GetFleetRoleCredentialsOutcome credentials =
 Aws::GameLift::Server::GetFleetRoleCredentials(getFleetRoleCredentialsRequest);
```

This example shows the use of the optional `RoleSessionName` value to assign a name to the credentials session for auditing purposes. If you don't provide a role session name, the default value "*[fleet-id]-[host-id]*" is used.

```
// form the fleet credentials request
Aws::GameLift::Server::Model::GetFleetRoleCredentialsRequest
 getFleetRoleCredentialsRequest;
getFleetRoleCredentialsRequest.SetRoleArn("arn:aws:iam::123456789012:role/service-role/
exampleGameLiftAction");
getFleetRoleCredentialsRequest.SetRoleSessionName("MyFleetRoleSession");

Aws::GameLift::GetFleetRoleCredentialsOutcome credentials =
 Aws::GameLift::Server::GetFleetRoleCredentials(getFleetRoleCredentialsRequest);
```

Destroy()

Frees the Amazon GameLift game server SDK from memory. As a best practice, call this method after `ProcessEnding()` and before terminating the process. If you're using an Anywhere fleet and you're not terminating server processes after every game session, call `Destroy()` and then `InitSDK()` to reinitialize before notifying Amazon GameLift that the process is ready to host a game session with `ProcessReady()`.

Syntax

```
GenericOutcome Aws::GameLift::Server::Destroy();
```

Parameters

There are no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
Aws::GameLift::GenericOutcome processEndingOutcome =
 Aws::GameLift::Server::ProcessEnding();
Aws::GameLift::Server::Destroy();
```

```
// Exit the process with success or failure
if (processEndingOutcome.IsSuccess()) {
  exit(0);
}
else {
  cout << "ProcessEnding() failed. Error: " <<
 processEndingOutcome.GetError().GetErrorMessage();
  exit(-1);
}
```

Amazon GameLift server SDK (C++) reference: Data types

You can use this Amazon GameLift C++ server SDK reference to help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server.

> **Note**
>
> This topic describes the Amazon GameLift C++ API that you can use when you build with the C++ Standard Library (`std`). Specifically, this documentation applies to code that you compile with the `-DGAMELIFT_USE_STD=1` option.

Data types

- LogParameters
- ProcessParameters
- UpdateGameSession
- GameSession
- ServerParameters
- StartMatchBackfillRequest
- Player
- DescribePlayerSessionsRequest
- StopMatchBackfillRequest
- AttributeValue
- GetFleetRoleCredentialsRequest
- AwsLongOutcome

- AwsStringOutcome
- DescribePlayerSessionsOutcome
- DescribePlayerSessionsResult
- GenericOutcome
- GenericOutcomeCallable
- PlayerSession
- StartMatchBackfillOutcome
- StartMatchBackfillResult
- GetComputeCertificateOutcome
- GetComputeCertificateResult
- GetFleetRoleCredentialsOutcome
- GetFleetRoleCredentialsResult
- InitSDKOutcome
- GameLiftError
- Enums

LogParameters

An object identifying files generated during a game session that you want Amazon GameLift to upload and store after the game session ends. The game server provides `LogParameters` to Amazon GameLift as part of a `ProcessParameters` object in a ProcessReady() call.

Properties	Description
LogPaths	The list of directory paths to game server log files you want Amazon GameLift to store for future access. The server process generates these files during each game session. You define file paths and names in your game server and store them in the root game build directory. The log paths must be absolute. For example, if your game build stores game session logs in

	a path like `MyGame\sessionLogs\` , then the path would be `c:\game\MyGame\sessionLogs` on a Windows instance. **Type:** `std:vector<std::string>` **Required:** No

ProcessParameters

This data type contains the set of parameters sent to Amazon GameLift in a ProcessReady().

Properties	Description
LogParameters	An object with directory paths to files that are generated during a game session. Amazon GameLift copies and stores the files for future access. **Type:** `Aws::GameLift::Server:: LogParameters` **Required:** No
OnHealthCheck	The callback function that Amazon GameLift invokes to request a health status report from the server process. Amazon GameLift calls this function every 60 seconds and waits 60 seconds for a response. The server process returns `TRUE` if healthy, `FALSE` if not healthy. If no response is returned, Amazon GameLift records the server process as not healthy. **Type:** `std::function<bool()> onHealthCheck` **Required:** No

OnProcessTerminate	The callback function that Amazon GameLift invokes to force the server process to shut down. After calling this function, Amazon GameLift waits 5 minutes for the server process to shut down and respond with a ProcessEnding() call before it shuts down the server process. **Type:** `std::function<void()> onProcessTerminate` **Required:** Yes
OnRefreshConnection	The name of the callback function that Amazon GameLift invokes to refresh the connection with the game server. **Type:** `void OnRefreshConnectionDelegate()` **Required:** Yes
OnStartGameSession	The callback function that Amazon GameLift invokes to activate a new game session. Amazon GameLift calls this function in response to a client request CreateGameSession. The callback function passes a GameSession object, as defined in the *Amazon GameLift API Reference*. **Type:** `const std::function<void (Aws::GameLift::Model::GameSession)> onStartGameSession` **Required:** Yes

OnUpdateGameSession	The callback function that Amazon GameLift invokes to pass an updated game session object to the server process. Amazon GameLift calls this function when a match backfill request has been processed to provide updated matchmaker data. It passes a GameSession object, a status update (`updateReason`), and the match backfill ticket ID. **Type:** `std::function<void(Aws::GameLift::Server::Model::UpdateGameSession)> onUpdateGameSession` **Required:** No
Port	The port number the server process listens on for new player connections. The value must fall into the port range configured for any fleet deploying this game server build. This port number is included in game session and player session objects, which game sessions use when connecting to a server process. **Type:** `Integer` **Required:** Yes

UpdateGameSession

This data type updates to a game session object, which includes the reason that the game session was updated and the related backfill ticket ID if backfill is used to fill player sessions in the game session.

Properties	Description
GameSession	A GameSession object defined by the Amazon GameLift API. The `GameSession` object contains properties describing a game session. **Type:** `Aws::GameLift::Server::GameSession` **Required:** Yes
UpdateReason	The reason that the game session is being updated. **Type:** `Aws::GameLift::Server::UpdateReason` **Required:** Yes
BackfillTicketId	The ID of the backfill ticket attempting to update the game session. **Type:** `std::string` **Required:** No

GameSession

This data type provides details of a game session.

Properties	Description
GameSessionId	A unique identifier for the game session. A game session ARN has the following format: `arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token> .` **Type:** `std::string`

Properties	Description
	Required: No
Name	A descriptive label of the game session. **Type:** `std::string` **Required**: No
FleetId	A unique identifier for the fleet that the game session is running on. **Type:** `std::string` **Required**: No
MaximumPlayerSessionCount	The maximum number of player connections to the game session. **Type:** `int` **Required**: No
Port	The port number for the game session. To connect to a Amazon GameLift game server, an app needs both the IP address and port number. **Type:** `in` **Required**: No
IpAddress	The IP address of the game session. To connect to a Amazon GameLift game server, an app needs both the IP address and port number. **Type:** `std::string` **Required**: No

Properties	Description
GameSessionData	A set of custom game session properties, formatted as a single string value. **Type:** `std::string` **Required**: No
MatchmakerData	Information about the matchmaking process that was used to create the game session, in JSON syntax, formatted as a string. In addition to the matchmaking configuration used, it contains data on all players assigned to the match, including player attributes and team assignments. **Type:** `std::string` **Required**: No
GameProperties	A set of custom properties for a game session, formatted as key:value pairs. These properties are passed with a request to start a new game session. **Type:** `std :: vector < GameProperty >` **Required**: No

Properties	Description
DnsName	The DNS identifier assigned to the instance that's running the game session. Values have the following format: • TLS-enabled fleets: `<unique identifier>.<region identifier>.amazongamelift.com .` • Non-TLS-enabled fleets: `ec2-<unique identifier>.compute.amazonaws.com .` When connecting to a game session that's running on a TLS-enabled fleet, you must use the DNS name, not the IP address. **Type:** `std::string` **Required**: No

ServerParameters

Information used to maintain the connection between game server on an Amazon GameLift Anywhere fleet and the Amazon GameLift service. This information is used when launching new server processes with InitSDK(). For servers hosted on Amazon GameLift managed EC2 instances, use an empty object.

Properties	Description
webSocketUrl	The `GameLiftServerSdkEndpoint` Amazon GameLift returns when you `RegisterCompute` for a Amazon GameLift Anywhere compute resource. **Type:** `std::string`

Properties	Description
	Required: Yes
processId	A unique identifier registered to the server process hosting your game. **Type:** `std::string` **Required**: Yes
hostId	The `HostID` is the `ComputeName` used when you registered your compute. For more information see, RegisterCompute. **Type:** `std::string` **Required**: Yes
fleetId	The unique identifier of the fleet that the compute is registered to. For more informati on see, RegisterCompute. **Type:** `std::string` **Required**: Yes
authToken	The authentication token generated by Amazon GameLift that authenticates your server to Amazon GameLift. For more information see, GetComputeAuthToken. **Type:** `std::string` **Required**: Yes

StartMatchBackfillRequest

Information used to create a matchmaking backfill request. The game server communicates this information to Amazon GameLift in a StartMatchBackfill() call.

Properties	Description
GameSessionArn	A unique game session identifier. The API operation `GetGameSessionId` returns the identifier in ARN format. **Type:** `std::string` **Required**: Yes
MatchmakingConfigurationArn	A unique identifier, in the form of an ARN, for the matchmaker to use for this request. The matchmaker ARN for the original game session is in the game session object in the matchmaker data property. Learn more about matchmaker data in Work with matchmaker data. **Type:** `std::string` **Required**: Yes
Players	A set of data representing all players who are in the game session. The matchmaker uses this information to search for new players who are good matches for the current players. **Type:** `std::vector<Player>` **Required**: Yes
TicketId	A unique identifier for a matchmaking or match backfill request ticket. If you don't provide a value, Amazon GameLift generates one. Use this identifier to track the match backfill ticket status or cancel the request if needed. **Type:** `std::string`

Properties	Description
	Required: No

Player

This data type represents a player in matchmaking. When starting a matchmaking request, a player has a player ID, attributes, and possibly latency data. Amazon GameLift adds team information after a match is made.

Properties	Description
LatencyInMS	A set of values expressed in milliseconds that indicate the amount of latency that a player experiences when connected to a location. If this property is used, the player is only matched for locations listed. If a matchmaker has a rule that evaluates player latency, players must report latency to be matched. **Type:** `Dictionary<string,int>` **Required**: No
PlayerAttributes	A collection of key:value pairs containing player information for use in matchmaking. Player attribute keys must match the PlayerAttributes used in a matchmaking rule set. For more information about player attributes, see AttributeValue. **Type:** `std::map<std::string,AttributeValue>` **Required**: No
PlayerId	A unique identifier for a player.

Properties	Description
	Type: `std::string` **Required**: No
Team	The name of the team that the player is assigned to in a match. You define team name in the matchmaking rule set. **Type:** `std::string` **Required**: No

DescribePlayerSessionsRequest

An object that specifies which player sessions to retrieve. The server process provides this information with a DescribePlayerSessions() call to Amazon GameLift.

Properties	Description
GameSessionId	A unique game session identifier. Use this parameter to request all player sessions for the specified game session. Game session ID format is `arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>` . The `GameSessionID` is a custom ID string or a **Type:** `std::string` **Required**: No
PlayerSessionId	The unique identifier for a player session. Use this parameter to request a single specific player session. **Type:** `std::string`

Properties	Description
	Required: No
PlayerId	The unique identifier for a player. Use this parameter to request all player sessions for a specific player. See Generate player IDs. **Type:** `std::string` **Required**: No
PlayerSessionStatusFilter	The player session status to filter results on. Possible player session statuses include: • RESERVED – The player session request was received, but the player hasn't connected to the server process or been validated. • ACTIVE – The player was validated by the server process and is connected. • COMPLETED – The player connection dropped. • TIMEDOUT – A player session request was received, but the player didn't connect or wasn't validated within the time-out limit (60 seconds). **Type:** `std::string` **Required**: No

Properties	Description
NextToken	The token indicating the start of the next page of results. To specify the start of the result set, don't provide a value. If you provide a player session ID, this parameter is ignored. **Type:** `std::string` **Required**: No
Limit	The maximum number of results to return. If you provide a player session ID, this parameter is ignored. **Type:** `int` **Required**: No

StopMatchBackfillRequest

Information used to cancel a matchmaking backfill request. The game server communicates this information to Amazon GameLift service in a StopMatchBackfill() call.

Properties	Description
GameSessionArn	A unique game session identifier of the request being canceled. **Type:** `char[]` **Required**: No
MatchmakingConfigurationArn	A unique identifier of the matchmaker this request was sent to. **Type:** `char[]` **Required**: No

Properties	Description
TicketId	A unique identifier of the backfill request ticket to be canceled. **Type:** `char[]` **Required**: No

AttributeValue

Use these values in Player attribute key-value pairs. This object lets you specify an attribute value using any of the valid data types: string, number, string array, or data map. Each `AttributeValue` object must use exactly one of the available properties: `S`, `N`, `SL`, or `SDM`.

Properties	Description
AttrType	Specifies the type of attribute value. Possible attribute value types include: • **NONE** • **STRING** • **DOUBLE** • **STRING_LIST** • **STRING_DOUBLE_MAP** **Required:** No
S	Represents a string attribute value. **Type:** `std::string` **Required:** No
N	Represents a numeric attribute value. **Type:** `double`

Properties	Description
	Required: No
SL	Represents an array of string attribute values. **Type:** `std::vector<std::string>` **Required:** No
SDM	Represents a dictionary of string keys and double values. **Type:** `std::map<std::string, double>` **Required:** No

GetFleetRoleCredentialsRequest

This data type gives the game server limited access to your other AWS resources. For more information see, Set up an IAM service role for Amazon GameLift.

Properties	Description
RoleArn	The Amazon Resource Name (ARN) of the service role that extends limited access to your AWS resources. **Type:** `std::string` **Required**: No
RoleSessionName	The role session name that you can use to uniquely identify an AWS Security Token Service AssumeRole session. This name is exposed in audit logs such as those in CloudTrail. **Type:** `std::string`

Properties	Description
	Required: No

AwsLongOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. **Type:** `long` **Required:** No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. **Type:** `long&&` **Required**: No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called “GameLiftError” **Required:** No

AwsStringOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. **Type:** `std::string` **Required:** No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. **Type:** `long&&` **Required**: No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called "GameLiftError" **Required:** No

DescribePlayerSessionsOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. **Type:** the section called "DescribePlayerSessionsResult"

Properties	Description
	Required: No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. **Type:** `Aws::GameLift::Server::Model::DescribePlayerSessionsResult&&` **Required**: No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called "GameLiftError" **Required:** No

DescribePlayerSessionsResult

A collection of objects containing properties for each player session that matches the request.

Properties	Description
NextToken	A token that indicates the start of the next sequential page of results. Use the token that is returned with a previous call to this operation. To start at the beginning of the result set, do not specify a value. If a player session ID is specified, this parameter is ignored.

Properties	Description
	Type: `std::string` **Required:** Yes
PlayerSessions	**Type:** `IList<`the section called "PlayerSession" `>` **Required:**
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. **Type:** `std::string&&` **Required**: No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called "GameLiftError" **Required:** No

GenericOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Success	Whether the action was successful or not. **Type:** `bool`

Properties	Description
	Required: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called “GameLiftError” **Required:** No

GenericOutcomeCallable

This data type is an asynchronous generic outcome. It has the following properties:

Properties	Description
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called “GameLiftError” **Required:** No

PlayerSession

This data type represents a player session that Amazon GameLift passes to the game server. For more information, see PlayerSession.

Properties	Description
CreationTime	**Type:** `long`

Properties	Description
	Required: No
FleetId	**Type:** `std::string` **Required:** No
GameSessionId	**Type:** `std::string` **Required:** No
IpAddress	**Type:** `std::string` **Required:** No
PlayerData	**Type:** `std::string` **Required:** No
PlayerId	**Type:** `std::string` **Required:** No
PlayerSessionId	**Type:** `std::string` **Required:** No
Port	**Type:** `int` **Required:** No

Properties	Description
Status	Player session status to filter results on. When a PlayerSessionId or PlayerId is provided, then the PlayerSessionStatusFilter has no effect on the response. **Type:** A `PlayerSessionStatus` enum. Possible values include the following: • **ACTIVE** • **COMPLETED** • **NOT_SET** • **RESERVED** • **TIMEDOUT** **Required:** No
TerminationTime	**Type:** `long` **Required:** No
DnsName	**Type:** `std::string` **Required:** No

StartMatchBackfillOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. **Type:** the section called “StartMatchBackfillResult”

Properties	Description
	Required: No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. **Type:** `StartMatchBackfillResult&&` **Required**: No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called "GameLiftError" **Required:** No

StartMatchBackfillResult

This data type results from an action and produces an object with the following properties:

Properties	Description
TicketId	A unique identifier for a matchmaking ticket. If no ticket ID is specified here, Amazon GameLift will generate one in the form of a UUID. Use this identifier to track the match backfill ticket status and retrieve match results. **Type:** `std::string`

Properties	Description
	Required: No

GetComputeCertificateOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. **Type:** the section called "GetComputeCertificateResult" **Required:** No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. **Type:** `Aws::GameLift::Server::Model::GetComputeCertificateResult&&` **Required**: No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called "GameLiftError" **Required:** No

GetComputeCertificateResult

The path to the TLS certificate on your compute and the compute's host name.

Properties	Description
CertificatePath	The path to the TLS certificate on your compute resource. When using an Amazon GameLift managed fleet, this path contains: • `certificate.pem` : The end-user certificate. The full certificate chain is the combination of `certificateChain.pem` appended to this certificate. • `certificateChain.pem` : The certificate chain that contains the root certificate and intermediate certificates. • `rootCertificate.pem` : The root certificate. • `privateKey.pem` : The private key for the end-user certificate. **Type:** `std::string` **Required:** No
ComputeName	The name of your compute resource. **Type:** `std::string` **Required:** No

GetFleetRoleCredentialsOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. **Type:** the section called "GetFleetRoleCredentialsResult" **Required:** No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. **Type:** `Aws::GameLift::Server::Model::GetFleetRoleCredentialsResult` **Required**: No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called "GameLiftError" **Required:** No

GetFleetRoleCredentialsResult

Properties	Description
AccessKeyId	The access key ID to authenticate and provide access to your AWS resources.

Properties	Description
	Type: `string` **Required:** No
AssumedRoleId	The ID of the user that the service role belongs to. **Type:** `string` **Required:** No
AssumedRoleUserArn	The Amazon Resource Name (ARN) of the user that the service role belongs to. **Type:** `string` **Required:** No
Expiration	The amount of time until your session credentials expire. **Type:** `DateTime` **Required:** No
SecretAccessKey	The secret access key ID for authentication. **Type:** `string` **Required:** No
SessionToken	A token to identify the current active session interacting with your AWS resources. **Type:** `string` **Required:** No

Properties	Description
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called "GameLiftError" **Required:** No

InitSDKOutcome

> **Note**
>
> `InitSDKOutcome` is returned only when you build the SDK with the `std` flag. If you build with the `nostd` flag, then the section called "GenericOutcome" is returned instead.

Properties	Description
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called "GameLiftError" **Required:** No

GameLiftError

Properties	Description
ErrorType	The type of error. **Type:** A `GameLiftErrorType` enum. **Required:** No
ErrorName	The name of the error. **Type:** `std::string` **Required:** No
ErrorMessage	The error message. **Type:** `std::string` **Required:** No

Enums

Enums defined for the Amazon GameLift server SDK (C++) are defined as follows:

GameLiftErrorType

String value indicating the error type. Valid values include:

- **BAD_REQUEST_EXCEPTION**
- **GAMESESSION_ID_NOT_SET** – The game session ID has not been set.
- **INTERNAL_SERVICE_EXCEPTION**
- **LOCAL_CONNECTION_FAILED** – The local connection to Amazon GameLift failed.
- **NETWORK_NOT_INITIALIZED** – The network has not been initialized.
- **SERVICE_CALL_FAILED** – A call to an AWS service has failed.
- **WEBSOCKET_CONNECT_FAILURE**
- **WEBSOCKET_CONNECT_FAILURE_FORBIDDEN**
- **WEBSOCKET_CONNECT_FAILURE_INVALID_URL**

- **WEBSOCKET_CONNECT_FAILURE_TIMEOUT**
- **ALREADY_INITIALIZED** – The Amazon GameLift Server or Client has already been initialized with Initialize().
- **FLEET_MISMATCH** – The target fleet does not match the fleet of a gameSession or playerSession.
- **GAMELIFT_CLIENT_NOT_INITIALIZED** – The Amazon GameLift client has not been initialized.
- **GAMELIFT_SERVER_NOT_INITIALIZED** – The Amazon GameLift server has not been initialized.
- **GAME_SESSION_ENDED_FAILED** – The Amazon GameLift Server SDK could not contact the service to report the game session ended.
- **GAME_SESSION_NOT_READY** – The Amazon GameLift Server Game Session was not activated.
- **GAME_SESSION_READY_FAILED** – The Amazon GameLift Server SDK could not contact the service to report the game session is ready.
- **INITIALIZATION_MISMATCH** – A client method was called after Server::Initialize(), or vice versa.
- **NOT_INITIALIZED** – The Amazon GameLift Server or Client has not been initialized with Initialize().
- **NO_TARGET_ALIASID_SET** – A target aliasId has not been set.
- **NO_TARGET_FLEET_SET** – A target fleet has not been set.
- **PROCESS_ENDING_FAILED** – The Amazon GameLift Server SDK could not contact the service to report the process is ending.
- **PROCESS_NOT_ACTIVE** – The server process is not yet active, not bound to a GameSession, and cannot accept or process PlayerSessions.
- **PROCESS_NOT_READY** – The server process is not yet ready to be activated.
- **PROCESS_READY_FAILED** – The Amazon GameLift Server SDK could not contact the service to report the process is ready.
- **SDK_VERSION_DETECTION_FAILED** – SDK version detection failed.
- **STX_CALL_FAILED** – A call to the XStx server backend component has failed.
- **STX_INITIALIZATION_FAILED** – The XStx server backend component has failed to initialize.
- **UNEXPECTED_PLAYER_SESSION** – An unregistered player session was encountered by the server.

- **WEBSOCKET_CONNECT_FAILURE**
- **WEBSOCKET_CONNECT_FAILURE_FORBIDDEN**
- **WEBSOCKET_CONNECT_FAILURE_INVALID_URL**
- **WEBSOCKET_CONNECT_FAILURE_TIMEOUT**
- **WEBSOCKET_RETRIABLE_SEND_MESSAGE_FAILURE** – Retriable failure to send a message to the GameLift Service WebSocket.
- **WEBSOCKET_SEND_MESSAGE_FAILURE** – Failure to send a message to the GameLift Service WebSocket.
- **MATCH_BACKFILL_REQUEST_VALIDATION** – Validation of the request failed.
- **PLAYER_SESSION_REQUEST_VALIDATION** – Validation of the request failed.

PlayerSessionCreationPolicy

String value indicating whether the game session accepts new players. Valid values include:

- **ACCEPT_ALL** – Accept all new player sessions.
- **DENY_ALL** – Deny all new player sessions.
- **NOT_SET** – The game session is not set to accept or deny new player sessions.

Amazon GameLift C++ server SDK 3.x reference

You can use this Amazon GameLift C++ server SDK 3.x reference to help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server.

Topics

- Amazon GameLift server SDK (C++) reference: Actions
- Amazon GameLift server SDK (C++) reference: Data types

Amazon GameLift server SDK (C++) reference: Actions

You can use this Amazon GameLift C++ server SDK reference to help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server.

Actions

- AcceptPlayerSession()

- ActivateGameSession()
- DescribePlayerSessions()
- GetGameSessionId()
- GetInstanceCertificate()
- GetSdkVersion()
- GetTerminationTime()
- InitSDK()
- ProcessEnding()
- ProcessReady()
- ProcessReadyAsync()
- RemovePlayerSession()
- StartMatchBackfill()
- StopMatchBackfill()
- TerminateGameSession()
- UpdatePlayerSessionCreationPolicy()
- Destroy()

AcceptPlayerSession()

Notifies the Amazon GameLift service that a player with the specified player session ID has connected to the server process and needs validation. Amazon GameLift verifies that the player session ID is valid—that is, that the player ID has reserved a player slot in the game session. Once validated, Amazon GameLift changes the status of the player slot from RESERVED to ACTIVE.

Syntax

```
GenericOutcome AcceptPlayerSession(const std::string& playerSessionId);
```

Parameters

playerSessionId

Unique ID issued by the Amazon GameLift service in response to a call to the AWS SDK Amazon GameLift API action CreatePlayerSession. The game client references this ID when connecting to the server process.

Type: std::string

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example illustrates a function for handling a connection request, including validating and rejecting invalid player session IDs.

```
void ReceiveConnectingPlayerSessionID (Connection& connection, const std::string&
 playerSessionId){
    Aws::GameLift::GenericOutcome connectOutcome =
        Aws::GameLift::Server::AcceptPlayerSession(playerSessionId);
    if(connectOutcome.IsSuccess())
    {
        connectionToSessionMap.emplace(connection, playerSessionId);
        connection.Accept();
    }
    else
    {
        connection.Reject(connectOutcome.GetError().GetMessage();
    }
}
```

ActivateGameSession()

Notifies the Amazon GameLift service that the server process has started a game session and is now ready to receive player connections. This action should be called as part of the on`StartGameSession()` callback function, after all game session initialization has been completed.

Syntax

```
GenericOutcome ActivateGameSession();
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example shows `ActivateGameSession()` being called as part of the `onStartGameSession()` callback function.

```
void onStartGameSession(Aws::GameLift::Model::GameSession myGameSession)
{
   // game-specific tasks when starting a new game session, such as loading map
   GenericOutcome outcome = Aws::GameLift::Server::ActivateGameSession();
}
```

DescribePlayerSessions()

Retrieves player session data, including settings, session metadata, and player data. Use this action to get information for a single player session, for all player sessions in a game session, or for all player sessions associated with a single player ID.

Syntax

```
DescribePlayerSessionsOutcome DescribePlayerSessions (
    const Aws::GameLift::Server::Model::DescribePlayerSessionsRequest
 &describePlayerSessionsRequest);
```

Parameters

describePlayerSessionsRequest

A DescribePlayerSessionsRequest object describing which player sessions to retrieve.

Required: Yes

Return value

If successful, returns a `DescribePlayerSessionsOutcome` object containing a set of player session objects that fit the request parameters. Player session objects have a structure identical to the AWS SDK Amazon GameLift API PlayerSession data type.

Example

This example illustrates a request for all player sessions actively connected to a specified game session. By omitting `NextToken` and setting the `Limit` value to 10, Amazon GameLift returns the first 10 player sessions records matching the request.

```
// Set request parameters
Aws::GameLift::Server::Model::DescribePlayerSessionsRequest request;
request.SetPlayerSessionStatusFilter(Aws::GameLift::Server::Model::PlayerSessionStatusMapper::G
request.SetLimit(10);
request.SetGameSessionId("the game session ID");    // can use GetGameSessionId()

// Call DescribePlayerSessions
Aws::GameLift::DescribePlayerSessionsOutcome playerSessionsOutcome =
    Aws::GameLift::Server::DescribePlayerSessions(request);
```

GetGameSessionId()

Retrieves a unique identifier for the game session currently being hosted by the server process, if the server process is active. The identifier is returned in ARN format: `arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>`.

For idle process that are not yet activated with a game session, the call returns `Success=True` and `GameSessionId=""` (an empty string).

Syntax

```
AwsStringOutcome GetGameSessionId();
```

Parameters

This action has no parameters.

Return value

If successful, returns the game session ID as an `AwsStringOutcome` object. If not successful, returns an error message.

Example

```
Aws::GameLift::AwsStringOutcome sessionIdOutcome =
```

```
    Aws::GameLift::Server::GetGameSessionId();
```

GetInstanceCertificate()

Retrieves the file location of a pem-encoded TLS certificate that is associated with the fleet and its instances. AWS Certificate Manager generates this certificate when you create a new fleet with the certificate configuration set to GENERATED. Use this certificate to establish a secure connection with a game client and to encrypt client/server communication.

Syntax

```
GetInstanceCertificateOutcome GetInstanceCertificate();
```

Parameters

This action has no parameters.

Return value

If successful, returns a `GetInstanceCertificateOutcome` object containing the location of the fleet's TLS certificate file and certificate chain, which are stored on the instance. A root certificate file, extracted from the certificate chain, is also stored on the instance. If not successful, returns an error message.

For more information about the certificate and certificate chain data, see GetCertificate Response Elements in the AWS Certificate Manager API Reference.

Example

```
Aws::GameLift::GetInstanceCertificateOutcome certificateOutcome =
    Aws::GameLift::Server::GetInstanceCertificate();
```

GetSdkVersion()

Returns the current version number of the SDK in use.

Syntax

```
AwsStringOutcome GetSdkVersion();
```

Parameters

This action has no parameters.

Return value

If successful, returns the current SDK version as an `AwsStringOutcome` object. The returned string includes the version number only (ex. "3.1.5"). If not successful, returns an error message.

Example

```
Aws::GameLift::AwsStringOutcome SdkVersionOutcome =
    Aws::GameLift::Server::GetSdkVersion();
```

GetTerminationTime()

Returns the time that a server process is scheduled to be shut down, if a termination time is available. A server process takes this action after receiving an on`ProcessTerminate()` callback from the Amazon GameLift service. Amazon GameLift may call on`ProcessTerminate()` for the following reasons: (1) when the server process has reported poor health or has not responded to Amazon GameLift, (2) when terminating the instance during a scale-down event, or (3) when an instance is being terminated due to a Spot interruption.

If the process has received an on`ProcessTerminate()` callback, the value returned is the estimated termination time. If the process has not received an on`ProcessTerminate()` callback, an error message is returned. Learn more about shutting down a server process.

Syntax

```
AwsLongOutcome GetTerminationTime();
```

Parameters

This action has no parameters.

Return value

If successful, returns the termination time as an `AwsLongOutcome` object. The value is the termination time, expressed in elapsed ticks since 0001 00:00:00. For example, the date time value

2020-09-13 12:26:40 -000Z is equal to 637355968000000000 ticks. If no termination time is available, returns an error message.

Example

```
Aws::GameLift::AwsLongOutcome TermTimeOutcome =
    Aws::GameLift::Server::GetTerminationTime();
```

InitSDK()

Initializes the Amazon GameLift SDK. This method should be called on launch, before any other Amazon GameLift-related initialization occurs.

Syntax

```
InitSDKOutcome InitSDK();
```

Parameters

This action has no parameters.

Return value

If successful, returns an InitSdkOutcome object indicating that the server process is ready to call ProcessReady().

Example

```
Aws::GameLift::Server::InitSDKOutcome initOutcome =
    Aws::GameLift::Server::InitSDK();
```

ProcessEnding()

Notifies the Amazon GameLift 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

```
GenericOutcome ProcessEnding();
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
Aws::GameLift::GenericOutcome outcome = Aws::GameLift::Server::ProcessEnding();
if (outcome.Success)
    exit(0);  // exit with success
// otherwise, exit with error code
exit(errorCode);
```

ProcessReady()

Notifies the Amazon GameLift service that the server process is ready to host game sessions. Call this method after successfully invoking InitSDK() and completing setup tasks that are required before the server process can host a game session. This method should be called only once per process.

This call is synchronous. To make an asynchronous call, use ProcessReadyAsync(). See Initialize the server process for more details.

Syntax

```
GenericOutcome ProcessReady(
    const Aws::GameLift::Server::ProcessParameters &processParameters);
```

Parameters

processParameters

A ProcessParameters object communicating the following information about the server process:

- Names of callback methods, implemented in the game server code, that the Amazon GameLift service invokes to communicate with the server process.
- Port number that the server process is listening on.
- Path to any game session-specific files that you want Amazon GameLift to capture and store.

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example illustrates both the ProcessReady() call and callback function implementations.

```
// Set parameters and call ProcessReady
std::string serverLog("serverOut.log");        // Example of a log file written by the
 game server
std::vector<std::string> logPaths;
logPaths.push_back(serverLog);

int listenPort = 9339;

Aws::GameLift::Server::ProcessParameters processReadyParameter =
 Aws::GameLift::Server::ProcessParameters(
    std::bind(&Server::onStartGameSession, this, std::placeholders::_1),
    std::bind(&Server::onProcessTerminate, this),
    std::bind(&Server::OnHealthCheck, this),
    std::bind(&Server::OnUpdateGameSession, this),
    listenPort,
    Aws::GameLift::Server::LogParameters(logPaths));

Aws::GameLift::GenericOutcome outcome =
   Aws::GameLift::Server::ProcessReady(processReadyParameter);

// Implement callback functions
void Server::onStartGameSession(Aws::GameLift::Model::GameSession myGameSession)
{
   // game-specific tasks when starting a new game session, such as loading map
   GenericOutcome outcome =
       Aws::GameLift::Server::ActivateGameSession (maxPlayers);
```

```
}

void Server::onProcessTerminate()
{
   // game-specific tasks required to gracefully shut down a game session,
   // such as notifying players, preserving game state data, and other cleanup
   GenericOutcome outcome = Aws::GameLift::Server::ProcessEnding();
}

bool Server::onHealthCheck()
{
    bool health;
    // complete health evaluation within 60 seconds and set health
    return health;
}
```

ProcessReadyAsync()

Notifies the Amazon GameLift service that the server process is ready to host game sessions. This method should be called once the server process is ready to host a game session. The parameters specify the names of callback functions for Amazon GameLift to call in certain circumstances. Game server code must implement these functions.

This call is asynchronous. To make a synchronous call, use ProcessReady(). See Initialize the server process for more details.

Syntax

```
GenericOutcomeCallable ProcessReadyAsync(
    const Aws::GameLift::Server::ProcessParameters &processParameters);
```

Parameters

processParameters

A ProcessParameters object communicating the following information about the server process:

- Names of callback methods, implemented in the game server code, that the Amazon GameLift service invokes to communicate with the server process.
- Port number that the server process is listening on.
- Path to any game session-specific files that you want Amazon GameLift to capture and store.

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
// Set parameters and call ProcessReady
std::string serverLog("serverOut.log");        // This is an example of a log file
 written by the game server
std::vector<std::string> logPaths;
logPaths.push_back(serverLog);

int listenPort = 9339;

Aws::GameLift::Server::ProcessParameters processReadyParameter =
 Aws::GameLift::Server::ProcessParameters(
    std::bind(&Server::onStartGameSession, this, std::placeholders::_1),
    std::bind(&Server::onProcessTerminate, this),
    std::bind(&Server::OnHealthCheck, this),
    std::bind(&Server::OnUpdateGameSession, this),
    listenPort,
    Aws::GameLift::Server::LogParameters(logPaths));

Aws::GameLift::GenericOutcomeCallable outcome =
   Aws::GameLift::Server::ProcessReadyAsync(processReadyParameter);

// Implement callback functions
void onStartGameSession(Aws::GameLift::Model::GameSession myGameSession)
{
   // game-specific tasks when starting a new game session, such as loading map
   GenericOutcome outcome = Aws::GameLift::Server::ActivateGameSession (maxPlayers);
}

void onProcessTerminate()
{
   // game-specific tasks required to gracefully shut down a game session,
   // such as notifying players, preserving game state data, and other cleanup
   GenericOutcome outcome = Aws::GameLift::Server::ProcessEnding();
}

bool onHealthCheck()
```

```
{
    // perform health evaluation and complete within 60 seconds
    return health;
}
```

RemovePlayerSession()

Notifies the Amazon GameLift service that a player with the specified player session ID has disconnected from the server process. In response, Amazon GameLift changes the player slot to available, which allows it to be assigned to a new player.

Syntax

```
GenericOutcome RemovePlayerSession(
    const std::string& playerSessionId);
```

Parameters

playerSessionId

Unique ID issued by the Amazon GameLift service in response to a call to the AWS SDK Amazon GameLift API action CreatePlayerSession. The game client references this ID when connecting to the server process.

Type: std::string

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
Aws::GameLift::GenericOutcome disconnectOutcome =
    Aws::GameLift::Server::RemovePlayerSession(playerSessionId);
```

StartMatchBackfill()

Sends a request to find new players for open slots in a game session created with FlexMatch. See also the AWS SDK action StartMatchBackfill(). With this action, match backfill requests can

be initiated by a game server process that is hosting the game session. Learn more about the FlexMatch backfill feature.

This action is asynchronous. If new players are successfully matched, the Amazon GameLift service delivers updated matchmaker data by invoking the callback function `OnUpdateGameSession()`.

A server process can have only one active match backfill request at a time. To send a new request, first call StopMatchBackfill() to cancel the original request.

Syntax

```
StartMatchBackfillOutcome StartMatchBackfill (
    const Aws::GameLift::Server::Model::StartMatchBackfillRequest
 &startBackfillRequest);
```

Parameters

StartMatchBackfillRequest

A StartMatchBackfillRequest object that communicates the following information:

- A ticket ID to assign to the backfill request. This information is optional; if no ID is provided, Amazon GameLift will autogenerate one.
- The matchmaker to send the request to. The full configuration ARN is required. This value can be acquired from the game session's matchmaker data.
- The ID of the game session that is being backfilled.
- Available matchmaking data for the game session's current players.

Required: Yes

Return value

Returns a StartMatchBackfillOutcome object with the match backfill ticket or failure with an error message. Ticket status can be tracked using the AWS SDK action DescribeMatchmaking().

Example

```
// Build a backfill request
std::vector<Player> players;
Aws::GameLift::Server::Model::StartMatchBackfillRequest startBackfillRequest;
```

```
startBackfillRequest.SetTicketId("a ticket ID");
  //optional, autogenerated if not provided
startBackfillRequest.SetMatchmakingConfigurationArn("the matchmaker configuration
 ARN"); //from the game session matchmaker data
startBackfillRequest.SetGameSessionArn("the game session ARN");
  // can use GetGameSessionId()
startBackfillRequest.SetPlayers(players);
    //from the game session matchmaker data

// Send backfill request
Aws::GameLift::StartMatchBackfillOutcome backfillOutcome =
    Aws::GameLift::Server::StartMatchBackfill(startBackfillRequest);

// Implement callback function for backfill
void Server::OnUpdateGameSession(Aws::GameLift::Server::Model::GameSession gameSession,
 Aws::GameLift::Server::Model::UpdateReason updateReason, std::string backfillTicketId)
{
   // handle status messages
   // perform game-specific tasks to prep for newly matched players
}
```

StopMatchBackfill()

Cancels an active match backfill request that was created with StartMatchBackfill(). See also the AWS SDK action StopMatchmaking(). Learn more about the FlexMatch backfill feature.

Syntax

```
GenericOutcome StopMatchBackfill (
    const Aws::GameLift::Server::Model::StopMatchBackfillRequest &stopBackfillRequest);
```

Parameters

StopMatchBackfillRequest

A StopMatchBackfillRequest object identifying the matchmaking ticket to cancel:

- ticket ID assigned to the backfill request being canceled
- matchmaker the backfill request was sent to
- game session associated with the backfill request

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
// Set backfill stop request parameters

Aws::GameLift::Server::Model::StopMatchBackfillRequest stopBackfillRequest;
stopBackfillRequest.SetTicketId("the ticket ID");
stopBackfillRequest.SetGameSessionArn("the game session ARN");
  // can use GetGameSessionId()
stopBackfillRequest.SetMatchmakingConfigurationArn("the matchmaker configuration ARN");
  // from the game session matchmaker data

Aws::GameLift::GenericOutcome stopBackfillOutcome =
    Aws::GameLift::Server::StopMatchBackfillRequest(stopBackfillRequest);
```

TerminateGameSession()

This method is deprecated with version 4.0.1. Instead, the server process should call ProcessEnding() after a game session has ended.

Notifies the Amazon GameLift service that the server process has ended the current game session. This action is called when the server process will remain active and ready to host a new game session. It should be called only after your game session termination procedure is complete, because it signals to Amazon GameLift that the server process is immediately available to host a new game session.

This action is not called if the server process will be shut down after the game session stops. Instead, call ProcessEnding() to signal that both the game session and the server process are ending.

Syntax

```
GenericOutcome TerminateGameSession();
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

UpdatePlayerSessionCreationPolicy()

Updates the current game session's ability to accept new player sessions. A game session can be set to either accept or deny all new player sessions. See also the AWS SDK action UpdateGameSession().

Syntax

```
GenericOutcome UpdatePlayerSessionCreationPolicy(
    Aws::GameLift::Model::PlayerSessionCreationPolicy newPlayerSessionPolicy);
```

Parameters

newPlayerSessionPolicy

String value indicating whether the game session accepts new players.

Type: Aws::GameLift::Model::PlayerSessionCreationPolicy enum. Valid values include:

- **ACCEPT_ALL** – Accept all new player sessions.
- **DENY_ALL** – Deny all new player sessions.

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example sets the current game session's join policy to accept all players.

```
Aws::GameLift::GenericOutcome outcome =
 Aws::GameLift::Server::UpdatePlayerSessionCreationPolicy(Aws::GameLift::Model::PlayerSessionCr
```

Destroy()

Cleans up memory allocated by initSDK() during game server initialization. Use this method after you end a game server process to avoid wasting server memory.

Syntax

```
GenericOutcome Aws::GameLift::Server::Destroy();
```

Parameters

There are no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example cleans up the memory allocated by initSDK after a game server process has ended.

```
if (Aws::GameLift::Server::ProcessEnding().IsSuccess()) {
  Aws::GameLift::Server::Destroy();
  exit(0);
}
```

Amazon GameLift server SDK (C++) reference: Data types

You can use this Amazon GameLift C++ server SDK reference to help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server.

This API is defined in `GameLiftServerAPI.h`, `LogParameters.h`, and `ProcessParameters.h`.

- Actions
- 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 a previous call to this action. To specify the start of the result set, do not specify a value. If a player session ID is specified, this parameter is ignored.

Type: String

Required: No

PlayerId

Unique identifier for a player. Player IDs are defined by the developer. See Generate player IDs.

Type: String

Required: No

PlayerSessionId

Unique identifier for a player session.

Type: String

Required: No

PlayerSessionStatusFilter

Player session status to filter results on. Possible player session statuses include the following:

- RESERVED – The player session request has been received, but the player has not yet connected to the server process and/or been validated.
- ACTIVE – The player has been validated by the server process and is currently connected.
- COMPLETED – The player connection has been dropped.
- TIMEDOUT – A player session request was received, but the player did not connect and/or was not validated within the time-out limit (60 seconds).

Type: String

Required: No

LogParameters

This data type is used to identify which files generated during a game session that you want Amazon GameLift to upload and store once the game session ends. This information is communicated to the Amazon GameLift service in a ProcessReady() call.

Contents

logPaths

Directory paths to game server log files that you want Amazon GameLift to store for future access. These files are generated during each game session. File paths and names are defined in your game server and stored in the root game build directory. The log paths must be absolute. For example, if your game build stores game session logs in a path like `MyGame\sessionlogs\`, then the log path would be `c:\game\MyGame\sessionLogs` (on a Windows instance) or `/local/game/MyGame/sessionLogs` (on a Linux instance).

Type: std:vector<std::string>

Required: No

ProcessParameters

This data type contains the set of parameters sent to the Amazon GameLift service in a ProcessReady() call.

Contents

port

Port number the server process listens on for new player connections. The value must fall into the port range configured for any fleet deploying this game server build. This port number is included in game session and player session objects, which game sessions use when connecting to a server process.

Type: Integer

Required: Yes

logParameters

Object with a list of directory paths to game session log files.

Type: Aws::GameLift::Server::LogParameters

Required: No

onStartGameSession

Name of callback function that the Amazon GameLift service invokes to activate a new game session. Amazon GameLift calls this function in response to the client request CreateGameSession. The callback function passes a GameSession object (defined in the *Amazon GameLift Service API Reference*).

Type: `const std::function<void(Aws::GameLift::Model::GameSession)> onStartGameSession`

Required: Yes

onProcessTerminate

Name of callback function that the Amazon GameLift service invokes to force the server process to shut down. After calling this function, Amazon GameLift waits five minutes for the server process to shut down and respond with a ProcessEnding() call. If no response is receive, it shuts down the server process.

Type: `std::function<void()> onProcessTerminate`

Required: No

onHealthCheck

Name of callback function that the Amazon GameLift service invokes to request a health status report from the server process. Amazon GameLift calls this function every 60 seconds. After calling this function Amazon GameLift waits 60 seconds for a response, and if none is received. records the server process as unhealthy.

Type: `std::function<bool()> onHealthCheck`

Required: No

onUpdateGameSession

Name of callback function that the Amazon GameLift service invokes to pass an updated game session object to the server process. Amazon 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 Amazon GameLift service in a StartMatchBackfill() call.

Contents

GameSessionArn

Unique game session identifier. The API action GetGameSessionId() returns the identifier in ARN format.

Type: String

Required: Yes

MatchmakingConfigurationArn

Unique identifier, in the form of an ARN, for the matchmaker to use for this request. To find the matchmaker that was used to create the original game session, look in the game session object, in the matchmaker data property. Learn more about matchmaker data in Word with matchmaker data.

Type: String

Required: Yes

Players

A set of data representing all players who are currently in the game session. The matchmaker uses this information to search for new players who are good matches for the current players. See the *Amazon GameLift API Reference Guide* for a description of the Player object format. To find player attributes, IDs, and team assignments, look in the game session object, in the matchmaker data property. If latency is used by the matchmaker, gather updated latency for the current region and include it in each player's data.

Type: std:vector<player>

Required: Yes

TicketId

Unique identifier for a matchmaking or match backfill request ticket. If no value is provided here, Amazon GameLift will generate one in the form of a UUID. Use this identifier to track the match backfill ticket status or cancel the request if needed.

Type: String

Required: No

StopMatchBackfillRequest

This data type is used to cancel a matchmaking backfill request. The information is communicated to the Amazon GameLift service in a StopMatchBackfill() 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

Amazon GameLift server SDK reference for C#

You can use this Amazon GameLift C# server SDK reference to help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server.

Topics

- Amazon GameLift server SDK 5.x reference for C# and Unity
- Amazon GameLift server SDK 4.x reference for C#

Amazon GameLift server SDK 5.x reference for C# and Unity

You can use this Amazon GameLift C# server SDK 5.x reference to help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server and for information on using the C# server SDK plugin for Unity, see Integrate Amazon GameLift into a Unity project. The Amazon GameLift server SDK 5.x for C# supports .NET 4.6 and .NET 6.

Topics

- Amazon GameLift server SDK reference for C# and Unity: Actions
- Amazon GameLift server SDK reference for C# and Unity: Data types

Amazon GameLift server SDK reference for C# and Unity: Actions

This Amazon GameLift C# server SDK reference helps you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server and for information on using the C# server SDK plugin for Unity, see Integrate Amazon GameLift into a Unity project.

Actions

- GetSdkVersion()
- InitSDK()
- InitSDK()
- ProcessReady()
- ProcessEnding()
- ActivateGameSession()
- UpdatePlayerSessionCreationPolicy()
- GetGameSessionId()
- GetTerminationTime()
- AcceptPlayerSession()
- RemovePlayerSession()
- DescribePlayerSessions()
- StartMatchBackfill()
- StopMatchBackfill()
- GetComputeCertificate()
- GetFleetRoleCredentials()
- Destroy()

GetSdkVersion()

Returns the current version number of the SDK built into the server process.

Syntax

```
AwsStringOutcome GetSdkVersion();
```

Return value

If successful, returns the current SDK version as an the section called “AwsStringOutcome” object. The returned string includes the version number (example `5.0.0`). If not successful, returns an error message.

Example

```
var getSdkVersionOutcome = GameLiftServerAPI.GetSdkVersion();
```

InitSDK()

Initializes the Amazon GameLift SDK for a managed EC2 fleet. Call this method on launch, before any other initialization related to Amazon GameLift occurs. This method reads server parameters from the host environment to set up communication between the server and the Amazon GameLift service.

Syntax

```
GenericOutcome InitSDK();
```

Return value

If successful, returns an InitSdkOutcome object to indicate that the server process is ready to call ProcessReady().

Example

```
//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
GenericOutcome initSDKOutcome = GameLiftServerAPI.InitSDK();
```

InitSDK()

Initializes the Amazon GameLift SDK for an Anywhere fleet. Call this method on launch, before any other initialization related to Amazon GameLift occurs. This method requires explicit server parameters to set up communication between the server and the Amazon GameLift service.

Syntax

```
GenericOutcome InitSDK(ServerParameters serverParameters);
```

Parameters

ServerParameters

To initialize a game server on an Amazon GameLift Anywhere fleet, construct a `ServerParameters` object with the following information:

- The URL of the WebSocket used to connect to your game server.
- The ID of the process used to host your game server.
- The ID of the compute hosting your game server processes.
- The ID of the Amazon GameLift fleet containing your Amazon GameLift Anywhere compute.
- The authorization token generated by the Amazon GameLift operation.

Return value

If successful, returns an InitSdkOutcome object to indicate that the server process is ready to call ProcessReady().

> **Note**
>
> If calls to `InitSDK()` are failing for game builds deployed to Anywhere fleets, check the `ServerSdkVersion` parameter used when creating the build resource. You must explicitly set this value to the server SDK version in use. The default value for this parameter is 4.x, which is not compatible. To resolve this issue, create a new build and deploy it to a new fleet.

Example

```
//Define the server parameters
string websocketUrl = "wss://us-west-1.api.amazongamelift.com";
string processId = "PID1234";
string fleetId = "aarn:aws:gamelift:us-west-1:111122223333:fleet/
fleet-9999ffff-88ee-77dd-66cc-5555bbbb44aa";
```

```
string hostId = "HardwareAnywhere";
string authToken = "1111aaaa-22bb-33cc-44dd-5555eeee66ff";
ServerParameters serverParameters =
  new ServerParameters(webSocketUrl, processId, hostId, fleetId, authToken);

//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
GenericOutcome initSDKOutcome = GameLiftServerAPI.InitSDK(serverParameters);
```

ProcessReady()

Notifies Amazon GameLift that the server process is ready to host game sessions. Call this method after invoking InitSDK(). This method should be called only once per process.

Syntax

```
GenericOutcome ProcessReady(ProcessParameters processParameters)
```

Parameters

ProcessParameters

A `ProcessParameters` object holds information about the server process.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example illustrates both the method and delegate function implementations.

```
// Set parameters and call ProcessReady
ProcessParameters processParams = new ProcessParameters(
  this.OnStartGameSession,
  this.OnProcessTerminate,
  this.OnHealthCheck,
  this.OnUpdateGameSession,
  port,
  new LogParameters(new List<string>()
  // Examples of log and error files written by the game server
```

```
  {
    "C:\\game\\logs",
    "C:\\game\\error"
  })
);
GenericOutcome processReadyOutcome = GameLiftServerAPI.ProcessReady(processParams);
```

ProcessEnding()

Notifies Amazon GameLift that the server process is terminating. Call this method after all other cleanup tasks (including shutting down the active game session) and before terminating the process. Depending on the result of `ProcessEnding()`, the process exits with success (0) or error (-1) and generates a fleet event. If the process terminates with an error, the fleet event generated is `SERVER_PROCESS_TERMINATED_UNHEALTHY`.

Syntax

```
GenericOutcome ProcessEnding()
```

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example calls `ProcessEnding()` and `Destroy()` before terminating the server process with a success or error exit code.

```
GenericOutcome processEndingOutcome = GameLiftServerAPI.ProcessEnding();
GameLiftServerAPI.Destroy();

if (processEndingOutcome.Success)
  {
    Environment.Exit(0);
  }
else
  {
    Console.WriteLine("ProcessEnding() failed. Error: " +
 processEndingOutcome.Error.ToString());
    Environment.Exit(-1);
  }
```

ActivateGameSession()

Notifies Amazon GameLift that the server process has activated a game session and is now ready to receive player connections. This action should be called as part of the `onStartGameSession()` callback function, after all game session initialization.

Syntax

```
GenericOutcome ActivateGameSession()
```

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example shows `ActivateGameSession()` being called as part of the `onStartGameSession()` delegate function.

```
void OnStartGameSession(GameSession gameSession)
{
  // game-specific tasks when starting a new game session, such as loading map
  // When ready to receive players
  GenericOutcome activateGameSessionOutcome = GameLiftServerAPI.ActivateGameSession();
}
```

UpdatePlayerSessionCreationPolicy()

Updates the current game session's ability to accept new player sessions. A game session can be set to either accept or deny all new player sessions.

Syntax

```
GenericOutcome UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy
 playerSessionPolicy)
```

Parameters

playerSessionPolicy

String value that indicates whether the game session accepts new players.

Valid values include:

- **ACCEPT_ALL** – Accept all new player sessions.
- **DENY_ALL** – Deny all new player sessions.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example sets the current game session's join policy to accept all players.

```
GenericOutcome updatePlayerSessionPolicyOutcome =

 GameLiftServerAPI.UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy.ACCEPT_ALL);
```

GetGameSessionId()

Retrieves the ID of the game session hosted by the active server process.

For idle processes that aren't activated with a game session, the call returns a the section called “GameLiftError”.

Syntax

```
AwsStringOutcome GetGameSessionId()
```

Return value

If successful, returns the game session ID as an the section called “AwsStringOutcome” object. If not successful, returns an error message."

Example

```
AwsStringOutcome getGameSessionIdOutcome = GameLiftServerAPI.GetGameSessionId();
```

GetTerminationTime()

Returns the time that a server process is scheduled to be shut down, if a termination time is available. A server process takes this action after receiving an onProcessTerminate() callback

from Amazon GameLift. Amazon GameLift calls on`ProcessTerminate()` for the following reasons:

- When the server process has reported poor health or has not responded to Amazon GameLift.
- When terminating the instance during a scale-down event.
- When an instance is terminated due to a spot-instance interruption.

Syntax

```
AwsDateTimeOutcome GetTerminationTime()
```

Return value

If successful, returns the termination time as an the section called "AwsDateTimeOutcome" object. The value is the termination time, expressed in elapsed ticks since `0001 00:00:00`. For example, the date time value `2020-09-13 12:26:40 -000Z` is equal to `637355968000000000` ticks. If no termination time is available, returns an error message.

Example

```
AwsDateTimeOutcome getTerminationTimeOutcome = GameLiftServerAPI.GetTerminationTime();
```

AcceptPlayerSession()

Notifies Amazon GameLift that a player with the specified player session ID has connected to the server process and needs validation. Amazon GameLift verifies that the player session ID is valid. After the player session is validated, Amazon GameLift changes the status of the player slot from RESERVED to ACTIVE.

Syntax

```
GenericOutcome AcceptPlayerSession(String playerSessionId)
```

Parameters

playerSessionId

Unique ID issued by GameLift when a new player session is created.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example illustrates a function for handling a connection request, including validating and rejecting invalid player session IDs.

```
void ReceiveConnectingPlayerSessionID (Connection connection, String playerSessionId)
{
  GenericOutcome acceptPlayerSessionOutcome =
 GameLiftServerAPI.AcceptPlayerSession(playerSessionId);
  if(acceptPlayerSessionOutcome.Success)
  {
    connectionToSessionMap.emplace(connection, playerSessionId);
    connection.Accept();
  }
  else
  {
    connection.Reject(acceptPlayerSessionOutcome.Error.ErrorMessage);
  }
}
```

RemovePlayerSession()

Notifies Amazon GameLift that a player has disconnected from the server process. In response, Amazon GameLift changes the player slot to available.

Syntax

```
GenericOutcome RemovePlayerSession(String playerSessionId)
```

Parameters

playerSessionId

Unique ID issued by Amazon GameLift when a new player session is created.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
GenericOutcome removePlayerSessionOutcome =
 GameLiftServerAPI.RemovePlayerSession(playerSessionId);
```

DescribePlayerSessions()

Retrieves player session data which includes settings, session metadata, and player data. Use this action to get information for a single player session, for all player sessions in a game session, or for all player sessions associated with a single player ID.

Syntax

```
DescribePlayerSessionsOutcome DescribePlayerSessions(DescribePlayerSessionsRequest
 describePlayerSessionsRequest)
```

Parameters

DescribePlayerSessionsRequest

A the section called "DescribePlayerSessionsRequest" object that describes which player sessions to retrieve.

Return value

If successful, returns a the section called "DescribePlayerSessionsOutcome" object that contains a set of player session objects that fit the request parameters.

Example

This example illustrates a request for all player sessions actively connected to a specified game session. By omitting *NextToken* and setting the *Limit* value to 10, Amazon GameLift will return the first 10 player session records matching the request.

```
// Set request parameters
DescribePlayerSessionsRequest describePlayerSessionsRequest = new
 DescribePlayerSessionsRequest()
{
  GameSessionId = GameLiftServerAPI.GetGameSessionId().Result,    //gets the ID for the
 current game session
```

```
  Limit = 10,
  PlayerSessionStatusFilter =
    PlayerSessionStatusMapper.GetNameForPlayerSessionStatus(PlayerSessionStatus.ACTIVE)
};
// Call DescribePlayerSessions
DescribePlayerSessionsOutcome describePlayerSessionsOutcome =
  GameLiftServerAPI.DescribePlayerSessions(describePlayerSessionsRequest);
```

StartMatchBackfill()

Sends a request to find new players for open slots in a game session created with FlexMatch. For more information, see FlexMatch backfill feature.

This action is asynchronous. If new players are matched, Amazon GameLift delivers updated matchmaker data using the callback function `OnUpdateGameSession()`.

A server process can have only one active match backfill request at a time. To send a new request, first call StopMatchBackfill() to cancel the original request.

Syntax

```
StartMatchBackfillOutcome StartMatchBackfill (StartMatchBackfillRequest
 startBackfillRequest);
```

Parameters

StartMatchBackfillRequest

A `StartMatchBackfillRequest` object holds information about the backfill request.

Return value

Returns a the section called “StartMatchBackfillOutcome” object with the match backfill ticket ID, or failure with an error message.

Example

```
// Build a backfill request
StartMatchBackfillRequest startBackfillRequest = new StartMatchBackfillRequest()
{
```

```
  TicketId = "1111aaaa-22bb-33cc-44dd-5555eeee66ff", //optional
  MatchmakingConfigurationArn = "arn:aws:gamelift:us-
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig",
  GameSessionId = GameLiftServerAPI.GetGameSessionId().Result,    // gets ID for
 current game session
  MatchmakerData matchmakerData =
    MatchmakerData.FromJson(gameSession.MatchmakerData),  // gets matchmaker data for
 current players
  // get matchmakerData.Players
  // remove data for players who are no longer connected
  Players = ListOfPlayersRemainingInTheGame
};

// Send backfill request
StartMatchBackfillOutcome startBackfillOutcome =
 GameLiftServerAPI.StartMatchBackfill(startBackfillRequest);

// Implement callback function for backfill
void OnUpdateGameSession(GameSession myGameSession)
{
  // game-specific tasks to prepare for the newly matched players and update matchmaker
 data as needed
}
```

StopMatchBackfill()

Cancels an active match backfill request. For more information, see FlexMatch backfill feature.

Syntax

```
GenericOutcome StopMatchBackfill (StopMatchBackfillRequest stopBackfillRequest);
```

Parameters

StopMatchBackfillRequest

A `StopMatchBackfillRequest` object that provides details about the matchmaking ticket you are stopping.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
// Set backfill stop request parameters
StopMatchBackfillRequest stopBackfillRequest = new StopMatchBackfillRequest(){
  TicketId = "1111aaaa-22bb-33cc-44dd-5555eeee66ff", //optional, if not provided one is
 autogenerated
  MatchmakingConfigurationArn = "arn:aws:gamelift:us-
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig",
  GameSessionId = GameLiftServerAPI.GetGameSessionId().Result    //gets the ID for the
 current game session
};
GenericOutcome stopBackfillOutcome =
 GameLiftServerAPI.StopMatchBackfillRequest(stopBackfillRequest);
```

GetComputeCertificate()

Retrieves the path to the TLS certificate used to encrypt the network connection between the game server and your game client. You can use the certificate path when you register your compute device to a Amazon GameLift Anywhere fleet. For more information see, RegisterCompute.

Syntax

```
GetComputeCertificateOutcome GetComputeCertificate();
```

Return value

Returns a GetComputeCertificateResponse object that contains the following:

- CertificatePath: The path to the TLS certificate on your compute resource. When using an Amazon GameLift managed fleet, this path contains:
  - `certificate.pem`: The end-user certificate. The full certificate chain is the combination of `certificateChain.pem` appended to this certificate.
  - `certificateChain.pem`: The certificate chain that contains the root certificate and intermediate certificates.
  - `rootCertificate.pem`: The root certificate.
  - `privateKey.pem`: The private key for the end-user certificate.
- ComputeName: The name of your compute resource.

Example

```
GetComputeCertificateOutcome getComputeCertificateOutcome =
 GameLiftServerAPI.GetComputeCertificate();
```

GetFleetRoleCredentials()

Retrieves IAM role credentials that authorize Amazon GameLift to interact with other AWS services. For more information, see Communicate with other AWS resources from your fleets.

Syntax

```
GetFleetRoleCredentialsOutcome GetFleetRoleCredentials(GetFleetRoleCredentialsRequest
 request);
```

Parameters

GetFleetRoleCredentialsRequest

Role credentials that extend limited access to your AWS resources to the game server.

Return value

Returns a the section called “GetFleetRoleCredentialsOutcome” object.

Example

```
// form the fleet credentials request
GetFleetRoleCredentialsRequest getFleetRoleCredentialsRequest = new
 GetFleetRoleCredentialsRequest(){
  RoleArn = "arn:aws:iam::123456789012:role/service-role/exampleGameLiftAction"
};
GetFleetRoleCredentialsOutcome GetFleetRoleCredentialsOutcome credentials =
 GetFleetRoleCredentials(getFleetRoleCredentialsRequest);
```

Destroy()

Frees the Amazon GameLift game server SDK from memory. As a best practice, call this method after `ProcessEnding()` and before terminating the process. If you're using an Anywhere fleet and you're not terminating server processes after every game session, call `Destroy()` and then

`InitSDK()` to reinitialize before notifying Amazon GameLift that the process is ready to host a game session with `ProcessReady()`.

Syntax

```
GenericOutcome Destroy()
```

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
// Operations to end game sessions and the server process
GenericOutcome processEndingOutcome = GameLiftServerAPI.ProcessEnding();

// Shut down and destroy the instance of the GameLift Game Server SDK
GenericOutcome destroyOutcome = GameLiftServerAPI.Destroy();

// Exit the process with success or failure
if (processEndingOutcome.Success)
  {
    Environment.Exit(0);
  }
else
  {
    Console.WriteLine("ProcessEnding() failed. Error: " +
 processEndingOutcome.Error.ToString());
    Environment.Exit(-1);
  }
```

Amazon GameLift server SDK reference for C# and Unity: Data types

This Amazon GameLift C# Server SDK reference can help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server and for information on using the C# server SDK plugin for Unity, see Integrate Amazon GameLift into a Unity project .

Data types

- LogParameters
- ProcessParameters

- UpdateGameSession
- GameSession
- ServerParameters
- StartMatchBackfillRequest
- Player
- DescribePlayerSessionsRequest
- StopMatchBackfillRequest
- GetFleetRoleCredentialsRequest
- AttributeValue
- AwsStringOutcome
- GenericOutcome
- DescribePlayerSessionsOutcome
- DescribePlayerSessionsResult
- PlayerSession
- StartMatchBackfillOutcome
- StartMatchBackfillResult
- GetComputeCertificateOutcome
- GetComputeCertificateResult
- GetFleetRoleCredentialsOutcome
- GetFleetRoleCredentialsResult
- AwsDateTimeOutcome
- GameLiftError
- Enums

LogParameters

Use this data type to identify which files generated during a game session that you want the game server to upload to Amazon GameLift after the game session ends. The game server communicates `LogParameters` to Amazon GameLift in a ProcessReady() call.

Properties	Description

LogPaths	The list of directory paths to game server log files you want Amazon GameLift to store for future access. The server process generates these files during each game session. You define file paths and names in your game server and store them in the root game build directory. The log paths must be absolute. For example, if your game build stores game session logs in a path like `MyGame\sessionLogs\` , then the path would be `c:\game\MyGame\sessionLogs` on a Windows instance. **Type:** `List<String>` **Required:** No

ProcessParameters

This data type contains the set of parameters sent to Amazon GameLift in a ProcessReady() call.

Properties	Description
LogParameters	The object with a list of directory paths to game session log files. **Type:** `Aws::GameLift::Server::` LogParameters **Required:** Yes
OnHealthCheck	The name of callback function that Amazon GameLift invokes to request a health status report from the server process. Amazon GameLift calls this function every 60 seconds. After calling this function Amazon GameLift waits 60 seconds for a response, if none is

	received, Amazon GameLift records the server process as unhealthy. **Type:** `void OnHealthCheckDelegate()` **Required:** Yes
OnProcessTerminate	The name of callback function that Amazon GameLift invokes to force the server process to shut down. After calling this function, Amazon GameLift waits five minutes for the server process to shut down and respond with a ProcessEnding() call before it shuts down the server process. **Type:** `void OnProcessTerminate Delegate()` **Required:** Yes
OnStartGameSession	The name of callback function that Amazon GameLift invokes to activate a new game session. Amazon GameLift calls this function in response to the client request CreateGameSession. The callback function takes a GameSession object defined in the Amazon GameLift API Reference. **Type:** `void OnStartGameSession Delegate( GameSession )` **Required:** Yes

OnUpdateGameSession	The name of callback function that Amazon GameLift invokes to pass an updated game session object to the server process. Amazon GameLift calls this function when a match backfill request has been processed to provide updated matchmaker data. It passes a GameSession object, a status update (`updateReason`), and the match backfill ticket ID. **Type:** void OnUpdateGameSessionDelegate (UpdateGameSession) **Required:** No
Port	The port number that the server process listens on for new player connections. The value must fall into the port range configure d for any fleet deploying this game server build. This port number is included in game session and player session objects, which game sessions use when connecting to a server process. **Type:** `Integer` **Required:** Yes

UpdateGameSession

Updated information for a game session object, includes the reason that the game session was updated. If the update is related to a match backfill action, this data type includes the backfill ticket ID.

Properties	Description
GameSession	A GameSession object defined by the Amazon GameLift API. The `GameSession` object contains properties describing a game session. **Type:** `GameSession GameSession()` **Required:** Yes
UpdateReason	The reason that the game session is being updated. **Type:** `UpdateReason UpdateReason()` **Required:** Yes
BackfillTicketId	The ID of the backfill ticket attempting to update the game session. **Type:** `String` **Required:** Yes

GameSession

Details of a game session.

Properties	Description
GameSessionId	A unique identifier for the game session. A game session ARN has the following format: `arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token> .` **Type:** `String` **Required**: No

Properties	Description
Name	A descriptive label of the game session. **Type:** `String` **Required**: No
FleetId	A unique identifier for the fleet that the game session is running on. **Type:** `String` **Required**: No
MaximumPlayerSessionCount	The maximum number of player connections to the game session. **Type:** `Integer` **Required**: No
Port	The port number for the game session. To connect to a Amazon GameLift game server, an app needs both the IP address and port number. **Type:** `Integer` **Required**: No
IpAddress	The IP address of the game session. To connect to a Amazon GameLift game server, an app needs both the IP address and port number. **Type:** `String` **Required**: No

Properties	Description
GameSessionData	A set of custom game session properties, formatted as a single string value. **Type:** `String` **Required**: No
MatchmakerData	The information about the matchmaking process that was used to create the game session, in JSON syntax, formatted as a string. In addition the matchmaking configuration used, it contains data on all players assigned to the match, including player attributes and team assignments. **Type:** `String` **Required**: No
GameProperties	A set of custom properties for a game session, formatted as key:value pairs. These properties are passed with a request to start a new game session. **Type:** `Dictionary<string, string>` **Required**: No

Properties	Description
DnsName	The DNS identifier assigned to the instance that's running the game session. Values have the following format: • TLS-enabled fleets: `<unique identifier>.<region identifier>.amazongamelift.com .` • Non-TLS-enabled fleets: `ec2-<unique identifier>.compute.amazonaws.com .` When connecting to a game session that's running on a TLS-enabled fleet, you must use the DNS name, not the IP address. **Type:** `String` **Required**: No

ServerParameters

Information used to maintain the connection between an Amazon GameLift Anywhere server and the Amazon GameLift service. This information is used when launching new server processes with InitSDK(). For servers hosted on Amazon GameLift managed EC2 instances, use an empty object.

Properties	Description
WebSocketUrl	The `GameLiftServerSdkEndpoint` returned when you `RegisterCompute` as part of Amazon GameLift Anywhere. **Type:** `String` **Required**: Yes

Properties	Description
ProcessId	A unique identifier registered to the server process hosting your game. **Type:** `String` **Required**: Yes
HostId	A unique identifier for the host with the server processes hosting your game. The hostId is the ComputeName used when you registered your compute. For more information see, RegisterCompute **Type:** `String` **Required**: Yes
FleetId	The fleet ID of the fleet that the compute is registered to. For more information see, RegisterCompute. **Type:** `String` **Required**: Yes
AuthToken	The authentication token generated by Amazon GameLift that authenticates your server to Amazon GameLift. For more information see, GetComputeAuthToken. **Type:** `String` **Required**: Yes

StartMatchBackfillRequest

Information used to create a matchmaking backfill request. The game server communicates this information to Amazon GameLift in a StartMatchBackfill() call.

Properties	Description
GameSessionArn	The unique game session identifier. The API operation `GetGameSessionId` returns the identifier in ARN format. **Type:** `String` **Required**: Yes
MatchmakingConfigurationArn	The unique identifier, in the form of an ARN, for the matchmaker to use for this request. The matchmaker ARN for the original game session is in the game session object in the matchmaker data property. Learn more about matchmaker data in Work with matchmaker data. **Type:** `String` **Required**: Yes
Players	A set of data that represents all players who are currently in the game session. The matchmaker uses this information to search for new players who are good matches for the current players. **Type:** `List<Player>` **Required**: Yes
TicketId	The unique identifier for a matchmaking or match backfill request ticket. If you don't provide a value, Amazon GameLift generates one. Use this identifier to track the match backfill ticket status or cancel the request if needed.

Properties	Description
	Type: `String` **Required**: No

Player

Represents a player in matchmaking. When a matchmaking request starts, a player has a player ID, attributes, and possibly latency data. Amazon GameLift adds team information after a match is made.

Properties	Description
LatencyInMS	A set of values expressed in milliseconds, that indicate the amount of latency that a player experiences when connected to a location. If this property is used, the player is only matched for locations listed. If a matchmaker has a rule that evaluates player latency, players must report latency to be matched. **Type:** `Dictionary<string, int>` **Required**: No
PlayerAttributes	A collection of key:value pairs that contain player information for use in matchmaking. Player attribute keys must match the PlayerAttributes used in a matchmaking rule set. For more information about player attributes, see AttributeValue. **Type:** `Dictionary<string, AttributeValue` **Required**: No

Properties	Description
PlayerId	A unique identifier for a player. **Type:** `String` **Required**: No
Team	The name of the team that the player is assigned to in a match. You define team name in the matchmaking rule set. **Type:** `String` **Required**: No

DescribePlayerSessionsRequest

This data type is used to specify which player session(s) to retrieve. It can be used in several ways: (1) provide a PlayerSessionId to request a specific player session; (2) provide a GameSessionId to request all player sessions in the specified game session; or (3) provide a PlayerId to request all player sessions for the specified player. For large collections of player sessions, use the pagination parameters to retrieve results as sequential pages.

Properties	Description
GameSessionId	The unique game session identifier. Use this parameter to request all player sessions for the specified game session. Game session ID format is as follows: `arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>` . The value of <ID string> is either a custom ID string (if one was specified when the game session was created) a generated string. **Type:** `String`

Properties	Description
	Required: No
PlayerSessionId	The unique identifier for a player session. **Type:** `String` **Required**: No
PlayerId	The unique identifier for a player. See Generate player IDs. **Type:** `String` **Required**: No
PlayerSessionStatusFilter	The player session status to filter results on. Possible player session statuses include the following: • RESERVED – The player session request has been received, but the player has not yet connected to the server process and/or been validated. • 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

Properties	Description
NextToken	The token indicating the start of the next page of results. To specify the start of the result set, don't provide a value. If you provide a player session ID, this parameter is ignored. **Type:** `String` **Required**: No
Limit	The maximum number of results to return. If you provide a player session ID, this parameter is ignored. **Type:** `int` **Required**: No

StopMatchBackfillRequest

Information used to cancel a matchmaking backfill request. The game server communicates this information to Amazon GameLift service in a StopMatchBackfill() call.

Properties	Description
GameSessionArn	The unique game session identifier of the request being canceled. **Type:** `string` **Required**: Yes
MatchmakingConfigurationArn	The unique identifier of the matchmaker this request was sent to. **Type:** `string` **Required**: Yes

Properties	Description
TicketId	The unique identifier of the backfill request ticket to be canceled. **Type:** `string` **Required**: Yes

GetFleetRoleCredentialsRequest

This data type gives the game server limited access to your other AWS resources. For more information see, Set up an IAM service role for Amazon GameLift.

Properties	Description
RoleArn	The Amazon Resource Name (ARN) of the service role that extends limited access to your AWS resources. **Type:** `string` **Required**: Yes
RoleSessionName	The name of the session that describes the use of the role credentials. **Type:** `string` **Required**: No

AttributeValue

Use these values in Player attribute key-value pairs. This object lets you specify an attribute value using any of the valid data types: string, number, string array, or data map. Each `AttributeValue` object can use only one of the available properties.

Properties	Description
attrType	Specifies the type of attribute value. **Type:** An `AttrType` enum value. **Required:** No
S	Represents a string attribute value. **Type:** `string` **Required:** Yes
N	Represents a numeric attribute value. **Type:** `double` **Required:** Yes
SL	Represents an array of string attribute values. **Type:** `string[]` **Required:** Yes
SDM	Represents a dictionary of string keys and double values. **Type:** `Dictionary<string, double>` **Required:** Yes

AwsStringOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action.

Properties	Description
	Type: `string` **Required:** No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called “GameLiftError” **Required:** No

GenericOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called “GameLiftError” **Required:** No

DescribePlayerSessionsOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. **Type:** the section called "DescribePlayerSessionsResult" **Required:** No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called "GameLiftError" **Required:** No

DescribePlayerSessionsResult

Properties	Description
NextToken	The token indicating the start of the next page of results. To specify the start of the result set, don't provide a value. If you provide a player session ID, this parameter is ignored. **Type:** `string` **Required:** Yes

Properties	Description
PlayerSessions	A collection of objects containing properties for each player session that matches the request. **Type:** `IList<`the section called "PlayerSession"`>` **Required:**
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called "GameLiftError" **Required:** No

PlayerSession

Properties	Description
CreationTime	**Type:** `long` **Required:** Yes
FleetId	**Type:** `string` **Required:** Yes
GameSessionId	**Type:** `string` **Required:** Yes

Properties	Description
IpAddress	**Type:** `string` **Required:** Yes
PlayerData	**Type:** `string` **Required:** Yes
PlayerId	**Type:** `string` **Required:** Yes
PlayerSessionId	**Type:** `string` **Required:** Yes
Port	**Type:** `int` **Required:** Yes
Status	**Type:** A `PlayerSessionStatus` enum. **Required:** Yes
TerminationTime	**Type:** `long` **Required:** Yes
DnsName	**Type:** `string` **Required:** Yes

StartMatchBackfillOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action.

Properties	Description
	Type: the section called "StartMatchBackfillResult" **Required:** No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called "GameLiftError" **Required:** No

StartMatchBackfillResult

Properties	Description
TicketId	**Type:** `string` **Required:** Yes

GetComputeCertificateOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. **Type:** the section called "GetComputeCertificateResult"

Properties	Description
	Required: No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called “GameLiftError” **Required:** No

GetComputeCertificateResult

The path to the TLS certificate on your compute and the compute's host name.

Properties	Description
CertificatePath	**Type:** `string` **Required:** Yes
ComputeName	**Type:** `string` **Required:** Yes

GetFleetRoleCredentialsOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action.

Properties	Description
	Type: the section called "GetFleetRoleCredentialsResult" **Required:** No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called "GameLiftError" **Required:** No

GetFleetRoleCredentialsResult

Properties	Description
AccessKeyId	The access key ID to authenticate and provide access to your AWS resources. **Type:** `string` **Required:** No
AssumedRoleId	The ID of the user that the service role belongs to. **Type:** `string` **Required:** No
AssumedRoleUserArn	The Amazon Resource Name (ARN) of the user that the service role belongs to.

Properties	Description
	Type: `string` **Required:** No
Expiration	The amount of time until your session credentials expire. **Type:** `DateTime` **Required:** No
SecretAccessKey	The secret access key ID for authentication. **Type:** `string` **Required:** No
SessionToken	A token to identify the current active session interacting with your AWS resources. **Type:** `string` **Required:** No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called “GameLiftError” **Required:** No

AwsDateTimeOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. **Type:** `DateTime` **Required:** No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called "GameLiftError" **Required:** No

GameLiftError

Properties	Description
ErrorType	The type of error. **Type:** A `GameLiftErrorType` enum. **Required:** No
ErrorName	The name of the error. **Type:** `string` **Required:** No
ErrorMessage	The error message. **Type:** `string`

Properties	Description
	Required: No

Enums

Enums defined for the Amazon GameLift server SDK (C#) are defined as follows:

AttrType

- **NONE**
- **STRING**
- **DOUBLE**
- **STRING_LIST**
- **STRING_DOUBLE_MAP**

GameLiftErrorType

String value indicating the error type. Valid values include:

- **SERVICE_CALL_FAILED** – A call to an AWS service has failed.
- **LOCAL_CONNECTION_FAILED** – The local connection to Amazon GameLift failed.
- **NETWORK_NOT_INITIALIZED** – The network has not been initialized.
- **GAMESESSION_ID_NOT_SET** – The game session ID has not been set.
- **BAD_REQUEST_EXCEPTION**
- **INTERNAL_SERVICE_EXCEPTION**
- **ALREADY_INITIALIZED** – The Amazon GameLift Server or Client has already been initialized with Initialize().
- **FLEET_MISMATCH** – The target fleet does not match the fleet of a gameSession or playerSession.
- **GAMELIFT_CLIENT_NOT_INITIALIZED** – The Amazon GameLift client has not been initialized.
- **GAMELIFT_SERVER_NOT_INITIALIZED** – The Amazon GameLift server has not been initialized.
- **GAME_SESSION_ENDED_FAILED** – The Amazon GameLift Server SDK could not contact the service to report the game session ended.
- **GAME_SESSION_NOT_READY** – The Amazon GameLift Server Game Session was not activated.

- **GAME_SESSION_READY_FAILED** – The Amazon GameLift Server SDK could not contact the service to report the game session is ready.
- **INITIALIZATION_MISMATCH** – A client method was called after Server::Initialize(), or vice versa.
- **NOT_INITIALIZED** – The Amazon GameLift Server or Client has not been initialized with Initialize().
- **NO_TARGET_ALIASID_SET** – A target aliasId has not been set.
- **NO_TARGET_FLEET_SET** – A target fleet has not been set.
- **PROCESS_ENDING_FAILED** – The Amazon GameLift Server SDK could not contact the service to report the process is ending.
- **PROCESS_NOT_ACTIVE** – The server process is not yet active, not bound to a GameSession, and cannot accept or process PlayerSessions.
- **PROCESS_NOT_READY** – The server process is not yet ready to be activated.
- **PROCESS_READY_FAILED** – The Amazon GameLift Server SDK could not contact the service to report the process is ready.
- **SDK_VERSION_DETECTION_FAILED** – SDK version detection failed.
- **STX_CALL_FAILED** – A call to the XStx server backend component has failed.
- **STX_INITIALIZATION_FAILED** – The XStx server backend component has failed to initialize.
- **UNEXPECTED_PLAYER_SESSION** – An unregistered player session was encountered by the server.
- **WEBSOCKET_CONNECT_FAILURE**
- **WEBSOCKET_CONNECT_FAILURE_FORBIDDEN**
- **WEBSOCKET_CONNECT_FAILURE_INVALID_URL**
- **WEBSOCKET_CONNECT_FAILURE_TIMEOUT**
- **WEBSOCKET_RETRIABLE_SEND_MESSAGE_FAILURE** – Retriable failure to send a message to the GameLift Service WebSocket.
- **WEBSOCKET_SEND_MESSAGE_FAILURE** – Failure to send a message to the GameLift Service WebSocket.
- **MATCH_BACKFILL_REQUEST_VALIDATION** – Validation of the request failed.
- **PLAYER_SESSION_REQUEST_VALIDATION** – Validation of the request failed.

PlayerSessionCreationPolicy

String value indicating whether the game session accepts new players. Valid values include:

- **ACCEPT_ALL** – Accept all new player sessions.
- **DENY_ALL** – Deny all new player sessions.
- **NOT_SET** – The game session is not set to accept or deny new player sessions.

PlayerSessionStatus

- **ACTIVE**
- **COMPLETED**
- **NOT_SET**
- **RESERVED**
- **TIMEDOUT**

Amazon GameLift server SDK 4.x reference for C#

This Amazon GameLift C# Server SDK 4.x reference can help you prepare your multiplayer game for use with Amazon GameLift. For details on the integration process, see Add Amazon GameLift to your game server.

Topics

- Amazon GameLift server SDK (C#) reference: Actions
- Amazon GameLift server SDK (C#) reference: Data types

Amazon GameLift server SDK (C#) reference: Actions

You can use this Amazon GameLift C# server SDK reference to help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server.

- Actions
- Data Types

AcceptPlayerSession()

Notifies the Amazon GameLift service that a player with the specified player session ID has connected to the server process and needs validation. Amazon GameLift verifies that the player session ID is valid—that is, that the player ID has reserved a player slot in the game session. Once validated, Amazon GameLift changes the status of the player slot from RESERVED to ACTIVE.

Syntax

```
GenericOutcome AcceptPlayerSession(String playerSessionId)
```

Parameters

playerSessionId

Unique ID issued by Amazon 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: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example illustrates a function for handling a connection request, including validating and rejecting invalid player session IDs.

```
void ReceiveConnectingPlayerSessionID (Connection connection, String playerSessionId){
    var acceptPlayerSessionOutcome =
 GameLiftServerAPI.AcceptPlayerSession(playerSessionId);
     if(acceptPlayerSessionOutcome.Success)
    {
        connectionToSessionMap.emplace(connection, playerSessionId);
        connection.Accept();
    }
     else
    {
        connection.Reject(acceptPlayerSessionOutcome.Error.ErrorMessage);    }
}
```

ActivateGameSession()

Notifies the Amazon GameLift service that the server process has activated a game session and is now ready to receive player connections. This action should be called as part of the `onStartGameSession()` callback function, after all game session initialization has been completed.

Syntax

```
GenericOutcome ActivateGameSession()
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example shows `ActivateGameSession()` being called as part of the `onStartGameSession()` delegate function.

```
void OnStartGameSession(GameSession gameSession)
{
    // game-specific tasks when starting a new game session, such as loading map

    // When ready to receive players
    var activateGameSessionOutcome = GameLiftServerAPI.ActivateGameSession();
}
```

DescribePlayerSessions()

Retrieves player session data, including settings, session metadata, and player data. Use this action to get information for a single player session, for all player sessions in a game session, or for all player sessions associated with a single player ID.

Syntax

```
DescribePlayerSessionsOutcome DescribePlayerSessions(DescribePlayerSessionsRequest
 describePlayerSessionsRequest)
```

Parameters

describePlayerSessionsRequest

A DescribePlayerSessionsRequest object describing which player sessions to retrieve.

Required: Yes

Return value

If successful, returns a `DescribePlayerSessionsOutcome` object containing a set of player session objects that fit the request parameters. Player session objects have a structure identical to the AWS SDK Amazon GameLift API PlayerSession data type.

Example

This example illustrates a request for all player sessions actively connected to a specified game session. By omitting *NextToken* and setting the *Limit* value to 10, Amazon GameLift will return the first 10 player sessions records matching the request.

```
// Set request parameters
var describePlayerSessionsRequest = new
 Aws.GameLift.Server.Model.DescribePlayerSessionsRequest()
{
    GameSessionId = GameLiftServerAPI.GetGameSessionId().Result,    //gets the ID for
 the current game session
    Limit = 10,
    PlayerSessionStatusFilter =
 PlayerSessionStatusMapper.GetNameForPlayerSessionStatus(PlayerSessionStatus.ACTIVE)
};
// Call DescribePlayerSessions
Aws::GameLift::DescribePlayerSessionsOutcome playerSessionsOutcome =
    Aws::GameLift::Server::Model::DescribePlayerSessions(describePlayerSessionRequest);
```

GetGameSessionId()

Retrieves the ID of the game session currently being hosted by the server process, if the server process is active.

For idle process that are not yet activated with a game session, the call returns `Success=True` and `GameSessionId=""` (an empty string).

Syntax

```
AwsStringOutcome GetGameSessionId()
```

Parameters

This action has no parameters.

Return value

If successful, returns the game session ID as an `AwsStringOutcome` object. If not successful, returns an error message.

Example

```
var getGameSessionIdOutcome = GameLiftServerAPI.GetGameSessionId();
```

GetInstanceCertificate()

Retrieves the file location of a pem-encoded TLS certificate that is associated with the fleet and its instances. AWS Certificate Manager generates this certificate when you create a new fleet with the certificate configuration set to GENERATED. Use this certificate to establish a secure connection with a game client and to encrypt client/server communication.

Syntax

```
GetInstanceCertificateOutcome GetInstanceCertificate();
```

Parameters

This action has no parameters.

Return value

If successful, returns a `GetInstanceCertificateOutcome` object containing the location of the fleet's TLS certificate file and certificate chain, which are stored on the instance. A root certificate file, extracted from the certificate chain, is also stored on the instance. If not successful, returns an error message.

For more information about the certificate and certificate chain data, see GetCertificate Response Elements in the AWS Certificate Manager API Reference.

Example

```
var getInstanceCertificateOutcome = GameLiftServerAPI.GetInstanceCertificate();
```

GetSdkVersion()

Returns the current version number of the SDK built into the server process.

Syntax

```
AwsStringOutcome GetSdkVersion()
```

Parameters

This action has no parameters.

Return value

If successful, returns the current SDK version as an `AwsStringOutcome` object. The returned string includes the version number only (ex. "3.1.5"). If not successful, returns an error message.

Example

```
var getSdkVersionOutcome = GameLiftServerAPI.GetSdkVersion();
```

GetTerminationTime()

Returns the time that a server process is scheduled to be shut down, if a termination time is available. A server process takes this action after receiving an `onProcessTerminate()` callback from the Amazon GameLift service. Amazon GameLift may call `onProcessTerminate()` for the following reasons: (1) for poor health (the server process has reported port health or has not responded to Amazon GameLift, (2) when terminating the instance during a scale-down event, or (3) when an instance is being terminated due to a spot-instance interruption.

If the process has received an `onProcessTerminate()` callback, the value returned is the estimated termination time. If the process has not received an `onProcessTerminate()` callback, an error message is returned. Learn more about shutting down a server process.

Syntax

```
AwsDateTimeOutcome GetTerminationTime()
```

Parameters

This action has no parameters.

Return value

If successful, returns the termination time as an `AwsDateTimeOutcome` object. The value is the termination time, expressed in elapsed ticks since 0001 00:00:00. For example, the date time value 2020-09-13 12:26:40 -000Z is equal to 637355968000000000 ticks. If no termination time is available, returns an error message.

Example

```
var getTerminationTimeOutcome = GameLiftServerAPI.GetTerminationTime();
```

InitSDK()

Initializes the Amazon GameLift SDK. This method should be called on launch, before any other Amazon GameLift-related initialization occurs.

Syntax

```
InitSDKOutcome InitSDK()
```

Parameters

This action has no parameters.

Return value

If successful, returns an InitSdkOutcome object indicating that the server process is ready to call ProcessReady().

Example

```
var initSDKOutcome = GameLiftServerAPI.InitSDK();
```

ProcessEnding()

Notifies the Amazon GameLift 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

```
GenericOutcome ProcessEnding()
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
var processEndingOutcome = GameLiftServerAPI.ProcessEnding();
if (processReadyOutcome.Success)
   Environment.Exit(0);
// otherwise, exit with error code
Environment.Exit(errorCode);
```

ProcessReady()

Notifies the Amazon GameLift service that the server process is ready to host game sessions. Call this method after successfully invoking InitSDK() and completing setup tasks that are required before the server process can host a game session. This method should be called only once per process.

Syntax

```
GenericOutcome ProcessReady(ProcessParameters processParameters)
```

Parameters

processParameters

A ProcessParameters object communicating the following information about the server process:

- Names of callback methods, implemented in the game server code, that the Amazon GameLift service invokes to communicate with the server process.
- Port number that the server process is listening on.
- Path to any game session-specific files that you want Amazon GameLift to capture and store.

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example illustrates both the ProcessReady() call and delegate function implementations.

```
// Set parameters and call ProcessReady
var processParams = new ProcessParameters(
   this.OnGameSession,
   this.OnProcessTerminate,
   this.OnHealthCheck,
   this.OnGameSessionUpdate,
   port,
   new LogParameters(new List<string>()          // Examples of log and error files
 written by the game server
   {
      "C:\\game\\logs",
      "C:\\game\\error"
   })
);

var processReadyOutcome = GameLiftServerAPI.ProcessReady(processParams);

// Implement callback functions
void OnGameSession(GameSession gameSession)
{
   // game-specific tasks when starting a new game session, such as loading map
```

```
   // When ready to receive players
   var activateGameSessionOutcome = GameLiftServerAPI.ActivateGameSession();
}

void OnProcessTerminate()
{
   // game-specific tasks required to gracefully shut down a game session,
   // such as notifying players, preserving game state data, and other cleanup
    var ProcessEndingOutcome = GameLiftServerAPI.ProcessEnding();
}

bool OnHealthCheck()
{
    bool isHealthy;
    // complete health evaluation within 60 seconds and set health
    return isHealthy;
}
```

RemovePlayerSession()

Notifies the Amazon GameLift service that a player with the specified player session ID has disconnected from the server process. In response, Amazon GameLift changes the player slot to available, which allows it to be assigned to a new player.

Syntax

```
GenericOutcome RemovePlayerSession(String playerSessionId)
```

Parameters

playerSessionId

Unique ID issued by Amazon 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: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
Aws::GameLift::GenericOutcome disconnectOutcome =
    Aws::GameLift::Server::RemovePlayerSession(playerSessionId);
```

StartMatchBackfill()

Sends a request to find new players for open slots in a game session created with FlexMatch. See also the AWS SDK action StartMatchBackfill(). With this action, match backfill requests can be initiated by a game server process that is hosting the game session. Learn more about the FlexMatch backfill feature.

This action is asynchronous. If new players are successfully matched, the Amazon GameLift service delivers updated matchmaker data using the callback function `OnUpdateGameSession()`.

A server process can have only one active match backfill request at a time. To send a new request, first call StopMatchBackfill() to cancel the original request.

Syntax

```
StartMatchBackfillOutcome StartMatchBackfill (StartMatchBackfillRequest
 startBackfillRequest);
```

Parameters

StartMatchBackfillRequest

A StartMatchBackfillRequest object that communicates the following information:

- A ticket ID to assign to the backfill request. This information is optional; if no ID is provided, Amazon GameLift will autogenerate one.
- The matchmaker to send the request to. The full configuration ARN is required. This value can be acquired from the game session's matchmaker data.
- The ID of the game session that is being backfilled.
- Available matchmaking data for the game session's current players.

Required: Yes

Return value

Returns a StartMatchBackfillOutcome object with the match backfill ticket ID or failure with an error message.

Example

```
// Build a backfill request
var startBackfillRequest = new AWS.GameLift.Server.Model.StartMatchBackfillRequest()
{
    TicketId = "a ticket ID", //optional
    MatchmakingConfigurationArn = "the matchmaker configuration ARN",
    GameSessionId = GameLiftServerAPI.GetGameSessionId().Result,    // gets ID for
 current game session
        //get player data for all currently connected players
            MatchmakerData matchmakerData =
              MatchmakerData.FromJson(gameSession.MatchmakerData);  // gets matchmaker
 data for current players
            // get matchmakerData.Players
            // remove data for players who are no longer connected
    Players = ListOfPlayersRemainingInTheGame
};

// Send backfill request
var startBackfillOutcome = GameLiftServerAPI.StartMatchBackfill(startBackfillRequest);

// Implement callback function for backfill
void OnUpdateGameSession(GameSession myGameSession)
{
   // game-specific tasks to prepare for the newly matched players and update
 matchmaker data as needed
}
```

StopMatchBackfill()

Cancels an active match backfill request that was created with StartMatchBackfill(). See also the AWS SDK action StopMatchmaking(). Learn more about the FlexMatch backfill feature.

Syntax

```
GenericOutcome StopMatchBackfill (StopMatchBackfillRequest stopBackfillRequest);
```

Parameters

StopMatchBackfillRequest

A StopMatchBackfillRequest object identifying the matchmaking ticket to cancel:

- ticket ID assigned to the backfill request being canceled
- matchmaker the backfill request was sent to
- game session associated with the backfill request

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
// Set backfill stop request parameters

var stopBackfillRequest = new AWS.GameLift.Server.Model.StopMatchBackfillRequest()
{
    TicketId = "a ticket ID", //optional, if not provided one is autogenerated
    MatchmakingConfigurationArn = "the matchmaker configuration ARN", //from the game
 session matchmaker data
    GameSessionId = GameLiftServerAPI.GetGameSessionId().Result    //gets the ID for
 the current game session
};

var stopBackfillOutcome =
 GameLiftServerAPI.StopMatchBackfillRequest(stopBackfillRequest);
```

TerminateGameSession()

This method is deprecated with version 4.0.1. Instead, the server process should call ProcessEnding() after a game session has ended.

Notifies the Amazon GameLift service that the server process has ended the current game session. This action is called when the server process will remain active and ready to host a new game session. It should be called only after your game session termination procedure is complete,

because it signals to Amazon GameLift that the server process is immediately available to host a new game session.

This action is not called if the server process will be shut down after the game session stops. Instead, call ProcessEnding() to signal that both the game session and the server process are ending.

Syntax

```
GenericOutcome TerminateGameSession()
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example illustrates a server process at the end of a game session.

```
// game-specific tasks required to gracefully shut down a game session,
// such as notifying players, preserving game state data, and other cleanup

var terminateGameSessionOutcome = GameLiftServerAPI.TerminateGameSession();
var processReadyOutcome = GameLiftServerAPI.ProcessReady(processParams);
```

UpdatePlayerSessionCreationPolicy()

Updates the current game session's ability to accept new player sessions. A game session can be set to either accept or deny all new player sessions. (See also the UpdateGameSession() action in the *Amazon GameLift Service API Reference*).

Syntax

```
GenericOutcome UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy
 playerSessionPolicy)
```

Parameters

newPlayerSessionPolicy

String value indicating whether the game session accepts new players.

Type: PlayerSessionCreationPolicy enum. Valid values include:

- **ACCEPT_ALL** – Accept all new player sessions.
- **DENY_ALL** – Deny all new player sessions.

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example sets the current game session's join policy to accept all players.

```
var updatePlayerSessionCreationPolicyOutcomex =

 GameLiftServerAPI.UpdatePlayerSessionCreationPolicy(PlayerSessionCreationPolicy.ACCEPT_ALL);
```

Amazon GameLift server SDK (C#) reference: Data types

You can use this Amazon GameLift C# server SDK reference to help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server.

- Actions
- Data types

LogParameters

This data type is used to identify which files generated during a game session that you want Amazon GameLift to upload and store once the game session ends. This information is communicated to the Amazon GameLift service in a ProcessReady() call.

Contents

logPaths

List of directory paths to game server log files you want Amazon GameLift to store for future access. These files are generated by a server process during each game session; file paths and names are defined in your game server and stored in the root game build directory. The log paths must be absolute. For example, if your game build stores game session logs in a path like `MyGame\sessionlogs\`, then the log path would be `c:\game\MyGame\sessionLogs` (on a Windows instance) or `/local/game/MyGame/sessionLogs` (on a Linux instance).

Type: List<String>

Required: No

DescribePlayerSessionsRequest

This data type is used to specify which player session(s) to retrieve. It can be used in several ways: (1) provide a PlayerSessionId to request a specific player session; (2) provide a GameSessionId to request all player sessions in the specified game session; or (3) provide a PlayerId to request all player sessions for the specified player. For large collections of player sessions, use the pagination parameters to retrieve results as sequential pages.

Contents

GameSessionId

Unique game session identifier. Use this parameter to request all player sessions for the specified game session. Game session ID format is as follows: `arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>`. The value of <ID string> is either a custom ID string (if one was specified when the game session was created) a generated string.

Type: String

Required: No

Limit

Maximum number of results to return. Use this parameter with *NextToken* to get results as a set of sequential pages. If a player session ID is specified, this parameter is ignored.

Type: Integer

Required: No

NextToken

Token indicating the start of the next sequential page of results. Use the token that is returned with a previous call to this action. To specify the start of the result set, do not specify a value. If a player session ID is specified, this parameter is ignored.

Type: String

Required: No

PlayerId

Unique identifier for a player. Player IDs are defined by the developer. See Generate player IDs.

Type: String

Required: No

PlayerSessionId

Unique identifier for a player session.

Type: String

Required: No

PlayerSessionStatusFilter

Player session status to filter results on. Possible player session statuses include the following:

- RESERVED – The player session request has been received, but the player has not yet connected to the server process and/or been validated.
- ACTIVE – The player has been validated by the server process and is currently connected.
- COMPLETED – The player connection has been dropped.
- TIMEDOUT – A player session request was received, but the player did not connect and/or was not validated within the time-out limit (60 seconds).

Type: String

Required: No

ProcessParameters

This data type contains the set of parameters sent to the Amazon GameLift service in a ProcessReady() call.

Contents

port

Port number the server process will listen on for new player connections. The value must fall into the port range configured for any fleet deploying this game server build. This port number is included in game session and player session objects, which game sessions use when connecting to a server process.

Type: Integer

Required: Yes

logParameters

Object with a list of directory paths to game session log files.

Type: `Aws::GameLift::Server::LogParameters`

Required: Yes

onStartGameSession

Name of callback function that the Amazon GameLift service invokes to activate a new game session. Amazon GameLift calls this function in response to the client request CreateGameSession. The callback function takes a GameSession object (defined in the *Amazon GameLift Service API Reference*).

Type: `void OnStartGameSessionDelegate(GameSession gameSession)`

Required: Yes

onProcessTerminate

Name of callback function that the Amazon GameLift service invokes to force the server process to shut down. After calling this function, Amazon GameLift waits five minutes for the server process to shut down and respond with a ProcessEnding() call before it shuts down the server process.

Type: `void OnProcessTerminateDelegate()`

Required: Yes

onHealthCheck

Name of callback function that the Amazon GameLift service invokes to request a health status report from the server process. Amazon GameLift calls this function every 60 seconds. After calling this function Amazon GameLift waits 60 seconds for a response, and if none is received. records the server process as unhealthy.

Type: `bool OnHealthCheckDelegate()`

Required: Yes

onUpdateGameSession

Name of callback function that the Amazon GameLift service invokes to pass an updated game session object to the server process. Amazon 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 Amazon GameLift service in a StartMatchBackfill() call.

Contents

GameSessionArn

Unique game session identifier. The SDK method GetGameSessionId() returns the identifier in ARN format.

Type: String

Required: Yes

MatchmakingConfigurationArn

Unique identifier, in the form of an ARN, for the matchmaker to use for this request. To find the matchmaker that was used to create the original game session, look in the game session object, in the matchmaker data property. Learn more about matchmaker data in Work with matchmaker data.

Type: String

Required: Yes

Players

A set of data representing all players who are currently in the game session. The matchmaker uses this information to search for new players who are good matches for the current players. See the *Amazon GameLift API Reference Guide* for a description of the Player object format. To find player attributes, IDs, and team assignments, look in the game session object, in the matchmaker data property. If latency is used by the matchmaker, gather updated latency for the current region and include it in each player's data.

Type: Player[ ]

Required: Yes

TicketId

Unique identifier for a matchmaking or match backfill request ticket. If no value is provided here, Amazon GameLift will generate one in the form of a UUID. Use this identifier to track the match backfill ticket status or cancel the request if needed.

Type: String

Required: No

StopMatchBackfillRequest

This data type is used to cancel a matchmaking backfill request. The information is communicated to the Amazon GameLift service in a StopMatchBackfill() 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

Amazon GameLift server SDK reference for Go

You can use this Amazon GameLift Go server SDK reference to help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server.

Topics

- Amazon GameLift server SDK (Go) reference: Actions
- Amazon GameLift server SDK (Go) reference: Data types

Amazon GameLift server SDK (Go) reference: Actions

You can use this Amazon GameLift Go server SDK reference to help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server.

`GameLiftServerAPI.go` defines the Go server SDK actions.

Actions

- GetSdkVersion()

- InitSDK()
- ProcessReady()
- ProcessEnding()
- ActivateGameSession()
- UpdatePlayerSessionCreationPolicy()
- GetGameSessionId()
- GetTerminationTime()
- AcceptPlayerSession()
- RemovePlayerSession()
- DescribePlayerSessions()
- StartMatchBackfill()
- StopMatchBackfill()
- GetComputeCertificate()
- GetFleetRoleCredentials()
- Destroy()

GetSdkVersion()

Returns the current version number of the SDK built into the server process.

Syntax

```
func GetSdkVersion() (string, error)
```

Return value

If successful, returns the current SDK version as a string. The returned string includes the version number (example `5.0.0`). If not successful, returns an error message such as `common.SdkVersionDetectionFailed`.

Example

```
version, err := server.GetSdkVersion()
```

InitSDK()

Initializes the Amazon GameLift SDK. Call this method on launch before any other initialization related to Amazon GameLift occurs. This method sets up communication between the server and the Amazon GameLift service.

Syntax

```
func InitSDK(params ServerParameters) error
```

Parameters

ServerParameters

To initialize a game server on an Amazon GameLift Anywhere fleet, construct a `ServerParameters` object with the following information:

- The URL of the WebSocket used to connect to your game server.
- The ID of the process used to host your game server.
- The ID of the compute hosting your game server processes.
- The ID of the Amazon GameLift fleet containing your Amazon GameLift Anywhere compute.
- The authorization token generated by the Amazon GameLift operation.

To initialize a game server on an Amazon GameLift managed EC2 fleet, construct a `ServerParameters` object with no parameters. With this call, the Amazon GameLift agent sets up the compute environment and automatically connects to the Amazon GameLift service for you.

Return value

If successful, returns `nil` error to indicate that the server process is ready to call ProcessReady().

> **Note**
>
> If calls to `InitSDK()` are failing for game builds deployed to Anywhere fleets, check the `ServerSdkVersion` parameter used when creating the build resource. You must explicitly set this value to the server SDK version in use. The default value for this parameter is 4.x, which is not compatible. To resolve this issue, create a new build and deploy it to a new fleet.

Example

Amazon GameLift Anywhere example

```
//Define the server parameters
serverParameters := ServerParameters {
  WebSocketURL: "wss://us-west-1.api.amazongamelift.com",
  ProcessID: "PID1234",
  HostID: "HardwareAnywhere",
  FleetID: "aarn:aws:gamelift:us-west-1:111122223333:fleet/
fleet-9999ffff-88ee-77dd-66cc-5555bbbb44aa",
  AuthToken: "1111aaaa-22bb-33cc-44dd-5555eeee66ff"
}

//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
err := server.InitSDK(serverParameters)
```

Amazon GameLift managed EC2 example

```
//Define the server parameters
serverParameters := ServerParameters {}

//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
err := server.InitSDK(serverParameters)
```

ProcessReady()

Notifies Amazon GameLift that the server process is ready to host game sessions. Call this method after invoking InitSDK(). This method should be called only one time per process.

Syntax

```
func ProcessReady(param ProcessParameters) error
```

Parameters

ProcessParameters

A ProcessParameters object communicates the following information about the server process:

- The names of the callback methods implemented in the game server code that the Amazon GameLift service invokes to communicate with the server process.
- The port number that the server process is listening on.
- The LogParameters data type containing the path to any game session-specific files that you want Amazon GameLift to capture and store.

Return value

Returns an error with an error message if the method fails. Returns `nil` if the method is successful.

Example

This example illustrates both the ProcessReady() call and delegate function implementations.

```
// Define the process parameters
processParams := ProcessParameters {
  OnStartGameSession: gameProcess.OnStartGameSession,
  OnUpdateGameSession: gameProcess.OnGameSessionUpdate,
  OnProcessTerminate: gameProcess.OnProcessTerminate,
  OnHealthCheck: gameProcess.OnHealthCheck,
  Port: port,
  LogParameters: LogParameters {    // logging and error example
    []string {"C:\\game\\logs", "C:\\game\\error"}
  }
}

err := server.ProcessReady(processParams)
```

ProcessEnding()

Notifies Amazon GameLift that the server process is terminating. Call this method after all other cleanup tasks (including shutting down the active game session) and before terminating the process. Depending on the result of `ProcessEnding()`, the process exits with success (0) or error (-1) and generates a fleet event. If the process terminates with an error, the fleet event generated is `SERVER_PROCESS_TERMINATED_UNHEALTHY`.

Syntax

```
func ProcessEnding() error
```

Return value

Returns a 0 error code or a defined error code.

Example

```
// operations to end game sessions and the server process
defer func() {
  err := server.ProcessEnding()
  server.Destroy()
  if err != nil {
    fmt.Println("ProcessEnding() failed. Error: ", err)
    os.Exit(-1)
  } else {
    os.Exit(0)
  }
}
```

ActivateGameSession()

Notifies Amazon GameLift that the server process has activated a game session and is now ready to receive player connections. This action is called as part of the onStartGameSession() callback function, after all game session initialization.

Syntax

```
func ActivateGameSession() error
```

Return value

Returns an error with an error message if the method fails.

Example

This example shows ActivateGameSession() called as part of the onStartGameSession() delegate function.

```
func OnStartGameSession(GameSession gameSession) {
  // game-specific tasks when starting a new game session, such as loading map
  // Activate when ready to receive players
  err := server.ActivateGameSession();
}
```

UpdatePlayerSessionCreationPolicy()

Updates the current game session's ability to accept new player sessions. A game session can be set to either accept or deny all new player sessions.

Syntax

```
func UpdatePlayerSessionCreationPolicy(policy model.PlayerSessionCreationPolicy) error
```

Parameters

playerSessionCreationPolicy

String value that indicates whether the game session accepts new players.

Valid values include:

- **`model.AcceptAll`** – Accept all new player sessions.
- **`model.DenyAll`** – Deny all new player sessions.

Return value

Returns an error with an error message if failure occurs.

Example

This example sets the current game session's join policy to accept all players.

```
err := server.UpdatePlayerSessionCreationPolicy(model.AcceptAll)
```

GetGameSessionId()

Retrieves the ID of the game session hosted by the active server process.

Syntax

```
func GetGameSessionID() (string, error)
```

Parameters

This action has no parameters.

Return value

If successful, returns the game session ID and nil error. For idle processes that aren't yet activated with a game session, the call returns an empty string and `nil` error.

Example

```
gameSessionID, err := server.GetGameSessionID()
```

GetTerminationTime()

Returns the time that a server process is scheduled to be shut down if a termination time is available. A server process takes this action after receiving an on`ProcessTerminate()` callback from Amazon GameLift. Amazon GameLift calls on`ProcessTerminate()` for the following reasons:

- When the server process has reported poor health or hasn't responded to Amazon GameLift.
- When terminating the instance during a scale-down event.
- When an instance is terminated due to a spot-instance interruption.

Syntax

```
func GetTerminationTime() (int64, error)
```

Return value

If successful, returns the timestamp in epoch seconds that the server process is scheduled to shut down and a `nil` error termination. The value is the termination time, expressed in elapsed ticks from `0001 00:00:00`. For example, the date time value `2020-09-13 12:26:40 -000Z` is equal to `637355968000000000` ticks. If no termination time is available, returns an error message.

Example

```
terminationTime, err := server.GetTerminationTime()
```

AcceptPlayerSession()

Notifies Amazon GameLift that a player with the specified player session ID has connected to the server process and needs validation. Amazon GameLift verifies that the player session ID is valid.

After the player session is validated, Amazon GameLift changes the status of the player slot from RESERVED to ACTIVE.

Syntax

```
func AcceptPlayerSession(playerSessionID string) error
```

Parameters

`playerSessionId`

Unique ID issued by Amazon GameLift when a new player session is created.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example handles a connection request that includes validating and rejecting non-valid player session IDs.

```
func ReceiveConnectingPlayerSessionID(conn Connection, playerSessionID string) {
    err := server.AcceptPlayerSession(playerSessionID)
    if err != nil {
        connection.Accept()
    } else {
        connection.Reject(err.Error())
    }
}
```

RemovePlayerSession()

Notifies Amazon GameLift that a player has disconnected from the server process. In response, Amazon GameLift changes the player slot to available.

Syntax

```
func RemovePlayerSession(playerSessionID string) error
```

Parameters

`playerSessionId`

Unique ID issued by Amazon GameLift when a new player session is created.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
err := server.RemovePlayerSession(playerSessionID)
```

DescribePlayerSessions()

Retrieves player session data which includes settings, session metadata, and player data. Use this method to get information about the following:

- A single player session
- All player sessions in a game session
- All player sessions associated with a single player ID

Syntax

```
func DescribePlayerSessions(req request.DescribePlayerSessionsRequest)
 (result.DescribePlayerSessionsResult, error) {
 return srv.describePlayerSessions(&req)
}
```

Parameters

DescribePlayerSessionsRequest

A `DescribePlayerSessionsRequest` object describes which player sessions to retrieve.

Return value

If successful, returns a `DescribePlayerSessionsResult` object that contains a set of player session objects that fit the request parameters.

Example

This example requests all player sessions actively connected to a specified game session. By omitting *NextToken* and setting the *Limit* value to 10, Amazon GameLift returns the first 10 player session records matching the request.

```
// create request
describePlayerSessionsRequest := request.NewDescribePlayerSessions()
describePlayerSessionsRequest.GameSessionID, _ = server.GetGameSessionID() // get ID
 for the current game session
describePlayerSessionsRequest.Limit = 10                                  // return the
 first 10 player sessions
describePlayerSessionsRequest.PlayerSessionStatusFilter = "ACTIVE"          // Get all
 player sessions actively connected to the game session

describePlayerSessionsResult, err :=
 server.DescribePlayerSessions(describePlayerSessionsRequest)
```

StartMatchBackfill()

Sends a request to find new players for open slots in a game session created with FlexMatch. For more information, see FlexMatch backfill feature.

This action is asynchronous. If new players are matched, Amazon GameLift delivers updated matchmaker data using the callback function `OnUpdateGameSession()`.

A server process can have only one active match backfill request at a time. To send a new request, first call StopMatchBackfill() to cancel the original request.

Syntax

```
func StartMatchBackfill(req request.StartMatchBackfillRequest)
 (result.StartMatchBackfillResult, error)
```

Parameters

StartMatchBackfillRequest

A StartMatchBackfillRequest object communicates the following information:

- A ticket ID to assign to the backfill request. This information is optional; if no ID is provided, Amazon GameLift generates one.

- The matchmaker to send the request to. The full configuration ARN is required. This value is in the game session's matchmaker data.
- The ID of the game session to backfill.
- The available matchmaking data for the game session's current players.

Return value

Returns a `StartMatchBackfillResult` object with the match backfill ticket ID, or failure with an error message.

Example

```
// form the request
startBackfillRequest := request.NewStartMatchBackfill()
startBackfillRequest.RequestID = "1111aaaa-22bb-33cc-44dd-5555eeee66ff"          //
 optional
startBackfillRequest.MatchmakingConfigurationArn = "arn:aws:gamelift:us-
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig"
var matchMaker model.MatchmakerData
if err := matchMaker.UnmarshalJSON([]byte(gameSession.MatchmakerData)); err != nil {

    return
}
startBackfillRequest.Players = matchMaker.Players
res, err := server.StartMatchBackfill(startBackfillRequest)

// Implement callback function for backfill
func OnUpdateGameSession(myGameSession model.GameSession) {
    // game-specific tasks to prepare for the newly matched players and update
 matchmaker data as needed
}
```

StopMatchBackfill()

Cancels an active match backfill request. For more information, see FlexMatch backfill feature.

Syntax

```
func StopMatchBackfill(req request.StopMatchBackfillRequest) error
```

Parameters

StopMatchBackfillRequest

A StopMatchBackfillRequest object that identifies the matchmaking ticket to cancel:

- The ticket ID assigned to the backfill request.
- The matchmaker the backfill request was sent to.
- The game session associated with the backfill request.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
stopBackfillRequest := request.NewStopMatchBackfill()  // Use this function to create
 request
stopBackfillRequest.TicketID = "1111aaaa-22bb-33cc-44dd-5555eeee66ff"
stopBackfillRequest.MatchmakingConfigurationArn = "arn:aws:gamelift:us-
west-2:111122223333:matchmakingconfiguration/MyMatchmakerConfig"

//error
err := server.StopMatchBackfill(stopBackfillRequest)
```

GetComputeCertificate()

Retrieves the path to the TLS certificate used to encrypt the network connection between the game server and your game client. You can use the certificate path when you register your compute device to a Amazon GameLift Anywhere fleet. For more information, see RegisterCompute.

Syntax

```
func GetComputeCertificate() (result.GetComputeCertificateResult, error)
```

Return value

Returns a `GetComputeCertificateResult` object that contains the following:

- CertificatePath: The path to the TLS certificate on your compute resource. When using an Amazon GameLift managed fleet, this path contains:
  - `certificate.pem`: The end-user certificate. The full certificate chain is the combination of `certificateChain.pem` appended to this certificate.
  - `certificateChain.pem`: The certificate chain that contains the root certificate and intermediate certificates.
  - `rootCertificate.pem`: The root certificate.
  - `privateKey.pem`: The private key for the end-user certificate.
- ComputeName: The name of your compute resource.

Example

```
tlsCertificate, err := server.GetFleetRoleCredentials(getFleetRoleCredentialsRequest)
```

GetFleetRoleCredentials()

Retrieves the service role credentials that you create to extend permissions to your other AWS services to Amazon GameLift. These credentials allow your game server to use your AWS resources. For more information, see Set up an IAM service role for Amazon GameLift.

Syntax

```
func GetFleetRoleCredentials(
  req request.GetFleetRoleCredentialsRequest,
) (result.GetFleetRoleCredentialsResult, error) {
  return srv.getFleetRoleCredentials(&req)
}
```

Parameters

GetFleetRoleCredentialsRequest

Role credentials that extend limited access to your AWS resources to the game server.

Return value

Returns a `GetFleetRoleCredentialsResult` object that contains the following:

- AssumedRoleUserArn - The Amazon Resource Name (ARN) of the user that the service role belongs to.
- AssumedRoleId - The ID of the user that the service role belongs to.
- AccessKeyId - The access key ID to authenticate and provide access to your AWS resources.
- SecretAccessKey - The secret access key ID for authentication.
- SessionToken - A token to identify the current active session interacting with your AWS resources.
- Expiration - The amount of time until your session credentials expire.

Example

```
// form the customer credentials request
getFleetRoleCredentialsRequest := request.NewGetFleetRoleCredentials()
getFleetRoleCredentialsRequest.RoleArn = "arn:aws:iam::123456789012:role/service-role/
exampleGameLiftAction"

credentials, err := server.GetFleetRoleCredentials(getFleetRoleCredentialsRequest)
```

Destroy()

Frees the Amazon GameLift game server SDK from memory. As a best practice, call this method after `ProcessEnding()` and before terminating the process. If you're using an Anywhere fleet and you're not terminating server processes after every game session, call `Destroy()` and then `InitSDK()` to reinitialize before notifying Amazon GameLift that the process is ready to host a game session with `ProcessReady()`.

Syntax

```
func Destroy() error {
 return srv.destroy()
}
```

Return value

Returns an error with an error message if the method fails.

Example

```
// operations to end game sessions and the server process
```

```
defer func() {
  err := server.ProcessEnding()
  server.Destroy()
  if err != nil {
    fmt.Println("ProcessEnding() failed. Error: ", err)
    os.Exit(-1)
  } else {
    os.Exit(0)
  }
}
```

Amazon GameLift server SDK (Go) reference: Data types

You can use this Amazon GameLift Go server SDK reference to help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server.

Data types

- LogParameters
- ProcessParameters
- UpdateGameSession
- GameSession
- ServerParameters
- StartMatchBackfillRequest
- Player
- DescribePlayerSessionsRequest
- StopMatchBackfillRequest
- GetFleetRoleCredentialsRequest

LogParameters

An object identifying files generated during a game session that you want Amazon GameLift to upload and store after the game session ends. The game server provides `LogParameters` to Amazon GameLift as part of a `ProcessParameters` object in a ProcessReady() call.

Properties	Description

`LogPaths`	The list of directory paths to game server log files that you want Amazon GameLift to store for future access. The server process generates these files during each game session. You define file paths and names in your game server and store them in the root game build directory. The log paths must be absolute. For example, if your game build stores game session logs in a path such as `MyGame\sessionLogs\` , then the path would be `c:\game\MyGame\sessionLogs` on a Windows instance. **Type:** `[]string` **Required:** No

ProcessParameters

An object describing the communication between a server process and Amazon GameLift. The server process provides this information to Amazon GameLift with a call to ProcessReady().

Properties	Description
`LogParameters`	An object with directory paths to files that are generated during a game session. Amazon GameLift copies and stores the files for future access. **Type:** `LogParameters` **Required:** No
`OnHealthCheck`	The callback function that Amazon GameLift invokes to request a health status report from the server process. Amazon GameLift calls this function every 60 seconds and waits 60 seconds for a response. The server process returns `TRUE` if healthy, `FALSE` if not healthy. If no response is returned, Amazon GameLift records the server process as not healthy. **Type:** `OnHealthCheck func() bool` **Required:** No
`OnProcessTerminate`	The callback function that Amazon GameLift invokes to force the server process to shut down. After calling this function, Amazon GameLift waits 5

	minutes for the server process to shut down and respond with a ProcessEnding() call before it shuts down the server process. **Type:** `OnProcessTerminate func()` **Required:** Yes
`OnStartGameSession`	The callback function that Amazon GameLift invokes to pass an updated game session object to the server process. Amazon GameLift calls this function when a match backfill request has been processed to provide updated matchmaker data. It passes a GameSession object, a status update (`updateReason`), and the match backfill ticket ID. **Type:** `OnStartGameSession func (model.GameSession )` **Required:** Yes
`OnUpdateGameSession`	The callback function that Amazon GameLift invokes to pass updated game session information to the server process. Amazon GameLift calls this function after processing a match backfill request to provide updated matchmaker data. **Type:** `OnUpdateGameSession func (model.UpdateGameSession)` **Required:** No
`Port`	The port number that the server process listens on for new player connections. The value must fall into the port range configured for any fleet deploying this game server build. This port number is included in game session and player session objects, which game sessions use when connecting to a server process. **Type:** `int` **Required:** Yes

UpdateGameSession

The updates to a game session object, which includes the reason that the game session was updated, and the related backfill ticket ID if backfill is being used to fill player sessions in the game session.

Properties	Description
GameSession	A GameSession object defined by the Amazon GameLift API. The `GameSession` object contains properties describing a game session. **Type:** `GameSession GameSession()` **Required:** Yes
UpdateReason	The reason that the game session is being updated. **Type:** `UpdateReason UpdateReason()` **Required:** Yes
BackfillTicketId	The ID of the backfill ticket attempting to update the game session. **Type:** `String` **Required:** No

GameSession

The details of a game session.

Properties	Description
GameSessionId	A unique identifier for the game session. A game session Amazon Resource Name (ARN) has the following format: `arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token>` . **Type:** `String`

Properties	Description
	Required: No
Name	A descriptive label of the game session. **Type:** `String` **Required**: No
FleetId	A unique identifier for the fleet that the game session is running on. **Type:** `String` **Required**: No
MaximumPlayerSessionCount	The maximum number of player connections to the game session. **Type:** `Integer` **Required**: No
Port	The port number for the game session. To connect to a Amazon GameLift game server, an app needs both the IP address and port number. **Type:** `Integer` **Required**: No
IpAddress	The IP address of the game session. To connect to a Amazon GameLift game server, an app needs both the IP address and port number. **Type:** `String` **Required**: No
GameSessionData	A set of custom game session properties, formatted as a single string value. **Type:** `String` **Required**: No

Properties	Description
MatchmakerData	The information about the matchmaking process that was used to create the game session, in JSON syntax, formatted as a string. In addition to the matchmaking configuration used, it contains data on all players assigned to the match, including player attributes and team assignments. **Type:** `String` **Required**: No
GameProperties	A set of custom properties for a game session, formatted as key:value pairs. These properties are passed with a request to start a new game session. **Type:** `map[string] string` **Required**: No
DnsName	The DNS identifier assigned to the instance that's running the game session. Values have the following format: • TLS-enabled fleets: `<unique identifier>.<region identifier>.amazongamelift.com` . • Non-TLS-enabled fleets: `ec2-<unique identifier>.compute.amazonaws.com` . When connecting to a game session that's running on a TLS-enabled fleet, you must use the DNS name, not the IP address. **Type:** `String` **Required**: No

ServerParameters

Information used to maintain the connection between an Amazon GameLift Anywhere server and the Amazon GameLift service. This information is used when launching new server processes with InitSDK(). For servers hosted on Amazon GameLift managed EC2 instances, use an empty object.

Properties	Description
`WebSocketURL`	The `GameLiftServerSdkEndpoint` Amazon GameLift returns when you `RegisterCompute` for an Amazon GameLift Anywhere compute resource. **Type:** `string` **Required**: Yes
`ProcessID`	A unique identifier registered to the server process hosting your game. **Type:** `string` **Required**: Yes
`HostID`	The unique identifier of the compute resource that's hosting the new server process. The `HostID` is the `ComputeName` used when you registered your compute. For more information, see RegisterCompute. **Type:** `string` **Required**: Yes
`FleetID`	The unique identifier of the fleet that the compute is registered to. For more information, see RegisterCompute. **Type:** `string` **Required**: Yes
`AuthToken`	The authentication token generated by Amazon GameLift that authentic ates your server to Amazon GameLift. For more information, see GetComput eAuthToken. **Type:** `string` **Required**: Yes

StartMatchBackfillRequest

Information used to create a matchmaking backfill request. The game server communicates this information to Amazon GameLift in a StartMatchBackfill() call.

Properties	Description
GameSessionArn	The unique game session identifier. The API operation `GetGameSessionId` returns the identifier in ARN format. **Type:** `String` **Required**: Yes
MatchmakingConfigurationArn	The unique identifier (in the form of an ARN) for the matchmaker to use for this request. The matchmaker ARN for the original game session is in the game session object in the matchmaker data property. For more information about matchmaker data, see Work with matchmaker data. **Type:** `String` **Required**: Yes
Players	A set of data that represents all players who are currently in the game session. The matchmaker uses this information to search for new players who are good matches for the current players. **Type:** `[]model.Player` **Required**: Yes
TicketId	The unique identifier for a matchmaking or match backfill request ticket. If you don't provide a value, Amazon GameLift generates one. Use this identifier to track the match backfill ticket status or cancel the request if needed. **Type:** `String` **Required**: No

Player

The object that represents a player in matchmaking. When a matchmaking request starts, a player has a player ID, attributes, and possibly latency data. Amazon GameLift adds team information after a match is made.

Properties	Description
LatencyInMS	A set of values expressed in milliseconds that indicate the amount of latency that a player experiences when connected to a location. If this property is used, the player is only matched for locations listed. If a matchmaker has a rule that evaluates player latency, players must report latency to be matched. **Type:** `map[string] int` **Required**: No
PlayerAttributes	A collection of key:value pairs that contain player information for use in matchmaking. Player attribute keys must match the PlayerAttributes used in a matchmaking rule set. For more information about player attributes, see AttributeValue. **Type:** `map[string] AttributeValue` **Required**: No
PlayerId	A unique identifier for a player. **Type:** `String` **Required**: No
Team	The name of the team that the player is assigned to in a match. You define the team name in the matchmaking rule set. **Type:** `String` **Required**: No

DescribePlayerSessionsRequest

An object that specifies which player sessions to retrieve. The server process provides this information with a DescribePlayerSessions() call to Amazon GameLift.

Properties	Description
`GameSessionID`	A unique game session identifier. Use this parameter to request all player sessions for the specified game session. Game session ID format is `arn:aws:gamelift:<region>::gamesession/fleet-<fleet ID>/<ID string>` . The `GameSessionID` is a custom ID string or a generated string. **Type:** `String` **Required**: No
`PlayerSessionID`	The unique identifier for a player session. Use this parameter to request a single specific player session. **Type:** `String` **Required**: No
`PlayerID`	The unique identifier for a player. Use this parameter to request all player sessions for a specific player. See Generate player IDs. **Type:** `String` **Required**: No
`PlayerSessionStatusFilter`	The player session status to filter results on. Possible player session statuses include: • RESERVED – The player session request was received, but the player hasn't connected to the server process or been validated. • ACTIVE – The player was validated by the server process and is connected. • COMPLETED – The player connection dropped.

Properties	Description
	• TIMEDOUT – A player session request was received, but the player didn't connect or wasn't validated within the time-out limit (60 seconds). **Type:** `String` **Required**: No
`NextToken`	The token indicating the start of the next page of results. To specify the start of the result set, don't provide a value. If you provide a player session ID, this parameter is ignored. **Type:** `String` **Required**: No
`Limit`	The maximum number of results to return. If you provide a player session ID, this parameter is ignored. **Type:** `int` **Required**: No

StopMatchBackfillRequest

Information used to cancel a matchmaking backfill request. The game server communicates this information to Amazon GameLift service in a StopMatchBackfill() call.

Properties	Description
GameSessionArn	The unique game session identifier of the request being canceled. **Type:** `string` **Required**: No
MatchmakingConfigurationArn	The unique identifier of the matchmaker this request was sent to. **Type:** `string`

Properties	Description
	Required: No
TicketId	The unique identifier of the backfill request ticket to be canceled. **Type:** `string` **Required**: No

GetFleetRoleCredentialsRequest

The role credentials that extend limited access to your AWS resources to the game server. For more information see, Set up an IAM service role for Amazon GameLift.

Properties	Description
RoleArn	The ARN of the service role that extends limited access to your AWS resources. **Type:** `string` **Required**: Yes
RoleSessionName	The name of the session that describes the use of the role credentials. **Type:** `string` **Required:** Yes

Amazon GameLift server SDK reference for Unreal Engine

This Amazon GameLift Server SDK reference can help you prepare your Unreal Engine game projects for use with Amazon GameLift. For details on the integration process, see Add Amazon GameLift to your game server.

This API is defined in `GameLiftServerSDK.h` and `GameLiftServerSDKModels.h`.

To set up the Unreal Engine plugin and see code examples Integrate Amazon GameLift into an Unreal Engine project.

Topics

Amazon GameLift Unreal Engine server SDK 5.x reference

You can use this Amazon GameLift Unreal Engine server SDK 5.x reference to help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server, and for information on using the Unreal SDK server plugin, see Integrate Amazon GameLift into an Unreal Engine project.

Topics

Amazon GameLift server SDK (Unreal) 5.x reference: Actions

You can use this Amazon GameLift Unreal server SDK reference to help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server and for information on using the Unreal SDK server plugin, see Integrate Amazon GameLift into an Unreal Engine project.

Actions

- RemovePlayerSession()
- DescribePlayerSessions()
- StartMatchBackfill()
- StopMatchBackfill()
- GetComputeCertificate()
- GetFleetRoleCredentials()

> **Note**
>
> This topic describes the Amazon GameLift C++ API that you can use when you build for the Unreal Engine. Specifically, this documentation applies to code that you compile with the `-DBUILD_FOR_UNREAL=1` option.

GetSdkVersion()

Returns the current version number of the SDK built into the server process.

Syntax

```
FGameLiftStringOutcome GetSdkVersion();
```

Return value

If successful, returns the current SDK version as an the section called "FGameLiftStringOutcome" object. The returned object includes the version number (example `5.0.0`). If not successful, returns an error message.

Example

```
Aws::GameLift::AwsStringOutcome SdkVersionOutcome =
 Aws::GameLift::Server::GetSdkVersion();
```

InitSDK()

Initializes the Amazon GameLift SDK for a managed EC2 fleet. Call this method on launch, before any other initialization related to Amazon GameLift occurs. This method reads server parameters

from the host environment to set up communication between the server and the Amazon GameLift service.

Syntax

```
FGameLiftGenericOutcome InitSDK()
```

Return value

If successful, returns an `InitSdkOutcome` object indicating that the server process is ready to call ProcessReady().

Example

```
//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
FGameLiftGenericOutcome initSdkOutcome = gameLiftSdkModule->InitSDK();
```

InitSDK()

Initializes the Amazon GameLift SDK for an Anywhere fleet. Call this method on launch, before any other initialization related to Amazon GameLift occurs. This method requires explicit server parameters to set up communication between the server and the Amazon GameLift service.

Syntax

```
FGameLiftGenericOutcome InitSDK(serverParameters)
```

Parameters

FServerParameters

To initialize a game server on an Amazon GameLift Anywhere fleet, construct a `ServerParameters` object with the following information:

- The URL of the WebSocket used to connect to your game server.
- The ID of the process used to host your game server.
- The ID of the compute hosting your game server processes.
- The ID of the Amazon GameLift fleet containing your Amazon GameLift Anywhere compute.

- The authorization token generated by the Amazon GameLift operation.

Return value

If successful, returns an `InitSdkOutcome` object indicating that the server process is ready to call ProcessReady().

> **Note**
>
> If calls to `InitSDK()` are failing for game builds deployed to Anywhere fleets, check the `ServerSdkVersion` parameter used when creating the build resource. You must explicitly set this value to the server SDK version in use. The default value for this parameter is 4.x, which is not compatible. To resolve this issue, create a new build and deploy it to a new fleet.

Example

```
//Define the server parameters
FServerParameters serverParameters;
parameters.m_authToken = "1111aaaa-22bb-33cc-44dd-5555eeee66ff";
parameters.m_fleetId = "arn:aws:gamelift:us-west-1:111122223333:fleet/
fleet-9999ffff-88ee-77dd-66cc-5555bbbb44aa";
parameters.m_hostId = "HardwareAnywhere";
parameters.m_processId = "PID1234";
parameters.m_webSocketUrl = "wss://us-west-1.api.amazongamelift.com";

//Call InitSDK to establish a local connection with the GameLift agent to enable
 further communication.
FGameLiftGenericOutcome initSdkOutcome = gameLiftSdkModule->InitSDK(serverParameters);
```

ProcessReady()

Notifies Amazon GameLift that the server process is ready to host game sessions. Call this method after invoking InitSDK(). This method should be called only once per process.

Syntax

```
GenericOutcome ProcessReady(const Aws::GameLift::Server::ProcessParameters
&processParameters);
```

Parameters

processParameters

An FProcessParameters object communicating the following information about the server process:

- Names of callback methods implemented in the game server code that the Amazon GameLift service invokes to communicate with the server process.
- Port number that the server process is listening on.
- Path to any game session-specific files that you want Amazon GameLift to capture and store.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example illustrates both the ProcessReady() call and delegate function implementations.

```
//Calling ProcessReady tells GameLift this game server is ready to receive incoming
 game sessions!
UE_LOG(GameServerLog, Log, TEXT("Calling Process Ready"));
FGameLiftGenericOutcome processReadyOutcome = gameLiftSdkModule-
>ProcessReady(*params);
```

ProcessEnding()

Notifies Amazon GameLift that the server process is terminating. Call this method after all other cleanup tasks (including shutting down the active game session) and before terminating the process. Depending on the result of `ProcessEnding()`, the process exits with success (0) or error (-1) and generates a fleet event. If the process terminates with an error, the fleet event generated is `SERVER_PROCESS_TERMINATED_UNHEALTHY`).

Syntax

```
FGameLiftGenericOutcome ProcessEnding()
```

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
//OnProcessTerminate callback. GameLift will invoke this callback before shutting down
 an instance hosting this game server.
//It gives this game server a chance to save its state, communicate with services,
 etc., before being shut down.
//In this case, we simply tell GameLift we are indeed going to shutdown.
params->OnTerminate.BindLambda([=]() {
  UE_LOG(GameServerLog, Log, TEXT("Game Server Process is terminating"));
  gameLiftSdkModule->ProcessEnding();
});
```

ActivateGameSession()

Notifies Amazon GameLift that the server process has activated a game session and is now ready to receive player connections. This action should be called as part of the `onStartGameSession()` callback function, after all game session initialization.

Syntax

```
FGameLiftGenericOutcome ActivateGameSession()
```

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example shows `ActivateGameSession()` called as part of the `onStartGameSession()` delegate function.

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

```
};
```

UpdatePlayerSessionCreationPolicy()

Updates the current game session's ability to accept new player sessions. A game session can be set to either accept or deny all new player sessions.

Syntax

```
FGameLiftGenericOutcome UpdatePlayerSessionCreationPolicy(EPlayerSessionCreationPolicy
 policy)
```

Parameters

playerCreationSessionPolicy

String value indicating whether the game session accepts new players.

Valid values include:

- **ACCEPT_ALL** – Accept all new player sessions.
- **DENY_ALL** – Deny all new player sessions.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example sets the current game session's join policy to accept all players.

```
FGameLiftGenericOutcome outcome = gameLiftSdkModule-
>UpdatePlayerSessionCreationPolicy(Aws::GameLift::Model::EPlayerSessionCreationPolicy::ACCEPT_A
```

GetGameSessionId()

Retrieves the ID of the game session hosted by the active server process.

For idle processes that aren't activated with a game session, the call returns a the section called “FGameLiftError”.

Syntax

```
FGameLiftStringOutcome GetGameSessionId()
```

Parameters

This action has no parameters.

Return value

If successful, returns the game session ID as an the section called “FGameLiftStringOutcome” object. If not successful, returns an error message."

For idle processes that aren't activated with a game session, the call returns `Success=True` and `GameSessionId=""`.

Example

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

GetTerminationTime()

Returns the time that a server process is scheduled to be shut down, if a termination time is available. A server process takes action after receiving an `onProcessTerminate()` callback from Amazon GameLift. Amazon GameLift calls `onProcessTerminate()` for the following reasons:

- When the server process has reported poor health or has not responded to Amazon GameLift.
- When terminating the instance during a scale-down event.
- When an instance is terminated due to a spot-instance interruption.

Syntax

```
AwsDateTimeOutcome GetTerminationTime()
```

Return value

If successful, returns the termination time as an `AwsDateTimeOutcome` object. The value is the termination time, expressed in elapsed ticks since `0001 00:00:00`. For example, the date time value `2020-09-13 12:26:40 -000Z` is equal to `637355968000000000` ticks. If no termination time is available, returns an error message.

If the process hasn't received a `ProcessParameters.OnProcessTerminate()` callback, an error message is returned. For more information about shutting down a server process, see Respond to a server process shutdown notification.

Example

```
AwsDateTimeOutcome TermTimeOutcome = gameLiftSdkModule->GetTerminationTime();
```

AcceptPlayerSession()

Notifies Amazon GameLift that a player with the specified player session ID has connected to the server process and needs validation. Amazon GameLift verifies that the player session ID is valid. After the player session is validated, Amazon GameLift changes the status of the player slot from RESERVED to ACTIVE.

Syntax

```
FGameLiftGenericOutcome AcceptPlayerSession(const FString& playerSessionId)
```

Parameters

playerSessionId

Unique ID issued by Amazon GameLift when a new player session is created.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

This example handles a connection request that includes validating and rejecting non-valid player session IDs.

```
bool GameLiftManager::AcceptPlayerSession(const FString& playerSessionId, const
 FString& playerId)
{
  #if WITH_GAMELIFT
  UE_LOG(GameServerLog, Log, TEXT("Accepting GameLift PlayerSession: %s . PlayerId:
 %s"), *playerSessionId, *playerId);
  FString gsId = GetCurrentGameSessionId();
  if (gsId.IsEmpty()) {
    UE_LOG(GameServerLog, Log, TEXT("No GameLift GameSessionId. Returning early!"));
    return false;
  }

  if (!gameLiftSdkModule->AcceptPlayerSession(playerSessionId).IsSuccess()) {
    UE_LOG(GameServerLog, Log, TEXT("PlayerSession not Accepted."));
    return false;
  }

  // Add PlayerSession from internal data structures keeping track of connected players
  connectedPlayerSessionIds.Add(playerSessionId);
  idToPlayerSessionMap.Add(playerSessionId, PlayerSession{ playerId,
 playerSessionId });
  return true;
  #else
  return false;
  #endif
}
```

RemovePlayerSession()

Notifies Amazon GameLift that a player has disconnected from the server process. In response, Amazon GameLift changes the player slot to available.

Syntax

```
FGameLiftGenericOutcome RemovePlayerSession(const FString& playerSessionId)
```

Parameters

`playerSessionId`

Unique ID issued by Amazon GameLift when a new player session is created.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
bool GameLiftManager::RemovePlayerSession(const FString& playerSessionId)
{
  #if WITH_GAMELIFT
  UE_LOG(GameServerLog, Log, TEXT("Removing GameLift PlayerSession: %s"),
 *playerSessionId);

  if (!gameLiftSdkModule->RemovePlayerSession(playerSessionId).IsSuccess()) {
    UE_LOG(GameServerLog, Log, TEXT("PlayerSession Removal Failed"));
    return false;
  }

  // Remove PlayerSession from internal data structures that are keeping track of
 connected players
  connectedPlayerSessionIds.Remove(playerSessionId);
  idToPlayerSessionMap.Remove(playerSessionId);

  // end the session if there are no more players connected
  if (connectedPlayerSessionIds.Num() == 0) {
    EndSession();
  }

  return true;
  #else
  return false;
  #endif
}
```

DescribePlayerSessions()

Retrieves player session data which includes settings, session metadata, and player data. Use this method to get information about the following:

- A single player session
- All player sessions in a game session
- All player sessions associated with a single player ID

Syntax

```
FGameLiftDescribePlayerSessionsOutcome DescribePlayerSessions(const
 FGameLiftDescribePlayerSessionsRequest &describePlayerSessionsRequest)
```

Parameters

FGameLiftDescribePlayerSessionsRequest

A the section called "FGameLiftDescribePlayerSessionsRequest" object that describes which player sessions to retrieve.

Return value

If successful, returns a the section called "FGameLiftDescribePlayerSessionsOutcome" object containing a set of player session objects that fit the request parameters.

Example

This example requests all player sessions actively connected to a specified game session. By omitting *NextToken* and setting the *Limit* value to 10, Amazon GameLift returns the first 10 player session records matching the request.

```
void GameLiftManager::DescribePlayerSessions()
{
  #if WITH_GAMELIFT
  FString localPlayerSessions;
  for (auto& psId : connectedPlayerSessionIds)
  {
    PlayerSession ps = idToPlayerSessionMap[psId];
    localPlayerSessions += FString::Printf(TEXT("%s : %s  ; "), *(ps.playerSessionId),
 *(ps.playerId));
  }
  UE_LOG(GameServerLog, Log, TEXT("LocalPlayerSessions: %s"), *localPlayerSessions);
```

```
  UE_LOG(GameServerLog, Log, TEXT("Describing PlayerSessions in this GameSession"));
  FGameLiftDescribePlayerSessionsRequest request;
  request.m_gameSessionId = GetCurrentGameSessionId();

  FGameLiftDescribePlayerSessionsOutcome outcome = gameLiftSdkModule-
>DescribePlayerSessions(request);
  LogDescribePlayerSessionsOutcome(outcome);
  #endif
}
```

StartMatchBackfill()

Sends a request to find new players for open slots in a game session created with FlexMatch. For more information, see FlexMatch backfill feature.

This action is asynchronous. If new players are matched, Amazon GameLift delivers updated matchmaker data using the callback function `OnUpdateGameSession()`.

A server process can have only one active match backfill request at a time. To send a new request, first call StopMatchBackfill() to cancel the original request.

Syntax

```
FGameLiftStringOutcome StartMatchBackfill (FStartMatchBackfillRequest
 &startBackfillRequest);
```

Parameters

FStartMatchBackfillRequest

A StartMatchBackfillRequest object that communicates the following information:

- A ticket ID to assign to the backfill request. This information is optional; if no ID is provided, Amazon GameLift will generate one.
- The matchmaker to send the request to. The full configuration ARN is required. This value is in the game session's matchmaker data.
- The ID of the game session to backfill.
- The available matchmaking data for the game session's current players.

Return value

Returns a `StartMatchBackfillOutcome` object with the match backfill ticket ID, or failure with an error message.

Example

```
FGameLiftStringOutcome FGameLiftServerSDKModule::StartMatchBackfill(const
 FStartMatchBackfillRequest& request)
{
  #if WITH_GAMELIFT
  Aws::GameLift::Server::Model::StartMatchBackfillRequest sdkRequest;
  sdkRequest.SetTicketId(TCHAR_TO_UTF8(*request.m_ticketId));
  sdkRequest.SetGameSessionArn(TCHAR_TO_UTF8(*request.m_gameSessionArn));

 sdkRequest.SetMatchmakingConfigurationArn(TCHAR_TO_UTF8(*request.m_matchmakingConfigurationArn
  for (auto player : request.m_players) {
    Aws::GameLift::Server::Model::Player sdkPlayer;
    sdkPlayer.SetPlayerId(TCHAR_TO_UTF8(*player.m_playerId));
    sdkPlayer.SetTeam(TCHAR_TO_UTF8(*player.m_team));
    for (auto entry : player.m_latencyInMs) {
      sdkPlayer.WithLatencyMs(TCHAR_TO_UTF8(*entry.Key), entry.Value);
    }

    std::map<std::string, Aws::GameLift::Server::Model::AttributeValue>
 sdkAttributeMap;
    for (auto attributeEntry : player.m_playerAttributes) {
      FAttributeValue value = attributeEntry.Value;
      Aws::GameLift::Server::Model::AttributeValue attribute;
      switch (value.m_type) {
        case FAttributeType::STRING:
          attribute =
 Aws::GameLift::Server::Model::AttributeValue(TCHAR_TO_UTF8(*value.m_S));
        break;
        case FAttributeType::DOUBLE:
          attribute = Aws::GameLift::Server::Model::AttributeValue(value.m_N);
        break;
        case FAttributeType::STRING_LIST:
          attribute =
 Aws::GameLift::Server::Model::AttributeValue::ConstructStringList();
          for (auto sl : value.m_SL) {
            attribute.AddString(TCHAR_TO_UTF8(*sl));
          };
        break;
```

```
        case FAttributeType::STRING_DOUBLE_MAP:
          attribute =
 Aws::GameLift::Server::Model::AttributeValue::ConstructStringDoubleMap();
          for (auto sdm : value.m_SDM) {
            attribute.AddStringAndDouble(TCHAR_TO_UTF8(*sdm.Key), sdm.Value);
          };
        break;
      }
      sdkPlayer.WithPlayerAttribute((TCHAR_TO_UTF8(*attributeEntry.Key)), attribute);
    }
    sdkRequest.AddPlayer(sdkPlayer);
  }
  auto outcome = Aws::GameLift::Server::StartMatchBackfill(sdkRequest);
  if (outcome.IsSuccess()) {
    return FGameLiftStringOutcome(outcome.GetResult().GetTicketId());
  }
  else {
    return FGameLiftStringOutcome(FGameLiftError(outcome.GetError()));
  }
  #else
  return FGameLiftStringOutcome("");
  #endif
}
```

StopMatchBackfill()

Cancels an active match backfill request. For more information, see FlexMatch backfill feature.

Syntax

```
FGameLiftGenericOutcome StopMatchBackfill (FStopMatchBackfillRequest
 &stopBackfillRequest);
```

Parameters

FStopMatchBackfillRequest

A StopMatchBackfillRequest object identifying the matchmaking ticket to cancel:

- The ticket ID assigned to the backfill request.
- The matchmaker the backfill request was sent to.
- The game session associated with the backfill request.

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

```
FGameLiftGenericOutcome FGameLiftServerSDKModule::StopMatchBackfill(const
 FStopMatchBackfillRequest& request)
{
  #if WITH_GAMELIFT
  Aws::GameLift::Server::Model::StopMatchBackfillRequest sdkRequest;
  sdkRequest.SetTicketId(TCHAR_TO_UTF8(*request.m_ticketId));
  sdkRequest.SetGameSessionArn(TCHAR_TO_UTF8(*request.m_gameSessionArn));

 sdkRequest.SetMatchmakingConfigurationArn(TCHAR_TO_UTF8(*request.m_matchmakingConfigurationArn
  auto outcome = Aws::GameLift::Server::StopMatchBackfill(sdkRequest);
  if (outcome.IsSuccess()) {
    return FGameLiftGenericOutcome(nullptr);
  }
  else {
    return FGameLiftGenericOutcome(FGameLiftError(outcome.GetError()));
  }
  #else
  return FGameLiftGenericOutcome(nullptr);
  #endif
}
```

GetComputeCertificate()

Retrieves the path to the TLS certificate used to encrypt the network connection between your Amazon GameLift Anywhere compute resource and Amazon GameLift. You can use the certificate path when you register your compute device to a Amazon GameLift Anywhere fleet. For more information see, RegisterCompute.

Syntax

```
FGameLiftGetComputeCertificateOutcome FGameLiftServerSDKModule::GetComputeCertificate()
```

Return value

Returns a `GetComputeCertificateResponse` object containing the following:

- CertificatePath: The path to the TLS certificate on your compute resource.

- HostName: The host name of your compute resource.

Example

```
FGameLiftGetComputeCertificateOutcome FGameLiftServerSDKModule::GetComputeCertificate()
{
  #if WITH_GAMELIFT
  auto outcome = Aws::GameLift::Server::GetComputeCertificate();
  if (outcome.IsSuccess()) {
    auto& outres = outcome.GetResult();
    FGameLiftGetComputeCertificateResult result;
    result.m_certificate_path = UTF8_TO_TCHAR(outres.GetCertificatePath());
    result.m_computeName = UTF8_TO_TCHAR(outres.GetComputeName());
    return FGameLiftGetComputeCertificateOutcome(result);
  }
  else {
    return FGameLiftGetComputeCertificateOutcome(FGameLiftError(outcome.GetError()));
  }
  #else
  return FGameLiftGetComputeCertificateOutcome(FGameLiftGetComputeCertificateResult());
  #endif
}
```

GetFleetRoleCredentials()

Retrieves IAM role credentials that authorize Amazon GameLift to interact with other AWS services. For more information, see Communicate with other AWS resources from your fleets.

Syntax

```
FGameLiftGetFleetRoleCredentialsOutcome
 FGameLiftServerSDKModule::GetFleetRoleCredentials(const
 FGameLiftGetFleetRoleCredentialsRequest &request)
```

Parameters

FGameLiftGetFleetRoleCredentialsRequest

Return value

Returns a the section called “FGameLiftGetFleetRoleCredentialsOutcome” object.

Example

```
FGameLiftGetFleetRoleCredentialsOutcome
 FGameLiftServerSDKModule::GetFleetRoleCredentials(const
 FGameLiftGetFleetRoleCredentialsRequest &request)
{
  #if WITH_GAMELIFT
  Aws::GameLift::Server::Model::GetFleetRoleCredentialsRequest sdkRequest;
  sdkRequest.SetRoleArn(TCHAR_TO_UTF8(*request.m_roleArn));
  sdkRequest.SetRoleSessionName(TCHAR_TO_UTF8(*request.m_roleSessionName));

  auto outcome = Aws::GameLift::Server::GetFleetRoleCredentials(sdkRequest);

  if (outcome.IsSuccess()) {
    auto& outres = outcome.GetResult();
    FGameLiftGetFleetRoleCredentialsResult result;
    result.m_assumedUserRoleArn = UTF8_TO_TCHAR(outres.GetAssumedUserRoleArn());
    result.m_assumedRoleId = UTF8_TO_TCHAR(outres.GetAssumedRoleId());
    result.m_accessKeyId = UTF8_TO_TCHAR(outres.GetAccessKeyId());
    result.m_secretAccessKey = UTF8_TO_TCHAR(outres.GetSecretAccessKey());
    result.m_sessionToken = UTF8_TO_TCHAR(outres.GetSessionToken());
    result.m_expiration = FDateTime::FromUnixTimestamp(outres.GetExpiration());
    return FGameLiftGetFleetRoleCredentialsOutcome(result);
  }
  else {
    return FGameLiftGetFleetRoleCredentialsOutcome(FGameLiftError(outcome.GetError()));
  }
  #else
  return
 FGameLiftGetFleetRoleCredentialsOutcome(FGameLiftGetFleetRoleCredentialsResult());
  #endif
}
```

Amazon GameLift server SDK (Unreal) reference: Data types

You can use this Amazon GameLift Unreal server SDK reference to help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server and for information on using the Unreal SDK server plugin, see Integrate Amazon GameLift into an Unreal Engine project.

Data types

- FProcessParameters

- UpdateGameSession
- GameSession
- FServerParameters
- FStartMatchBackfillRequest
- FPlayer
- FGameLiftDescribePlayerSessionsRequest
- FStopMatchBackfillRequest
- FAttributeValue
- FGameLiftGetFleetRoleCredentialsRequest
- FGameLiftLongOutcome
- FGameLiftStringOutcome
- FGameLiftDescribePlayerSessionsOutcome
- FGameLiftDescribePlayerSessionsResult
- FGenericOutcome
- FGameLiftPlayerSession
- FGameLiftGetComputeCertificateOutcome
- FGameLiftGetComputeCertificateResult
- FGameLiftGetFleetRoleCredentialsOutcome
- FGetFleetRoleCredentialsResult
- FGameLiftError
- Enums

Note

This topic describes the Amazon GameLift C++ API that you can use when you build for the Unreal Engine. Specifically, this documentation applies to code that you compile with the `-DBUILD_FOR_UNREAL=1` option.

FProcessParameters

This data type contains the set of parameters sent to Amazon GameLift in a ProcessReady().

Properties	Description
LogParameters	An object with directory paths to files that are generated during a game session. Amazon GameLift copies and stores the files for future access. **Type:** `TArray<FString>` **Required:** No
OnHealthCheck	The callback function that Amazon GameLift invokes to request a health status report from the server process. Amazon GameLift calls this function every 60 seconds and waits 60 seconds for a response. The server process returns `TRUE` if healthy, `FALSE` if not healthy. If no response is returned, Amazon GameLift records the server process as not healthy. This property is a delegate function defined as `DECLARE_DELEGATE_RetVal(bool, FOnHealthCheck) ;` **Type:** `FOnHealthCheck` **Required:** No
OnProcessTerminate	The callback function that Amazon GameLift invokes to force the server process to shut down. After calling this function, Amazon GameLift waits 5 minutes for the server process to shut down and respond with a ProcessEnding() call before it shuts down the server process. **Type:** `FSimpleDelegate` **Required:** Yes

OnStartGameSession	The callback function that Amazon GameLift invokes to activate a new game session. Amazon GameLift calls this function in response to a client request CreateGameSession. The callback function passes a GameSession object, as defined in the *Amazon GameLift API Reference*. This property is a delegate function defined as `DECLARE_DELEGATE_OneParam(FOnStartGameSession, Aws::GameLift::Server::Model::GameSession);` **Type:** `FOnStartGameSession` **Required:** Yes
OnUpdateGameSession	The callback function that Amazon GameLift invokes to pass an updated game session object to the server process. Amazon GameLift calls this function when a match backfill request has been processed to provide updated matchmaker data. It passes a GameSession object, a status update (`updateReason`), and the match backfill ticket ID. This property is a delegate function defined as `DECLARE_DELEGATE_OneParam(FOnUpdateGameSession, Aws::GameLift::Server::Model::UpdateGameSession);` **Type:** `FOnUpdateGameSession` **Required:** No

Port	The port number the server process listens on for new player connections. The value must fall into the port range configured for any fleet deploying this game server build. This port number is included in game session and player session objects, which game sessions use when connecting to a server process. **Type:** `int` **Required:** Yes

UpdateGameSession

This data type updates to a game session object, which includes the reason that the game session was updated and the related backfill ticket ID if backfill is used to fill player sessions in the game session.

Properties	Description
GameSession	A GameSession object defined by the Amazon GameLift API. The `GameSession` object contains properties describing a game session. **Type:** `Aws::GameLift::Server::GameSession` **Required:** No
UpdateReason	The reason that the game session is being updated. **Type:** `enum class UpdateReason` • MATCHMAKING_DATA_UPDATED • BACKFILL_FAILED • BACKFILL_TIMED_OUT • BACKFILL_CANCELLED

Properties	Description
	Required: No
BackfillTicketId	The ID of the backfill ticket attempting to update the game session. **Type:** `char[]` **Required:** No

GameSession

This data type provides details of a game session.

Properties	Description
GameSessionId	A unique identifier for the game session. A game session ARN has the following format: `arn:aws:gamelift:<region>::gamesession/<fleet ID>/<custom ID string or idempotency token> .` **Type:** `char[]` **Required**: No
Name	A descriptive label of the game session. **Type:** `char[]` **Required**: No
FleetId	A unique identifier for the fleet that the game session is running on. **Type:** `char[]` **Required**: No

Properties	Description
MaximumPlayerSessionCount	The maximum number of player connections to the game session. **Type:** `int` **Required**: No
Port	The port number for the game session. To connect to a Amazon GameLift game server, an app needs both the IP address and port number. **Type:** `int` **Required**: No
IpAddress	The IP address of the game session. To connect to a Amazon GameLift game server, an app needs both the IP address and port number. **Type:** `char[]` **Required**: No
GameSessionData	A set of custom game session properties, formatted as a single string value. **Type:** `char[]` **Required**: No

Properties	Description
MatchmakerData	Information about the matchmaking process that was used to create the game session, in JSON syntax, formatted as a string. In addition to the matchmaking configuration used, it contains data on all players assigned to the match, including player attributes and team assignments. **Type:** `char[]` **Required**: No
GameProperties	A set of custom properties for a game session, formatted as key:value pairs. These properties are passed with a request to start a new game session. **Type:** `GameProperty[]` **Required**: No

Properties	Description
DnsName	The DNS identifier assigned to the instance that's running the game session. Values have the following format: • TLS-enabled fleets: `<unique identifier>.<region identifier>.amazongamelift.com .` • Non-TLS-enabled fleets: `ec2-<unique identifier>.compute.amazonaws.com .` When connecting to a game session that's running on a TLS-enabled fleet, you must use the DNS name, not the IP address. **Type:** `char[]` **Required**: No

FServerParameters

Information used to maintain the connection between an Amazon GameLift Anywhere server and the Amazon GameLift service. This information is used when launching new server processes with InitSDK(). For servers hosted on Amazon GameLift managed EC2 instances, use an empty object.

Properties	Description
webSocketUrl	The `GameLiftServerSdkEndpoint` Amazon GameLift returns when you `RegisterCompute` for a Amazon GameLift Anywhere compute resource. **Type:** `char[]` **Required**: Yes

Properties	Description
processId	A unique identifier registered to the server process hosting your game. **Type:** `char[]` **Required**: Yes
hostId	The `HostID` is the `ComputeName` used when you registered your compute. For more information see, RegisterCompute. **Type:** `char[]` **Required**: Yes
fleetId	The unique identifier of the fleet that the compute is registered to. For more informati on see, RegisterCompute. **Type:** `char[]` **Required**: Yes
authToken	The authentication token generated by Amazon GameLift that authenticates your server to Amazon GameLift. For more information see, GetComputeAuthToken. **Type:** `char[]` **Required**: Yes

FStartMatchBackfillRequest

Information used to create a matchmaking backfill request. The game server communicates this information to Amazon GameLift in a StartMatchBackfill() call.

Properties	Description
GameSessionArn	A unique game session identifier. The API operation `GetGameSessionId` returns the identifier in ARN format. **Type:** `char[]` **Required**: Yes
MatchmakingConfigurationArn	A unique identifier, in the form of an ARN, for the matchmaker to use for this request. The matchmaker ARN for the original game session is in the game session object in the matchmaker data property. Learn more about matchmaker data in Work with matchmaker data. **Type:** `char[]` **Required**: Yes
Players	A set of data representing all players who are in the game session. The matchmaker uses this information to search for new players who are good matches for the current players. **Type:** `TArray<FPlayer>` **Required**: Yes
TicketId	A unique identifier for a matchmaking or match backfill request ticket. If you don't provide a value, Amazon GameLift generates one. Use this identifier to track the match backfill ticket status or cancel the request if needed. **Type:** `char[]`

Properties	Description
	Required: No

FPlayer

This data type represents a player in matchmaking. When starting a matchmaking request, a player has a player ID, attributes, and possibly latency data. Amazon GameLift adds team information after a match is made.

Properties	Description
LatencyInMS	A set of values expressed in milliseconds that indicate the amount of latency that a player experiences when connected to a location. If this property is used, the player is only matched for locations listed. If a matchmaker has a rule that evaluates player latency, players must report latency to be matched. **Type:** `TMap>FString, int32<` **Required**: No
PlayerAttributes	A collection of key:value pairs containing player information for use in matchmaking. Player attribute keys must match the PlayerAttributes used in a matchmaking rule set. For more information about player attributes, see AttributeValue. **Type:** `TMap>FString, FAttributeValue<` **Required**: No
PlayerId	A unique identifier for a player.

Properties	Description
	Type: `std::string` **Required**: No
Team	The name of the team that the player is assigned to in a match. You define team name in the matchmaking rule set. **Type:** `FString` **Required**: No

FGameLiftDescribePlayerSessionsRequest

An object that specifies which player sessions to retrieve. The server process provides this information with a DescribePlayerSessions() call to Amazon GameLift.

Properties	Description
GameSessionId	A unique game session identifier. Use this parameter to request all player sessions for the specified game session. Game session ID format is `FString`. The `GameSessionID` is a custom ID string or a **Type:** `std::string` **Required**: No
PlayerSessionId	The unique identifier for a player session. Use this parameter to request a single specific player session. **Type:** `FString` **Required**: No

Properties	Description
PlayerId	The unique identifier for a player. Use this parameter to request all player sessions for a specific player. See Generate player IDs. **Type:** `FString` **Required**: No
PlayerSessionStatusFilter	The player session status to filter results on. Possible player session statuses include: • RESERVED – The player session request was received, but the player hasn't connected to the server process or been validated. • ACTIVE – The player was validated by the server process and is connected. • COMPLETED – The player connection dropped. • TIMEDOUT – A player session request was received, but the player didn't connect or wasn't validated within the time-out limit (60 seconds). **Type:** `FString` **Required**: No
NextToken	The token indicating the start of the next page of results. To specify the start of the result set, don't provide a value. If you provide a player session ID, this parameter is ignored. **Type:** `FString` **Required**: No

Properties	Description
Limit	The maximum number of results to return. If you provide a player session ID, this parameter is ignored. **Type:** `int` **Required**: No

FStopMatchBackfillRequest

Information used to cancel a matchmaking backfill request. The game server communicates this information to Amazon GameLift service in a StopMatchBackfill() call.

Properties	Description
GameSessionArn	A unique game session identifier of the request being canceled. **Type:** `FString` **Required**: Yes
MatchmakingConfigurationArn	A unique identifier of the matchmaker this request was sent to. **Type:** `FString` **Required**: Yes
TicketId	A unique identifier of the backfill request ticket to be canceled. **Type:** `FString` **Required**: Yes

FAttributeValue

Use these values in FPlayer attribute key-value pairs. This object lets you specify an attribute value using any of the valid data types: string, number, string array, or data map. Each `AttributeValue` object can use only one of the available properties.

Properties	Description
attrType	Specifies the type of attribute value. **Type:** An `FAttributeType` enum value. **Required:** No
S	Represents a string attribute value. **Type:** `FString` **Required:** No
N	Represents a numeric attribute value. **Type:** `double` **Required:** No
SL	Represents an array of string attribute values. **Type:** `TArray<FString>` **Required:** No
SDM	Represents a dictionary of string keys and double values. **Type:** `TMap<FString, double>` **Required:** No

FGameLiftGetFleetRoleCredentialsRequest

This data type provides role credentials that extend limited access to your AWS resources to the game server. For more information see, Set up an IAM service role for Amazon GameLift.

Properties	Description
RoleArn	The Amazon Resource Name (ARN) of the service role that extends limited access to your AWS resources. **Type:** `FString` **Required**: No
RoleSessionName	The name of the session describing the use of the role credentials. **Type:** `FString` **Required**: No

FGameLiftLongOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. **Type:** `long` **Required:** No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. **Type:** `long&&`

Properties	Description
	Required: No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called "FGameLiftError" **Required:** No

FGameLiftStringOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. **Type:** `FString` **Required:** No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. **Type:** `FString&&` **Required**: No
Success	Whether the action was successful or not. **Type:** `bool`

Properties	Description
	Required: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called “FGameLiftError” **Required:** No

FGameLiftDescribePlayerSessionsOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. **Type:** the section called “FGameLiftDescribePlayerSessionsResult” **Required:** No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. **Type:** `FGameLiftDescribePlayerSessionsResult&&` **Required**: No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful.

Properties	Description
	Type: the section called “FGameLiftError” **Required:** No

FGameLiftDescribePlayerSessionsResult

Properties	Description
PlayerSessions	**Type:** `TArray<FGameLiftPlayerSession>` **Required:** Yes
NextToken	The token indicating the start of the next page of results. To specify the start of the result set, don't provide a value. If you provide a player session ID, this parameter is ignored. **Type:** `FString` **Required**: No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called “FGameLiftError” **Required:** No

FGenericOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called "FGameLiftError" **Required:** No

FGameLiftPlayerSession

Properties	Description
CreationTime	**Type:** `long` **Required:** Yes
FleetId	**Type:** `FString` **Required:** Yes
GameSessionId	**Type:** `FString` **Required:** Yes
IpAddress	**Type:** `FString` **Required:** Yes
PlayerData	**Type:** `FString`

Properties	Description
	Required: Yes
PlayerId	**Type:** `FString` **Required:** Yes
PlayerSessionId	**Type:** `FString` **Required:** Yes
Port	**Type:** `int` **Required:** Yes
Status	**Type:** A `PlayerSessionStatus` enum. **Required:** Yes
TerminationTime	**Type:** `long` **Required:** Yes
DnsName	**Type:** `FString` **Required:** Yes

FGameLiftGetComputeCertificateOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. **Type:** the section called "FGameLiftGetComputeCertificateResult" **Required:** No

Properties	Description
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. **Type:** `FGameLiftGetComputeCertificateResult&&` **Required**: No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called "FGameLiftError" **Required:** No

FGameLiftGetComputeCertificateResult

The path to the TLS certificate on your compute and the compute's host name.

Properties	Description
CertificatePath	**Type:** `FString` **Required:** Yes
ComputeName	**Type:** `FString` **Required:** Yes

FGameLiftGetFleetRoleCredentialsOutcome

This data type results from an action and produces an object with the following properties:

Properties	Description
Result	The result of the action. **Type:** the section called "FGetFleetRoleCredentialsResult" **Required:** No
ResultWithOwnership	The result of the action, cast as an rvalue, so that the calling code can take ownership of the object. **Type:** `FGameLiftGetFleetRoleCredentialsResult&&` **Required**: No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called "FGameLiftError" **Required:** No

FGetFleetRoleCredentialsResult

Properties	Description
AccessKeyId	The access key ID to authenticate and provide access to your AWS resources. **Type:** `FString` **Required:** No
AssumedRoleId	The ID of the user that the service role belongs to. **Type:** `FString` **Required:** No
AssumedRoleUserArn	The Amazon Resource Name (ARN) of the user that the service role belongs to. **Type:** `FString` **Required:** No
Expiration	The amount of time until your session credentials expire. **Type:** `FDateTime` **Required:** No
SecretAccessKey	The secret access key ID for authentication. **Type:** `FString` **Required:** No
SessionToken	A token to identify the current active session interacting with your AWS resources. **Type:** `FString`

Properties	Description
	Required: No
Success	Whether the action was successful or not. **Type:** `bool` **Required**: Yes
Error	The error that occurred if the action was unsuccessful. **Type:** the section called “GameLiftError” **Required:** No

FGameLiftError

Properties	Description
ErrorType	The type of error. **Type:** A `GameLiftErrorType` enum. **Required:** No
ErrorName	The name of the error. **Type:** `std::string` **Required:** No
ErrorMessage	The error message. **Type:** `std::string` **Required:** No

Enums

Enums defined for the Amazon GameLift server SDK (Unreal) are defined as follows:

FAttributeType

- **NONE**
- **STRING**
- **DOUBLE**
- **STRING_LIST**
- **STRING_DOUBLE_MAP**

GameLiftErrorType

String value indicating the error type. Valid values include:

- **SERVICE_CALL_FAILED** – A call to an AWS service has failed.
- **LOCAL_CONNECTION_FAILED** – The local connection to Amazon GameLift failed.
- **NETWORK_NOT_INITIALIZED** – The network has not been initialized.
- **GAMESESSION_ID_NOT_SET** – The game session ID has not been set.
- **BAD_REQUEST_EXCEPTION**
- **INTERNAL_SERVICE_EXCEPTION**
- **ALREADY_INITIALIZED** – The Amazon GameLift Server or Client has already been initialized with Initialize().
- **FLEET_MISMATCH** – The target fleet does not match the fleet of a gameSession or playerSession.
- **GAMELIFT_CLIENT_NOT_INITIALIZED** – The Amazon GameLift client has not been initialized.
- **GAMELIFT_SERVER_NOT_INITIALIZED** – The Amazon GameLift server has not been initialized.
- **GAME_SESSION_ENDED_FAILED** – The Amazon GameLift Server SDK could not contact the service to report the game session ended.
- **GAME_SESSION_NOT_READY** – The Amazon GameLift Server Game Session was not activated.
- **GAME_SESSION_READY_FAILED** – The Amazon GameLift Server SDK could not contact the service to report the game session is ready.
- **INITIALIZATION_MISMATCH** – A client method was called after Server::Initialize(), or vice versa.

- **NOT_INITIALIZED** – The Amazon GameLift Server or Client has not been initialized with Initialize().
- **NO_TARGET_ALIASID_SET** – A target aliasId has not been set.
- **NO_TARGET_FLEET_SET** – A target fleet has not been set.
- **PROCESS_ENDING_FAILED** – The Amazon GameLift Server SDK could not contact the service to report the process is ending.
- **PROCESS_NOT_ACTIVE** – The server process is not yet active, not bound to a GameSession, and cannot accept or process PlayerSessions.
- **PROCESS_NOT_READY** – The server process is not yet ready to be activated.
- **PROCESS_READY_FAILED** – The Amazon GameLift Server SDK could not contact the service to report the process is ready.
- **SDK_VERSION_DETECTION_FAILED** – SDK version detection failed.
- **STX_CALL_FAILED** – A call to the XStx server backend component has failed.
- **STX_INITIALIZATION_FAILED** – The XStx server backend component has failed to initialize.
- **UNEXPECTED_PLAYER_SESSION** – An unregistered player session was encountered by the server.
- **WEBSOCKET_CONNECT_FAILURE**
- **WEBSOCKET_CONNECT_FAILURE_FORBIDDEN**
- **WEBSOCKET_CONNECT_FAILURE_INVALID_URL**
- **WEBSOCKET_CONNECT_FAILURE_TIMEOUT**
- **WEBSOCKET_RETRIABLE_SEND_MESSAGE_FAILURE** – Retriable failure to send a message to the GameLift Service WebSocket.
- **WEBSOCKET_SEND_MESSAGE_FAILURE** – Failure to send a message to the GameLift Service WebSocket.
- **MATCH_BACKFILL_REQUEST_VALIDATION** – Validation of the request failed.
- **PLAYER_SESSION_REQUEST_VALIDATION** – Validation of the request failed.

EPlayerSessionCreationPolicy

String value indicating whether the game session accepts new players. Valid values include:

- **ACCEPT_ALL** – Accept all new player sessions.
- **DENY_ALL** – Deny all new player sessions.

- **NOT_SET** – The game session is not set to accept or deny new player sessions.

EPlayerSessionStatus

- **ACTIVE**
- **COMPLETED**
- **NOT_SET**
- **RESERVED**
- **TIMEDOUT**

Amazon GameLift Unreal Engine server SDK 3.x reference

You can use this Amazon GameLift Unreal Engine server SDK 3.x reference to help you prepare your multiplayer game for use with Amazon GameLift. For details about the integration process, see Add Amazon GameLift to your game server.

Topics

- Amazon GameLift server SDK reference for Unreal Engine: Actions
- Amazon GameLift server SDK reference for Unreal Engine: Data types

Amazon GameLift server SDK reference for Unreal Engine: Actions

This Amazon GameLift Server SDK reference can help you prepare your Unreal Engine game projects for use with Amazon GameLift. For details on the integration process, see Add Amazon GameLift to your game server.

This API is defined in `GameLiftServerSDK.h` and `GameLiftServerSDKModels.h`.

To set up the Unreal Engine plugin and see code examples Integrate Amazon GameLift into an Unreal Engine project.

- Actions
- Data types

AcceptPlayerSession()

Notifies the Amazon GameLift service that a player with the specified player session ID has connected to the server process and needs validation. Amazon GameLift verifies that the player

session ID is valid—that is, that the player ID has reserved a player slot in the game session. Once validated, Amazon GameLift changes the status of the player slot from RESERVED to ACTIVE.

Syntax

```
FGameLiftGenericOutcome AcceptPlayerSession(const FString& playerSessionId)
```

Parameters

playerSessionId

Unique ID issued by the Amazon GameLift service in response to a call to the AWS SDK Amazon GameLift API action CreatePlayerSession. The game client references this ID when connecting to the server process.

Type: FString

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

ActivateGameSession()

Notifies the Amazon GameLift service that the server process has activated a game session and is now ready to receive player connections. This action should be called as part of the `onStartGameSession()` callback function, after all game session initialization has been completed.

Syntax

```
FGameLiftGenericOutcome ActivateGameSession()
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

DescribePlayerSessions()

Retrieves player session data, including settings, session metadata, and player data. Use this action to get information for a single player session, for all player sessions in a game session, or for all player sessions associated with a single player ID.

Syntax

```
FGameLiftDescribePlayerSessionsOutcome DescribePlayerSessions(const
 FGameLiftDescribePlayerSessionsRequest &describePlayerSessionsRequest)
```

Parameters

describePlayerSessionsRequest

A FDescribePlayerSessionsRequest object describing which player sessions to retrieve.

Required: Yes

Return value

If successful, returns a FDescribePlayerSessionsRequest object containing a set of player session objects that fit the request parameters. Player session objects have a structure identical to the AWS SDK Amazon GameLift API PlayerSession data type.

GetGameSessionId()

Retrieves the ID of the game session currently being hosted by the server process, if the server process is active.

Syntax

```
FGameLiftStringOutcome GetGameSessionId()
```

Parameters

This action has no parameters.

Return value

If successful, returns the game session ID as an `FGameLiftStringOutcome` object. If not successful, returns an error message.

GetInstanceCertificate()

Retrieves the file location of a pem-encoded TLS certificate that is associated with the fleet and its instances. AWS Certificate Manager generates this certificate when you create a new fleet with the certificate configuration set to GENERATED. Use this certificate to establish a secure connection with a game client and to encrypt client/server communication.

Syntax

```
FGameLiftGetInstanceCertificateOutcome GetInstanceCertificate()
```

Parameters

This action has no parameters.

Return value

If successful, returns a `GetInstanceCertificateOutcome` object containing the location of the fleet's TLS certificate file and certificate chain, which are stored on the instance. A root certificate file, extracted from the certificate chain, is also stored on the instance. If not successful, returns an error message.

For more information about the certificate and certificate chain data, see GetCertificate Response Elements in the AWS Certificate Manager API Reference.

GetSdkVersion()

Returns the current version number of the SDK built into the server process.

Syntax

```
FGameLiftStringOutcome GetSdkVersion();
```

Parameters

This action has no parameters.

Return value

If successful, returns the current SDK version as an `FGameLiftStringOutcome` object. The returned string includes the version number only (ex. "3.1.5"). If not successful, returns an error message.

Example

```
Aws::GameLift::AwsStringOutcome SdkVersionOutcome =
    Aws::GameLift::Server::GetSdkVersion();
```

InitSDK()

Initializes the Amazon GameLift SDK. This method should be called on launch, before any other Amazon GameLift-related initialization occurs.

Syntax

```
FGameLiftGenericOutcome InitSDK()
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

ProcessEnding()

Notifies the Amazon GameLift service that the server process is shutting down. This method should be called after all other cleanup tasks, including shutting down all active game sessions. This method should exit with an exit code of 0; a non-zero exit code results in an event message that the process did not exit cleanly.

Syntax

```
FGameLiftGenericOutcome ProcessEnding()
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

ProcessReady()

Notifies the Amazon GameLift service that the server process is ready to host game sessions. Call this method after successfully invoking InitSDK() and completing setup tasks that are required before the server process can host a game session. This method should be called only once per process.

Syntax

```
FGameLiftGenericOutcome ProcessReady(FProcessParameters &processParameters)
```

Parameters

FProcessParameters

A FProcessParameters object communicating the following information about the server process:

- Names of callback methods, implemented in the game server code, that the Amazon GameLift service invokes to communicate with the server process.
- Port number that the server process is listening on.
- Path to any game session-specific files that you want Amazon GameLift to capture and store.

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

Example

See the sample code in Using the Unreal Engine Plugin.

RemovePlayerSession()

Notifies the Amazon GameLift service that a player with the specified player session ID has disconnected from the server process. In response, Amazon GameLift changes the player slot to available, which allows it to be assigned to a new player.

Syntax

```
FGameLiftGenericOutcome RemovePlayerSession(const FString& playerSessionId)
```

Parameters

playerSessionId

Unique ID issued by the Amazon GameLift service in response to a call to the AWS SDK Amazon GameLift API action CreatePlayerSession. The game client references this ID when connecting to the server process.

Type: FString

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

StartMatchBackfill()

Sends a request to find new players for open slots in a game session created with FlexMatch. See also the AWS SDK action StartMatchBackfill(). With this action, match backfill requests can be initiated by a game server process that is hosting the game session. Learn more about the FlexMatch backfill feature.

This action is asynchronous. If new players are successfully matched, the Amazon GameLift service delivers updated matchmaker data using the callback function `OnUpdateGameSession()`.

A server process can have only one active match backfill request at a time. To send a new request, first call StopMatchBackfill() to cancel the original request.

Syntax

```
FGameLiftStringOutcome StartMatchBackfill (FStartMatchBackfillRequest
 &startBackfillRequest);
```

Parameters

FStartMatchBackfillRequest

A FStartMatchBackfillRequest object that communicates the following information:

- A ticket ID to assign to the backfill request. This information is optional; if no ID is provided, Amazon GameLift will autogenerate one.

- The matchmaker to send the request to. The full configuration ARN is required. This value can be acquired from the game session's matchmaker data.
- The ID of the game session that is being backfilled.
- Available matchmaking data for the game session's current players.

Required: Yes

Return value

If successful, returns the match backfill ticket as a `FGameLiftStringOutcome` object. If not successful, returns an error message. Ticket status can be tracked using the AWS SDK action DescribeMatchmaking().

StopMatchBackfill()

Cancels an active match backfill request that was created with StartMatchBackfill(). See also the AWS SDK action StopMatchmaking(). Learn more about the FlexMatch backfill feature.

Syntax

```
FGameLiftGenericOutcome StopMatchBackfill (FStopMatchBackfillRequest
 &stopBackfillRequest);
```

Parameters

StopMatchBackfillRequest

A FStopMatchBackfillRequest object identifying the matchmaking ticket to cancel:

- ticket ID assigned to the backfill request being canceled
- matchmaker the backfill request was sent to
- game session associated with the backfill request

Required: Yes

Return value

Returns a generic outcome consisting of success or failure with an error message.

TerminateGameSession()

This method is deprecated with version 4.0.1. Instead, the server process should call ProcessEnding() after a game session has ended.

Notifies the Amazon GameLift service that the server process has ended the current game session. This action is called when the server process will remain active and ready to host a new game session. It should be called only after your game session termination procedure is complete, because it signals to Amazon GameLift that the server process is immediately available to host a new game session.

This action is not called if the server process will be shut down after the game session stops. Instead, call ProcessEnding() to signal that both the game session and the server process are ending.

Syntax

```
FGameLiftGenericOutcome TerminateGameSession()
```

Parameters

This action has no parameters.

Return value

Returns a generic outcome consisting of success or failure with an error message.

UpdatePlayerSessionCreationPolicy()

Updates the current game session's ability to accept new player sessions. A game session can be set to either accept or deny all new player sessions. (See also the `UpdateGameSession()` action in the *Amazon GameLift 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.

Amazon GameLift server SDK reference for Unreal Engine: Data types

This Amazon GameLift Server SDK reference can help you prepare your Unreal Engine game projects for use with Amazon GameLift. For details on the integration process, see Add Amazon GameLift to your game server.

This API is defined in `GameLiftServerSDK.h` and `GameLiftServerSDKModels.h`.

To set up the Unreal Engine plugin and see code examples Integrate Amazon GameLift into an Unreal Engine project.

- Actions
- 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.

Type: String

Required: No

PlayerSessionId

Unique identifier for a player session.

Type: String

Required: No

PlayerSessionStatusFilter

Player session status to filter results on. Possible player session statuses include the following:

- RESERVED – The player session request has been received, but the player has not yet connected to the server process and/or been validated.
- ACTIVE – The player has been validated by the server process and is currently connected.
- COMPLETED – The player connection has been dropped.
- TIMEDOUT – A player session request was received, but the player did not connect and/or was not validated within the time-out limit (60 seconds).

Type: String

Required: No

FProcessParameters

This data type contains the set of parameters sent to the Amazon GameLift service in a ProcessReady() call.

Contents

port

Port number the server process will listen on for new player connections. The value must fall into the port range configured for any fleet deploying this game server build. This port number is included in game session and player session objects, which game sessions use when connecting to a server process.

Type: Integer

Required: Yes

logParameters

Object with a list of directory paths to game session log files.

Type: TArray<FString>

Required: No

onStartGameSession

Name of callback function that the Amazon GameLift service invokes to activate a new game session. Amazon GameLift calls this function in response to the client request CreateGameSession. The callback function takes a GameSession object (defined in the *Amazon GameLift Service API Reference*).

Type: FOnStartGameSession

Required: Yes

onProcessTerminate

Name of callback function that the Amazon GameLift service invokes to force the server process to shut down. After calling this function, Amazon GameLift waits five minutes for the server process to shut down and respond with a ProcessEnding() call before it shuts down the server process.

Type: FSimpleDelegate

Required: No

onHealthCheck

Name of callback function that the Amazon GameLift service invokes to request a health status report from the server process. Amazon GameLift calls this function every 60 seconds. After calling this function Amazon GameLift waits 60 seconds for a response, and if none is received. records the server process as unhealthy.

Type: FOnHealthCheck

Required: No

onUpdateGameSession

Name of callback function that the Amazon GameLift service invokes to pass an updated game session object to the server process. Amazon 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

Required: No

FStartMatchBackfillRequest

This data type is used to send a matchmaking backfill request. The information is communicated to the Amazon GameLift service in a StartMatchBackfill() call.

Contents

GameSessionArn

Unique game session identifier. The API action GetGameSessionId() returns the identifier in ARN format.

Type: FString

Required: Yes

MatchmakingConfigurationArn

Unique identifier, in the form of an ARN, for the matchmaker to use for this request. To find the matchmaker that was used to create the original game session, look in the game session object, in the matchmaker data property. Learn more about matchmaker data in Work with matchmaker data.

Type: FString

Required: Yes

Players

A set of data representing all players who are currently in the game session. The matchmaker uses this information to search for new players who are good matches for the current players. See the *Amazon GameLift 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 Amazon GameLift service in a StopMatchBackfill() call.

Contents

GameSessionArn

Unique game session identifier associated with the request being canceled.

Type: FString

Required: Yes

MatchmakingConfigurationArn

Unique identifier of the matchmaker this request was sent to.

Type: FString

Required: Yes

TicketId

Unique identifier of the backfill request ticket to be canceled.

Type: FString

Required: Yes

Game session placement events

Amazon 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. 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 Amazon 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.

Detail syntax:

placementId

A unique identifier assigned to the game session placement request.

port

The port number for the new game session.

gameSessionArn

The ARN identifier for the new game session.

ipAddress

The IP address of the game session.

dnsName

The DNS identifier assigned to the instance that is running the new game session. The value format is different depending on whether the instance running the game session is TLS-enabled. When connecting to a game session on a TLS-enabled fleet, players must use the DNS name, not the IP address.

TLS-enabled fleets: `<unique identifier>.<region identifier>.amazongamelift.com`.

Non-TLS-enabled fleets: `ec2-<unique identifier>.compute.amazonaws.com`.

startTime

Time stamp indicating when this request was placed in the queue.

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


```
{
  "version": "0",
  "id": "1111aaaa-bb22-cc33-dd44-5555eeee66ff",
  "detail-type": "GameLift Queue Placement Event",
  "source": "aws.gamelift",
  "account": "123456789012",
  "time": "2021-03-01T15:50:52Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:gamelift:us-west-2:123456789012:gamesessionqueue/MegaFrogRace-NA"
  ],
  "detail": {
    "type": "PlacementFulfilled",
    "placementId": "9999ffff-88ee-77dd-66cc-5555bb44aa",
    "port": "6262",
    "gameSessionArn": "arn:aws:gamelift:us-west-2::gamesession/
fleet-2222bbbb-33cc-44dd-55ee-6666ffff77aa/4444dddd-55ee-66ff-77aa-8888bbbb99cc",
    "ipAddress": "98.987.98.987",
    "dnsName": "ec2-12-345-67-890.us-west-2.compute.amazonaws.com",
    "startTime": "2021-03-01T15:50:49.741Z",
    "endTime": "2021-03-01T15:50:52.084Z",
    "gameSessionRegion": "us-west-2",
    "placedPlayerSessions": [
      {
        "playerId": "player-1"
        "playerSessionId": "psess-123213123234124123123"
      }
```

```
    ]
  }
}
```

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": [
    "arn:aws:gamelift:us-west-2:123456789012:gamesessionqueue/MegaFrogRace-NA"
  ],
  "detail": {

    "type": "PlacementCancelled",
    "placementId": "9999ffff-88ee-77dd-66cc-5555bb44aa",
    "startTime": "2021-03-01T15:50:49.741Z",
    "endTime": "2021-03-01T15:50:52.084Z"
  }
```

```
}
```

PlacementTimedOut

Game session placement did not successfully complete before the queue's time limit expired. The placement request can be resubmitted as needed.

Detail:

placementId

A unique identifier assigned to the game session placement request.

startTime

Time stamp indicating when this request was placed in the queue.

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": [
    "arn:aws:gamelift:us-west-2:123456789012:gamesessionqueue/MegaFrogRace-NA"
  ],
  "detail": {

    "type": "PlacementTimedOut",
    "placementId": "9999ffff-88ee-77dd-66cc-5555bb44aa",
    "startTime": "2021-03-01T15:50:49.741Z",
    "endTime": "2021-03-01T15:50:52.084Z"
  }
}
```

PlacementFailed

Amazon GameLift 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",
  "account": "252386620677",
  "time": "2021-03-01T15:50:52Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:gamelift:us-west-2:252386620677:gamesessionqueue/MegaFrogRace-NA"
  ],
  "detail": {

    "type": "PlacementFailed",
    "placementId": "e4a1119a-39af-45cf-a990-ef150fe0d453",
    "startTime": "2021-03-01T15:50:49.741Z",
    "endTime": "2021-03-01T15:50:52.084Z"
  }
}
```

Generating Amazon GameLift pricing estimates

With AWS Pricing Calculator, you can create a pricing estimate for Amazon GameLift. You don't need an AWS account or in-depth knowledge of AWS to use the calculator.

AWS Pricing Calculator calculator guides you through the decisions that affect service costs to give you an idea of how much Amazon GameLift might cost for your game project. If you're not yet sure how you plan to use Amazon GameLift, then use the default values to generate an estimate. When planning for production usage, the calculator can help you test out potential scenarios and generate more accurate estimates.

You can use AWS Pricing Calculator to generate estimates for the following Amazon GameLift hosting options:

- Estimate Amazon GameLift hosting
- Estimate Amazon GameLift standalone FlexMatch

Estimate Amazon GameLift hosting

This option provides a cost estimate for hosting your games on Amazon GameLift managed servers, including the costs for server instance usage and data transfer. FlexMatch matchmaking is included in the cost for Amazon GameLift managed hosting.

If you are hosting or plan to host game servers in more than one AWS Region or on more than one instance type, create an estimate for each Region and instance type.

Amazon GameLift instances

This section helps you estimate the type and number of compute resources that you need to host game sessions for your players. Amazon GameLift uses Amazon Elastic Compute Cloud (Amazon EC2) instances to manage game servers. In Amazon GameLift, you deploy a fleet of instances with a specific instance type and operating system. If you have or plan to have multiple fleets, create an estimate for each fleet.

To get started, open the Configure Amazon GameLift page of AWS Pricing Calculator. Add a **Description**, choose a **Region**, and then choose **Estimate Amazon GameLift hosting (Instance + Data Transfer Out)**. Under **Amazon GameLift instances**, complete the following fields:

- **Peak concurrent players (peak CCU)**

  This is the maximum number of players who can connect to your game servers at the same time. This field indicates how much hosting capacity Amazon GameLift needs to meet peak player demand. Enter the daily peak number of players that you expect to host using instances in your chosen AWS Region.

  For example, if you want to let 1,000 players connect to your game at any one time, keep the default value of **1000**.

- **Average CCU per hour as a percentage of peak daily CCU**

  This is the average number of concurrent players per hour over a 24-hour period. We use this value to estimate the amount of sustained hosting capacity that Amazon GameLift needs to maintain for your players. If you're not sure what percentage value to use, keep the default value of **50** percent. For games with stable player demand, we recommend entering a value of **70** percent.

  For example, if your game has an average hourly CCU of 6,000 and a peak CCU of 10,000, then enter the value of **60** percent.

- **Game sessions per instance**

  This is the number of game sessions that each of your game server instances can host concurrently. Factors that can affect this number include the resource requirements of your game server, the number of players to host in each game session, and player performance expectations. If you know the number of concurrent game sessions for your game, then enter that value. Alternatively, keep the default value of **20**.

- **Players per game session**

  This is the average number of players who connect to a game session, as defined in your game design. If you have game modes with different number of players, estimate an average number of players per game session across your entire game. The default value is **8**.

- **Instance idle buffer %**

  This is the percentage of unused hosting capacity to maintain in reserve to handle sudden spikes in player demand. Buffer size is a percentage of the total number of instances in a fleet. The default value is **10** percent.

For example, with a 20 percent idle buffer, a fleet supporting players with 100 active instances maintains 20 idle instances.

- **Spot instance %**

  Amazon GameLift fleets can use a combination of On-Demand Instances and Spot Instances. While On-Demand Instances offer more reliable availability, Spot Instances offer a highly cost-efficient alternative. We recommend using a combination to optimize both cost savings and availability. For information about how Amazon GameLift uses Spot Instances, see On-Demand Instances versus Spot Instances.

  For this field, enter the percentage of Spot Instances to maintain in a fleet. We recommend a Spot Instance percentage between 50 and 85 percent. The default value is **50** percent.

  For example, if you deploy a fleet with 100 instances and specify **40** percent, Amazon GameLift works to maintain 60 On-Demand Instances and 40 Spot Instances.

- **Instance type**

  Amazon GameLift fleets can use a range of Amazon EC2 instance types that vary in computing power, memory, storage, and networking capabilities. When you configure a Amazon GameLift fleet, choose an instance type that best fits your game's needs. For information about selecting an instance type with Amazon GameLift, see Choosing Amazon GameLift compute resources.

  If you know the instance type that you're using or plan to use in your Amazon GameLift fleet, choose that type. If you're not sure what type to choose, consider choosing **c5.large**. This is a high-availability type with average size and capabilities.

- **Operating system**

  This field specifies the operating system that your game servers run on—either Linux or Windows. The default value is **Linux**.

Data transfer out (DTO)

This section helps you estimate the cost for traffic between your game clients and the game servers. Data transfer fees apply to outbound traffic only. Inbound data transfer has no cost.

On the Configure Amazon GameLift page of AWS Pricing Calculator, expand **Data transfer out (DTO)**, and then complete the following fields:

- **DTO estimate type**

  You can choose to estimate DTO in either of the following two ways, depending on how you track data transfer for your game.

  - **Per month (in GB)** – If you track monthly traffic for your game servers, choose this type.
  - **Per player** – If you track data transfer by player, choose this type. This is the default type.

    In the following field, you estimate per-player DTO based on the number of player hours that you calculated in the previous section.

- **DTO per month (in GB)**

  If you chose the **Per month (in GB)** DTO estimate type, then enter your estimated monthly DTO usage in GB from each instance, per Region.

- **DTO per player**

  If you chose the **Per player** DTO estimate type, then enter your game's estimated DTO usage per player in KB/sec. The default value is **4**.

When you're done configuring your Amazon GameLift pricing estimate, choose **Add to my estimate**. For more information about creating and managing estimates in AWS Pricing Calculator, see Create an estimate, configure a service, and add more services in the *AWS Pricing Calculator User Guide*.

Estimate Amazon GameLift standalone FlexMatch

This option provides a cost estimate for using FlexMatch matchmaking as a standalone service while hosting your games with another game server solution. This includes Amazon GameLift self-managed hosting with FleetIQ and on-premises hosting, peer-to-peer, or cloud compute primitive data types. Standalone FlexMatch costs are based on the computing power used.

If you have or plan to have more than one matchmaker in different AWS Regions, create an estimate for each Region.

> **Note**
>
> Amazon GameLift FlexMatch is available in the following Regions: US East (N.Virginia), US West (Oregon), Asia Pacific (Seoul), Asia Pacific (Sydney), Asia Pacific (Tokyo), Europe (Frankfurt), Europe (Ireland).

To get started, open the Configure Amazon GameLift page of AWS Pricing Calculator. Add a **Description**, choose a **Region**, and then choose **Estimate Amazon GameLift Standalone FlexMatch**. Under **Amazon GameLift FlexMatch**, complete the following fields:

- **Peak concurrent players (peak CCU)**

  This is the maximum number of players who can connect to your game servers at the same time and request matchmaking. Enter the daily peak number of players that you expect to match into game sessions in your chosen Region.

  For example, if you want to match as many as 1,000 players at a time, keep the default value of **1000**.

- **Average CCU per hour as a percentage of peak daily CCU**

  This is the average number of concurrent players per hour over a 24-hour period. This value helps estimate the volume of your matchmaking requests. If you're not sure what percentage value to use, keep the default value of **50** percent. For games with stable player demand, we recommend entering a value of **70** percent.

  For example, if your game has an average hourly CCU of 6,000 and a peak CCU of 10,000, then enter the value of **60** percent.

- **Number of players per match**

  This is the average number of players who match to a game session, as defined in your game design. If you have game modes with different numbers of players, estimate an average number of players per game session across your entire game. The default value is **8**.

- **Game duration (in minutes)**

  This is the average length of time that players remain in a game session from start to finish. This value helps determine how often players might require a new match. Enter an average game duration in minutes for your players. The default value is **1**.

- **Matchmaking rule complexity**

  Matchmaking rule complexity refers to the number and type of rules that you use to match players. The level of complexity of your rule set helps determine the amount of computing power required for each match.

  - **Lower complexity** – Choose this option if your matchmaking rule set includes few rules, uses simpler rule types (such as comparison rules), and has rules that form successful matches with fewer attempts.
  - **Higher complexity** – Choose this option if your matchmaking rule set includes multiple rules, uses more complex rule types (such as distance or latency rules), and has restrictive rules that result in more failures and require more matching attempts.

  For more information about rule complexity and pricing, see Amazon GameLift FlexMatch on the Amazon GameLift Pricing page.

When you're done configuring your Amazon GameLift FlexMatch pricing estimate, choose **Add to my estimate**. For more information about creating and managing estimates in AWS Pricing Calculator, see Create an estimate, configure a service, and add more services in the *AWS Pricing Calculator User Guide*.

Quotas and supported Regions

For AWS Amazon GameLift service quotas, see Amazon GameLift quotas.

For information about requesting quota increases for AWS resources, see AWS service lquotas.

For a list of the AWS Regions supporting Amazon GameLift, see Amazon GameLift Regions.

Amazon GameLift release notes

The Amazon GameLift release notes provide details about new features, updates, and fixes related to the service.

SDK versions

The following tables list all Amazon GameLift releases with SDK version information. There is no requirement to use comparable SDKs for your game server and client integrations. However, earlier versions of one SDK may not fully support the latest features in another.

For more information about Amazon GameLift SDKs, see Development support with Amazon GameLift.

To get the latest Amazon GameLift SDKs, see the Amazon GameLift SDKs download site.

Current version

	SDK
	for Unreal
	r ater

Previous versions

Service release	AWS SDK	Server SDK				Realti client SDK
		C# plugin for Unity	C++	C++ plugin for Unreal	Go	
2023-12-14	1.11.225 or later	5.1.0	5.1.1	5.1.0	5.0.0	1.2.0
2023-11-02	1.11.193 or later	5.1.0	5.1.1	5.1.0	5.0.0	1.2.0
2023-09-28	1.11.144 or later	5.1.0	5.1.1	5.1.0	5.0.0	1.2.0
2023-08-17	1.11.144 or later	5.1.0	5.1.1	5.1.0	5.0.0	1.2.0
2023-07-27	1.11.111 or later	5.1.0	5.1.0	5.0.2	5.0.0	1.2.0
2023-06-29	1.11.111 or later		5.0.4	5.0.2	5.0.0	1.2.0
2023-06-15	1.11.87 or later		5.0.4	5.0.2	5.0.0	1.2.0

Service release	AWS SDK	Server SDK				Realti client SDK
		C# plugin for Unity	C++	C++ plugin for Unreal	Go	
2023-05-25	1.11.87 or later		5.0.3	5.0.2	5.0.0	1.2.0
2023-04-20	1.11.63 or later		5.0.3	5.0.2	5.0.0	1.2.0
2023-04-13	1.10.21 or later		5.0.0	5.0.0	5.0.0	1.2.0
2023-02-09	1.10.21 or later		5.0.0	3.4.0	5.0.0	1.2.0
2023-01-31	1.10.21 or later	5.0.0	5.0.0	3.4.0	5.0.0	1.2.0
2022-12-01	1.10.21 or later		5.0.0	3.4.0		1.2.0
2022-08-25	1.9.333 or later		3.4.2	3.4.0		1.2.0
2021-10-28	1.9.133 or later		3.4.2	3.4.0		1.2.0
2021-06-03	1.8.168 or later		3.4.2	3.4.0		1.2.0

Service release	AWS SDK	Server SDK				Realti client SDK
		C# plugin for Unity	C++	C++ plugin for Unreal	Go	
2021-03-23	1.8.168 or later		3.4.1	3.3.3		1.1.0
2021-03-16	1.8.163 or later		3.4.1	3.3.3		1.1.0
2021-02-09	1.8.139 or later		3.4.1	3.3.3		1.1.0
2020-12-22	1.8.95 or later		3.4.1	3.3.3		1.1.0
2020-11-24	1.8.95 or later		3.4.1	3.3.2		1.1.0
2020-11-11	1.8.36 or later		3.4.1	3.3.2		1.1.0
2020-09-17	1.8.36 or later		3.4.1	3.3.2		1.1.0
2020-08-27	1.7.310 or later		3.4.0	3.3.1		1.1.0
2020-04-16	1.7.310 or later		3.4.0	3.3.1		1.1.0
2020-04-02	1.7.310 or later		3.4.0			1.1.0
2019-12-19	1.7.249 or later		3.4.0			1.1.0

Service release	AWS SDK	Server SDK				Realti client SDK
		C# plugin for Unity	C++	C++ plugin for Unreal	Go	
2019-11-14	1.7.210 or later		3.4.0			1.1.0
2019-10-24	1.7.210 or later		3.4.0			1.1.0
2019-09-03	1.7.175 or later		3.4.0			1.1.0
2019-07-09	1.7.140 or later		3.3.0			1.0.0
2019-04-25	1.7.91 or later		3.3.0			1.0.0
2019-03-07	1.7.65 or later		3.3.0			
2019-02-07	1.7.45 or later		3.3.0			
2018-12-14	1.6.20 or later		3.3.0			
2018-09-27	1.6.20 or later		3.2.1			
2018-06-14	1.4.47 or later		3.2.1			
2018-05-10	1.4.47 or later		3.2.1			

Service release	AWS SDK	Server SDK				Realti client SDK
		C# plugin for Unity	C++	C++ plugin for Unreal	Go	
2018-02-15	1.3.58 or later		3.2.1			
2018-02-08	1.3.52 or later		3.2.0			
2017-09-01	1.1.43 or later		3.1.7			
2017-08-16	1.1.31 or later		3.1.7			
2017-05-16	1.0.122 or later		3.1.5			
2017-04-11	1.0.103 or later		3.1.5			
2017-02-21	1.0.72 or later		3.1.5			
2016-11-18	1.0.31 or later		3.1.0			
2016-10-13	1.0.17 or later		3.1.0			
2016-09-01	0.14.9 or later		3.1.0			
2016-08-04	0.12.16 or later		3.0.7			

Release notes

The following release notes are in chronological order, with the latest updates listed first. Amazon GameLift was first released in 2016. For release notes dated earlier than those listed here, see the release date links in SDK versions.

February 13, 2024: Amazon GameLift launches improvements to SDKs, and simplifies installation of the Amazon GameLift plugin for Unreal Engine

Updated SDK versions:

- Go Server SDK, version 5.1.0
- C# Server SDK, version 5.1.2
- C++ Server SDK, version 5.1.2

We made the following improvements:

- Improved the reliability of the SDK by adding automatic reconnection in the event of network interruption.
- [Go] You can now call `InitSDK()` with or without server parameters. Game servers that run on Amazon GameLift managed EC2 fleets read the server parameters directly from environment variables. Game servers on Amazon GameLift Anywhere fleets must call `InitSDK()` with server parameters.

Updated plugin versions:

- Amazon GameLift plugin for Unreal Engine, version 1.1.0
- Amazon GameLift plugin for Unity, version 2.1.0
- C++ Server SDK Plugin for Unreal, version 5.1.1
- C# Server SDK Plugin for Unity, version 5.1.2

We made the following improvements:

- [Amazon GameLift plugin for Unreal Engine] Updated the installation instructions and simplified the packaging. This plugin now includes the latest version of the C++ Server SDK for Unreal.
- Upgraded the plugins to support the latest version of the GameLift Server SDK.

Learn more:

- Integrating games with the Amazon GameLift plugin for Unreal Engine, *Amazon GameLift Developer Guide*
- Amazon GameLift plugin and SDK downloads

December 14, 2023: Amazon GameLift adds ability to update the game properties of active game sessions

You've already been able to set game properties when creating game sessions, and to search game sessions for specified properties. Now you can also add and update these properties in an active game session.

For example, your players vote on a map that they want to play on. Your game client calls `UpdateGameSession` to modify a `GameProperty` value to `{"Key": "map", "Value":"jungle"}`. Your game then implements the new map for the players in the game session.

Game administrators can also retrieve useful data from game properties by using the `SearchGameSessions` operation. For example, administrators can list game sessions that have a `Status` value of `ACTIVE` and this game property: `{"Key": "map", "Value":"desert"}`.

Learn more:

- the section called "Add Amazon GameLift to a game client", *Amazon GameLift Developer Guide*
- GameProperty, *Amazon GameLift API Reference*
- UpdateGameSession, *Amazon GameLift API Reference*
- SearchGameSessions, *Amazon GameLift API Reference*

November 21, 2023: Amazon GameLift launches support for Infrastructure as Code tools like Terraform and Pulumi powered by AWS Cloud Control API

You can now manage your entire Amazon GameLift resource stack using Infrastructure as Code (IaC) tools. These tools include AWS CloudFormation, and also third-party tools such as Terraform and Pulumi. With this added support, you can now focus on building your game, and leverage DevOps strategies to take care of resource management, CI/CD, and deployment to your customers.

You can also now provision and configure all Amazon GameLift resources types by using the AWS Cloud Control API. You can continue to work with resources using the Amazon GameLift APIs or the AWS CloudFormation templates for Amazon GameLift.

For details about the Amazon GameLift resources available through IaC, see the Amazon GameLift resource type reference Amazon GameLift resource type reference.

In addition, you can now automatically scale your fleets using AWS CloudFormation templates or the AWS Cloud Control API by using the new Fleet property: `ScalingPolicies`.

The Cloud Control API gives developers a standard set of APIs to create, read, update, delete, and list resources (CRUDL) across hundreds of AWS services and multiple third-party tools like Terraform and Pulumi.

Learn more:

- AWS CloudFormation
- AWS Cloud Control API
- AWS CC Terraform Provider
- Pulumi

November 16, 2023: Amazon GameLift updates standalone plugin for Unity

Updated SDK versions: Amazon GameLift plugin for Unity, version 2.0.0

The Amazon GameLift plugin for Unity provides tools and workflows that streamline the steps to getting your Unity game up and running for cloud hosting with Amazon GameLift. Amazon GameLift is a fully managed service that lets game developers manage and scale dedicated game servers for session-based multiplayer games.

With this version, the plugin for Unity is updated to use the latest Amazon GameLift features, including server SDK version 5.x and support for local testing with Amazon GameLift Anywhere. The plugin is compatible with Unity versions Unity 2021.3 LTS and 2022.3 LTS.

Key plugin features include:

- Guided UI workflows in the Unity editor for the following scenarios:
  - Test your game integration with Amazon GameLift using your local workstation as a host. This workflow helps you set up an Amazon GameLift Anywhere fleet for your local machine, launch

instances of your game server and client, request a game session through Amazon GameLift, and join the game.

- Deploy cloud hosting solution for your integrated game server with Amazon GameLift managed EC2 and supporting AWS resources. This workflow helps you configure your game for cloud hosting, and provides three deployment scenarios:
  - Deploy the game server to a single fleet.
  - Deploy the game server to a set of low-cost Spot fleets in multiple AWS Regions.
  - Deploy the game server with a FlexMatch matchmaker.
- Ability to set up user profiles that link to an AWS account user and set a default AWS Region. You can maintain multiple profiles to work in different AWS accounts, account users, and regions.
- Special conveniences that help streamline the Amazon GameLift integration and deployment processes, including:
  - Each hosting solution includes supporting AWS resources, including an Amazon Cognito user pool that provides unique player IDs and player validation. The solutions also include an Amazon S3 bucket for storage, Amazon SNS event notification, AWS Lambda functions, and other resources.
  - For the Anywhere workflow, the plugin automates the required server parameter settings.
  - For the Amazon EC2 workflow, each deployment solution provides a built-in client backend service using Lambda functions. The backend service sits between the game client and the Amazon GameLift service and manages all direct calls to the Amazon GameLift service.
- Content for integration testing, including assets and code for a simple sample multiplayer game to illustrate game server and game client integration.
- Plugin documentation with detailed integration guidance and sample code.

All deployment scenarios, including for Anywhere and Amazon EC2 fleets, use AWS CloudFormation templates to describe and deploy the AWS resources for your game's solution. These templates are included in the Amazon GameLift plugin download. You can use them as is or customize them for your game.

Learn more:

- Amazon GameLift plugin for Unity guide for server SDK 5.x, *Amazon GameLift Developer Guide*
- Download the plugin from GitHub
- About Amazon GameLift hosting

- Amazon GameLift forum

November 2, 2023: Amazon GameLift adds support for shared credentials

Updated SDK versions: AWS SDK 1.11.193

The new Amazon GameLift shared credentials feature allows applications that are deployed on managed EC2 fleets to interact with other AWS resources. This update affects applications that you bundle and deploy along with game server binaries integrated with server SDK version 5.x or later. (Game server executables can already request credentials using the server SDK 5.x `GetFleetRoleCredentials()` action.)

For example, if you want to deploy your game server build with an Amazon CloudWatch agent to collect EC2 instance metrics and other data, the agent needs permission to interact with your CloudWatch resources. To do this, you must first set up an AWS Identity and Access Management IAM) role with permissions to use the CloudWatch resources, and then configure a fleet with the IAM role and shared credentials enabled. When Amazon GameLift deploys your game server build to each EC2 instance, it generates a shared credentials file and stores it on the instance. All applications on the instance can use the shared credentials. Amazon GameLift automatically refreshes the temporary credentials throughout the life of the instance.

You can enable shared credentials when you create a managed EC2 fleet using the following methods:

- In the Amazon GameLift console fleet creation workflow.
- When calling the Amazon GameLift service API operation `CreateFleet` using the new parameter `InstanceRoleCredentialsProvider`.
- When calling the AWS CLI operation `aws gamelift create-fleet` with the parameter `instance-role-credentials-provider`.

Learn more:

- Communicate with other AWS resources from your fleets, *Amazon GameLift Developer Guide*
- CreateFleet, InstanceRoleCredentialsProvider, *Amazon GameLift API Reference*
- Set up an IAM service role, *Amazon GameLift Developer Guide*

September 28, 2023: Amazon GameLift releases new standalone plugin for Unreal Engine

Updated SDK versions: Amazon GameLift plugin for Unreal Engine version 1.0.0

The Amazon GameLift plugin for Unreal Engine provides tools and workflows that streamline your steps to getting a game up and running with Amazon GameLift for cloud hosting. Amazon GameLift is a fully managed service that lets game developers manage and scale dedicated game servers for session-based multiplayer games. The plugin supports UE versions 5.0, 5.1, and 5.2. Key features include:

- Guided UI workflows in the Unreal editor]step through the following paths:
  - Test your game integration with Amazon GameLift using your local workstation as a host. This workflow helps you set up an Amazon GameLift Anywhere fleet for your local machine, launch instances of your game server and client, request a game session through Amazon GameLift, and get connection information for the new game session.
  - Deploy an Amazon EC2 cloud hosting solution for your integrated game server. This workflow helps you configure your game for cloud hosting, and provides three different deployment scenarios: deploy to a single fleet, deploy to a set of spot fleets in multiple regions, or deploy to a set of fleets with a FlexMatch matchmaker. The solution for each deployment scenario includes Amazon GameLift resources and supporting AWS resources.
- Ability to set up user profiles that link to an AWS account user and define a default AWS Region. You can maintain multiple profiles to work in different AWS accounts, account users, and regions.
- Special conveniences that help streamline the Amazon GameLift integration and deployment processes, including:
  - Each hosting solution includes supporting AWS resources, including a basic Amazon Cognito user pool that provides unique player IDs, an Amazon S3 bucket for storage, Amazon SNS event notification, and AWS Lambda functions.
  - For the Anywhere workflow, the plugin automates the required server parameter settings using command line arguments.
  - For the Amazon EC2 workflow, each deployment solution provides a built-in client backend service using Lambda functions. The backend service receives requests from game clients and passes them on to the Amazon GameLift service.
- Content for integration testing, including a starter game map and two testing maps with basic blueprints and UI elements.
- Plugin documentation with detailed integration guidance and sample code.

All deployment scenarios, including for Anywhere and Amazon EC2 fleets, use AWS CloudFormation templates to describe the solutions. The plugin uses these templates when deploying Amazon GameLift resources for your game. These templates are included in the Amazon GameLift plugin download and are editable. You can use them as is or modify them for your game.

Learn more:

- Integrating games with the Amazon GameLift plugin for Unreal Engine, *Amazon GameLift Developer Guide*
- Download the plugin from GitHub
- About Amazon GameLift hosting
- Amazon GameLift forum

August 17, 2023: Amazon GameLift offers game server hosting with AWS Graviton processors

Updated SDK versions: AWS SDK 1.11.144

With Amazon GameLift you can now host your games in the cloud using EC2 instances with AWS Graviton processors. Designed by AWS with Arm64-based processors, Graviton instances deliver the best price performance for cloud workloads using EC2, with up to 40% improvement over comparable x86-based instances. The latest Graviton3 processors offer up to 25% better compute performance over earlier versions.

With Amazon GameLift, you can now select from these new instances in the AWS Graviton family:

- Graviton2-based instances: c6g, c6gn, r6g, m6g, g5g
- Graviton3-based instances: c7g, r7g, m7g

Learn more:

- AWS Graviton Processor: Learn about the benefits and practical uses of Graviton-based EC2 instances.
- Getting started with Graviton: Get an overview of the Graviton-based instances and insights on how applications run on them depending on their operating system, languages, and run times.

> **Note**
>
> Graviton Arm instances require an Amazon GameLift server build on Linux OS. Server SDK 5.1.1 or newer is required for C++ and C#. Server SDK 5.0 or newer is required for Go. These instances provide no out-of-the-box support for Mono installation on Amazon Linux 2023 (AL2023) or Amazon Linux 2 (AL2).

July 27, 2023: Amazon GameLift releases server SDK 5.1.0 with added support for Unity development

Updated SDK versions: Server SDK for C++, C#/Unity, Unreal 5.1.0

The newest release of the Amazon GameLift server SDK delivers updates for C++, C#, and the Unreal plugin, and a new plugin for use with the Unity game engine. Game developers integrate the Amazon GameLift server SDK into game servers that they deploy for hosting on Amazon GameLift.

The latest server SDK version contains the following updates, which include a number of customer requests:

- **Download language-specific SDK packages** – The updated Amazon GameLift download site contains SDK packages for each language. You can download current or previous versions.
- **New C# server SDK plugin for Unity** – The new server SDK package for Unity contains built C# libraries that you can install using the package manager in Unity Editor (see the new Unity integration guide). These libraries include the required dependencies through UnityNuGet. You can use this plugin with Unity 2020.3 LTS, 2021.3 LTS and 2022.3 LTS for Windows and Mac OS. It supports Unity's .NET Framework and .NET Standard profiles, with .NET Standard 2.1 and .NET 4.x.
- **Consolidated .NET solution for C#** – The server SDK for C# now supports .NET Framework 4.6.2 (upgraded from 4.6.1) and .NET 6.0 in a single solution. .NET Standard 2.1 is available with the Unity-built libraries.
- **Server SDK 5.1.0 updates**
  - [C++, C#, Unreal] You can now call `InitSDK()` with or without server parameters. Game servers that run on Amazon GameLift managed EC2 fleets read the server parameters directly from environment variables. Game servers on Amazon GameLift Anywhere fleets must call `InitSDK()` with server parameters.

- [C++, C#, Unreal] Server SDK calls have improved error messaging.
- [C++ SDK] To improve Server SDK build times, the build flag `-DRUN_CLANG_FORMAT` is disabled by default . You can enable it with `-DRUN_CLANG_FORMAT=1`.
- [C++ SDK] When building the libraries without the standard libraries (`-DGAMELIFT_USE_STD=0`), `InitSDK()` no longer uses `std::` data types.

- **Expanded server SDK 5.x documentation**
  - Updated server SDK reference guides for C++, C#/Unity, and Unreal including expanded coverage of all data types.
    - Amazon GameLift server SDK 5.x reference for C# and Unity
    - Amazon GameLift server SDK 5.x reference for C++
    - Amazon GameLift Unreal Engine server SDK 5.x reference
  - New versions of the server SDK 5 integration guides for Unity and Unreal plugins
    - Integrate Amazon GameLift into a Unity project
    - Integrate Amazon GameLift into an Unreal Engine project
- **Additional documentation updates**
  - Revised documentation for Amazon GameLift service API operations GetComputeAccess and GetInstanceAccess to clarify remote access procedures based on the Amazon GameLift server SDK version in use.
  - Revised descriptions for GameSessionPlacement to document how game session information is transient when a placement is in "pending" status.

July 13, 2023: Amazon GameLift adds fleet hardware metrics

You can now track hardware performance metrics for your Amazon GameLift managed EC2 fleets. Metrics include EC2 instance metrics for CPU utilization, network traffic volume, and disk read/write activity. For Amazon GameLift, these metrics describe all active instances in a fleet location. You can view these fleet hardware metrics using an Amazon CloudWatch dashboard in the AWS Management Console. You can also view them in the Amazon GameLift console in fleet details.

Learn more:

- Monitor Amazon GameLift with Amazon CloudWatch (Metrics for fleets), *Amazon GameLift Developer Guide*

June 29, 2023: Amazon GameLift launches support for Amazon Linux 2023

Updated SDK versions: AWS SDK 1.11.111

Amazon GameLift customers can now use the Amazon Linux 2023 operating system to host their game servers. AL2023 offers several improvements over AL2 including security. This operating system is available in all AWS Regions with the exception of the China Regions.

Customers can use the newer Linux operating systems and continue to receive critical security updates when support ends for Amazon Linux (AL1) in December 2023. Support for Amazon Linux 2 continues through 2025.

Learn more:

- Amazon GameLift Linux Server FAQ
- Comparing Amazon Linux 2 and Amazon Linux 2023
- Amazon GameLift API Reference links:
  - AWS SDK action `CreateBuild`
  - CLI command `upload-build`
  - CLI command `create-build`

May 25, 2023: Amazon GameLift FleetIQ adds filter to exclude game session placements on draining instances

Updated SDK versions: AWS SDK 1.11.87

If you use Amazon GameLift FleetIQ for game hosting, you can now prevent game session placements on instances that are currently draining. Draining instances are flagged for shutdown, but they can still be selected to host new game sessions if no other hosting resources are available. With this new feature, you can exclude the use of draining instances entirely.

Use this feature when calling `ClaimGameServer` to find available game servers. Add the new `FilterOption` parameter and set allowed instance statuses to ACTIVE only. In response, Amazon GameLift FleetIQ looks only at active instances when searching for and claiming an available game server.

Learn more:

- ClaimGameServer in the *Amazon GameLift API Reference*

- How FleetIQ works in the *Amazon GameLift FleetIQ Developer Guide*

May 16, 2023: Amazon GameLift supports cost allocation tagging for fleets

Amazon GameLift customers can now use AWS Billing cost allocation tags to organize their game hosting costs. You can assign cost allocation tags to individual Amazon GameLift EC2 fleet resources to track how your fleets are contributing to the overall hosting costs.

Learn more:

- Manage your game hosting costs
- Using AWS cost allocation tags, *AWS Billing User Guide*

April 20, 2023: Amazon GameLift launches support for Windows Server 2016

Updated SDK versions: AWS SDK 1.11.63

Amazon GameLift customers can now use the Windows Server 2016 operating system to host their game servers. This operating system is available in all AWS Regions. Customers can use the newer Windows operating system and continue to receive critical security updates as Microsoft ends its support for Windows Server 2012 in October 2023.

Starting today, new customers who require a Windows runtime environment must specify Windows Server 2016 when creating new game server builds for hosting. Existing customers can continue to create new builds and fleets with Windows Server 2012 but must complete migration with Windows Server 2016 before the Microsoft end of support date on October 10, 2023.

This update includes the following service changes:

- When creating a game server build using Amazon GameLift SDK or CLI commands, you must now explicitly set the operating system. There is no longer a default value. To deploy your game server on Windows Server 2016, use the value `WINDOWS_2016`.
- When creating a game server build using the Amazon GameLift console, you must select an operating system from the available values. If you're an existing customer with active Windows Server 2012 fleets, you can choose either `WINDOWS_2012` or `WINDOWS_2016`.

Learn more:

- Amazon GameLift API Reference links:

- CLI command `upload-build`
- CLI command `create-build`
- AWS SDK action `CreateBuild`
- Amazon GameLift FAQ for Windows 2012

April 13, 2023: Amazon GameLift launches server SDK 5.x for Unreal

Updated SDK versions: Server SDK 5.0.0 for Unreal

The latest version of the Amazon GameLift lightweight plugin for Unreal Engine is now based on the Amazon GameLift server SDK 5.x. To start integrating your Unreal Engine environment with Amazon GameLift see the following links.

Learn more:

- Integrate Amazon GameLift into an Unreal Engine project
- Add Amazon GameLift to your game server
- Amazon GameLift server SDK 5.x reference for C++

March 14, 2023: Amazon GameLift launches a new console experience

The new Amazon GameLift console includes these improvements:

- **Improved navigation** – The new navigation pane facilitates navigation between Amazon GameLift resources.
- **Amazon GameLift landing page** – The new landing page provides links to helpful documentation, displays a high-level overview of Amazon GameLift, and provides support through links to documentation, frequently asked questions, and AWS re:Post.
- **Improved Amazon CloudWatch metrics** – Amazon GameLift metrics are now available in both the Amazon GameLift console and your CloudWatch dashboards. This update also includes new metrics for performance, utilization, and player sessions.

Learn more:

- Viewing your game data in the console
- Managing Amazon GameLift hosting resources

- Building a FlexMatch matchmaker

February 14, 2023: Amazon GameLift now supports server side encryption for Amazon SNS topics

Server Side Encryption ((SSE)) for SNS topics encrypts your sensitive data at rest. SSE uses AWS Key Management Service (AWS KMS) keys to protect the contents of your SNS topics.

Learn more:

- Set up event notification for game session placement
- FlexMatch matchmaking events
- Encryption at rest

February 9, 2023: Amazon GameLift server SDK supports .NET 6 with C#10

Updated SDK versions: Server SDK 5.0.0 for .NET 6. No SDK updates are required.

If you use the Unity Real-Time Development Platform, continue to use the Amazon GameLift server SDK 5.0.0 with .NET 4.6. Unity doesn't support .NET 6.

Learn more:

- Download the latest version of the Amazon GameLift server SDK at Amazon GameLift getting started
- Amazon GameLift server SDK 5.x reference for C# and Unity

January 31, 2023: Amazon GameLift server SDK supports the Go language

Updated SDK versions: Server SDK 5.0.0 for Go

Learn more:

- Download the latest version of the Amazon GameLift server SDK at Amazon GameLift getting started
- Amazon GameLift server SDK reference for Go

December 1, 2022: Amazon GameLift launches Amazon GameLift Anywhere and Amazon GameLift Server SDK 5.0

Updated SDK versions: AWS SDK 1.10.21, Server SDK 5.0.0 for C++ and C#

Amazon GameLift Anywhere uses your game server resources to host Amazon GameLift game servers. You can use Amazon GameLift Anywhere to integrate your own compute resources with Amazon GameLift managed EC2 compute to distribute your game servers across multiple compute types. You can also use Amazon GameLift Anywhere to iteratively test your game servers without uploading the build to Amazon GameLift for every iteration.

Highlights:

- New Amazon GameLift Anywhere fleet and compute types
- Amazon GameLift Anywhere compute resource registration
- Improved testing iteration cycle

Amazon GameLift Server SDK 5.0.0 introduces improvements to the existing server SDK and a new resource type, compute. Server SDK 5.0.0 supports Amazon GameLift Anywhere and the use of your own compute resources for game server hosting.

Learn more:

- Amazon GameLift server SDK reference
- Fleet location
- Choosing Amazon GameLift compute resources
- Create a Amazon GameLift Anywhere fleet

August 25, 2022: Amazon GameLift launches support for Local Zones

Updated SDK versions: AWS SDK 1.9.333

Amazon GameLift is now available in eight Local Zones in the United States, so you can deploy your fleets closer to players. You can use all managed Amazon GameLift features with Local Zones by adding the Local Zones to your fleets.

Local Zones extend AWS resources and services to the edge of the cloud, near large population, industry, and information technology (IT) centers. This means that you can deploy applications that require single-digit millisecond latency closer to end users or to on-premises data centers.

Learn more:

- Local Zones
- Fleet location
- Create a Amazon GameLift managed fleet

June 28, 2022: Amazon GameLift launches a new opt-in console experience

The new Amazon GameLift console includes these improvements:

- **Improved navigation** – The new navigation pane facilitates navigation between Amazon GameLift resources.
- **Amazon GameLift landing page** – The new landing page provides links to helpful documentation, displays a high-level overview of Amazon GameLift, and provides support through links to documentation, frequently asked questions, and AWS re:Post.
- **Improved Amazon CloudWatch metrics** – Amazon GameLift metrics are now available in both the Amazon GameLift console and your CloudWatch dashboards. This update also includes new metrics for performance, utilization, and player sessions.

Learn more:

- Viewing your game data in the console
- Managing Amazon GameLift hosting resources
- Building a FlexMatch matchmaker

February 15, 2022: FlexMatch adds compound rule and additional improvements

FlexMatch users now have access to the following features:

- **Compound rule** – Added support for compound matchmaking rules for matches of 40 or fewer players. You can now use logical statements to create a compound rule to form a match. Without a compound rule in your rule set, to form a match, all the rules in the rule set must be true. With

compound rules, you can choose which rules to apply using the following logical operators: `and`, `or`, `not`, and `xor`.

- **Flexible team selection** – Updated matchmaking property expressions to support selecting a subset of all available teams.
- **Longer string lists** – Increased the maximum number of strings from 10 to 100 in a list of strings of player attribute values.

Learn more:

- Amazon GameLift FlexMatch developer guide:
  - FlexMatch rule types
  - FlexMatch property expressions
- AttributeValue: SL

October 28, 2021: Amazon GameLift adds support for multi-Region fleets in the Asia Pacific (Osaka) Region; Amazon GameLift FleetIQ adds support for AWS Graviton2 processors

Updated SDK versions: AWS SDK 1.9.133

Amazon GameLift is now available in the Asia Pacific (Osaka) Region. Game developers can now deploy instances in Osaka using GameLift multi-Region fleet.

You can now use Graviton2-hosted game servers, based on the Arm-based processor architecture, to achieve increased performance at a lower cost when compared to the equivalent Intel-based compute options.

Highlights:

- Amazon GameLift is now available in the Asia Pacific (Osaka) Region.
- Amazon GameLift FleetIQ game server groups can now be configured to manage the Graviton2 instance families c6g, m6g, and r6g.

Learn more:

- Amazon GameLift multi-Region fleet

- CreateGameServerGroup
- AWS graviton processor

September 20, 2021: Amazon GameLift releases plugin for Unity

The Amazon GameLift plugin for Unity version 1.0.0 contains libraries and native UI that makes it easier to access Amazon GameLift resources and integrate Amazon GameLift into your Unity game. You can use the Amazon GameLift plugin for Unity to access Amazon GameLift APIs and deploy AWS CloudFormation templates for common gaming scenarios. The plugin also includes a sample game that works with the sample scenarios. You can use Amazon GameLift Local to see messages passed between the game client and the game server to learn how a typical game interacts with Amazon GameLift.

The plugin for Unity supports Unity 2019.4 LTS and 2020.3 LTS.

Highlights:

- Build, run, and modify a sample game with different scenarios, or create your own.
- Deploy sample AWS CloudFormation scenarios for typical game scenarios including auth only, single-Region fleet, multi-Region fleets with queue and custom matchmaker, Spot Fleets with queue and custom matchmaker, and FlexMatch.

Learn more:

- Integrating games with the Amazon GameLift plugin for Unity

June 30, 2021: FlexMatch adds batchDistance rule

You can use the batchDistance rule type to specify a string or numeric attribute, bringing a host of benefits to each segment.

Highlights:

- For large matches (>40 players), instead of evenly balancing players by skill only, you can now get that same balance based on skill, modes, and maps. Ensure that everyone in the match is in a skill band, band multiple numeric attributes such as league or play style, and group according to string attributes such as map or game mode. You can also create expansions over time. For

example, you can create an expansion to allow a greater skill level range to enter the match the longer the player is waiting.

For matches under 40 players, you can use a new simplified rules expression.

June 3, 2021: Amazon GameLift realtime client SDK and server SDK updates

Updated SDK versions: Realtime Client SDK 1.2.0, Server SDK 3.4.0 for Unreal

With this latest SDK update, you can now integrate IL2CPP into your mobile applications that use the RTS Client SDK and follow best practices with frameworks. You can also now build the Amazon GameLift Server SDK for Unreal Version 4.26. This update contains components that integrate with your Windows or Linux game server, including C++ and C# versions of the Amazon GameLift Server SDK, Amazon GameLift Local, and an Unreal Engine plugin.

Highlights:

- Added support for IL2CPP in the RTS Client SDK and for building the native libraries as frameworks, so you can build RTS clients for the latest mobile devices.
- You can use DescribePlayerSessions() to get information for a single player session, for all player sessions in a game session, or for all player sessions associated with a single player ID.
- You can use GetInstanceCertificate() to retrieve the file location of a PEM-encoded TLS certificate that is associated with the fleet and its instances.
- Created Server SDK support for Unreal version 4.26.
- The existing C# SDK, version 4.0.2, has been verified compatible with Unity 2020.3. No SDK updates were required.

Learn more:

- Amazon GameLift Developer Guide:
  - DescribePlayerSessions()
  - GetInstanceCertificate()

March 23, 2021: Amazon GameLift adds notifications to game session placement

Updated SDK versions: AWS SDK 1.8.168

You can now use events to monitor game session placement activity for a game session queue. Create an Amazon Simple Notification Service (Amazon SNS) topic to publish event notifications, or set up event tracking using CloudWatch Events.

Highlights:

- For each queue, you can set a custom text string to be included in all event messaging.
- When using an Amazon SNS topic, you can set additional access conditions that limit publishing to specific queues.

Learn more:

- Amazon GameLift Developer Guide:
  - Set up event notification for game session placement (new)
  - Game session placement events (new)
- API reference (AWS SDK)
  - New game session queue parameters `NotificationTarget` and `CustomEventData`: GameSessionQueue, CreateGameSessionQueue, UpdateGameSessionQueue
- Amazon GameLift forum

March 16, 2021: Amazon GameLift adds multi-region fleets, six new regions

Updated SDK versions: AWS SDK 1.8.163

Amazon 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 Amazon GameLift multi-location fleets feature, you can now set up a single fleet to host your game servers in any or all of 20 Amazon GameLift-supported Regions (Beijing Region excepted). This feature aims to significantly reduce the work required to set up and maintain Amazon 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 before this release are single-location fleets. Using multi-location fleets does not

affect your hosting costs. Amazon GameLift pricing is based on the type, location, and volume of instances that you use. (For more information, see Amazon GameLift pricing.) AWS CloudFormation support for multi-location fleets will be available soon.

> **Note**
>
> Multi-location fleets are not available in the China Regions. Amazon GameLift resources that reside in China Regions cannot interact with or be used by resources in other Amazon GameLift Regions.

Highlights:

- With a multi-location fleet, explicitly add a list of remote locations. Amazon 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 start by location, hosting cost, and player latency.
- Get hosting metrics in the Amazon 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 and filtering capability: CreateGameSessionQueue, DescribeGameSessionQueues, UpdateGameSessionQueue
  - Updated game session creation operations, with new location support: CreateGameSession, DescribeGameSessions, DescribeGameSessionDetails, SearchGameSessions

- Amazon GameLift Developer Guide:
  - Amazon GameLift hosting locations (updated)
  - Amazon GameLift fleet design guide (new)

    Scaling Amazon GameLift hosting capacity (updated)
  - Design a game session queue (new)
  - View fleet details (updated)
- Amazon GameLift forum

February 9, 2021: Amazon GameLift extends support for AMD instances, standalone FlexMatch

Updated SDK versions: AWS SDK 1.8.139

This release includes the following updates:

- Amazon GameLift FleetIQ game server groups can now be configured to manage the AMD instance families C5a, M5a, and R5a. The supported Amazon EC2 instance types, as listed for the GameServerGroup InstanceDefinition, now include the following:
  - c5a.large, c5a.xlarge, c5a.2xlarge, c5a.4xlarge, c5a.8xlarge, c5a.12xlarge, c5a.16xlarge, c5a.24xlarge
  - m5a.large, m5a.xlarge, m5a.2xlarge, m5a.4xlarge, m5a.8xlarge, m5a.12xlarge, m5a.16xlarge, m5a.24xlarge
  - r5a.large, r5a.xlarge, r5a.2xlarge, r5a.4xlarge, r5a.8xlarge, r5a.12xlarge, r5a.16xlarge, r5a.24xlarge

  Note: AMD instances for FleetIQ are currently not available for use in the China (Beijing) AWS Region. See Feature availability and implementation differences in China.
- Amazon GameLift managed game hosting now supports AMD instances in the China (Beijing) Region, operated by Sinnet. The new AMD instance families include M5a and R5a. Supported EC2 instance types, as listed for fleet InstanceType, now include the following:
  - m5a.large, m5a.xlarge, m5a.2xlarge, m5a.4xlarge, m5a.8xlarge, m5a.12xlarge, m5a.16xlarge, m5a.24xlarge
  - r5a.large, r5a.xlarge, r5a.2xlarge, r5a.4xlarge, r5a.8xlarge, r5a.12xlarge, r5a.16xlarge, r5a.24xlarge

- Amazon GameLift FlexMatch can now be used as a standalone matchmaking solution in the China (Beijing) Region, operated by Sinnet. Customers can create a FlexMatch matchmaker in the Beijing Region and configure the FlexMatchMode parameter to STANDALONE. For more information about FlexMatch, either with Amazon GameLift managed hosting or with a non-Amazon GameLift hosting solution, in the Amazon GameLift FlexMatch Developer Guide.
- When setting up event notifications for Amazon GameLift FlexMatch, you can now designate an Amazon SNS FIFO topic as the notification target. For more information, see:
  - MatchmakingConfiguration NotificationTarget, *Amazon GameLift API Reference*
  - Set up FlexMatch event notification , *Amazon GameLift FlexMatch Developer Guide*
  - Introducing Amazon SNS FIFO – First-in-first-out Pub/Sub messaging, *AWS News Blog*

December 22, 2020: Amazon GameLift server SDK supports Unreal Engine 4.25 and Unity 2020

Updated SDK versions: Amazon GameLift Server SDK 4.0.2, Unreal plugin version 3.3.3

The latest version of the Amazon 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 Amazon GameLift Server SDK at Amazon GameLift getting started.

November 24, 2020: Amazon GameLift FlexMatch now available for games hosted anywhere

Updated SDK versions: AWS SDK 1.8.95

Amazon GameLift FlexMatch is a customizable matchmaking service for multiplayer games. Initially designed for users of Amazon 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 primitive types. Games that use Amazon 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 you use FlexMatch with Amazon GameLift managed hosting or Realtime Servers, the matchmaker automatically uses Amazon GameLift to find hosting resources and start a new game session for newly formed matches. When using FlexMatch as a standalone service, the matchmaker delivers match results back to your game, which can then start a new game session using your hosting solution.

API operations for FlexMatch are part of the Amazon GameLift service API, which is included in the AWS SDK and the AWS Command Line Interface (AWS CLI). This release includes these updates to support standalone matchmaking:

- The API resource `MatchmakingConfiguration` has the following changes:
  - New property, `FlexMatchMode` indicates whether the matchmaker is being used with Amazon GameLift managed hosting or as standalone matchmaking.
  - Property `GameSessionQueueArns` is not required when `FlexMatchMode` is set to standalone.
  - These properties are not used with standalone matchmaking: `AdditionalPlayerCount`, `BackfillMode`, `GameProperties`, `GameSessionData`.
- The automatic backfill feature is not available with standalone matchmaking.

November 24, 2020: AMD instances now available on Amazon GameLift

Updated SDK versions: AWS SDK 1.8.95

The list of Amazon EC2 instance types supported by Amazon 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 Amazon 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
- Amazon GameLift forum

November 11, 2020: Version update to Amazon GameLift server SDK

Updated SDK versions: Amazon GameLift Server SDK 4.0.2

The new Server SDK version 4.0.2 fixes a known issue with the API operation `StartMatchBackfill()`. This operation now returns a correct response to a match backfill request.

The issue did not affect the match backfill process, and there is no change to how this feature works. The issue may have impacted log messaging and error handling for match backfill requests.

Download the latest version of the Amazon GameLift Server SDK at Amazon GameLift getting started.

November 5, 2020: New FlexMatch algorithm customizations

FlexMatch users can now adjust the following default behaviors for the matchmaking process. These customizations are set in a matchmaking rule set. There are no changes to the Amazon GameLift SDKs.

- Prioritize backfill tickets: You can choose to raise or lower how match backfill tickets are prioritized when searching for acceptable matches. Prioritizing backfill tickets is useful when the auto-backfill feature is enabled. Use the algorithm property `backfillPriority`.
- Pre-sort to optimize match consistency and efficiency: Configure your matchmaker to pre-sort the ticket pool before batching tickets for evaluation. By pre-sorting tickets based on key player attributes, your resulting matches tend to have players who are more similar in those attributes. You can also boost efficiency in the evaluation process by pre-sorting on the same attributes that are used in match rules. Use the algorithm property `sortByAttributes` with the `strategy` property set to "sorted".
- Adjust how expansion wait times are triggered: Choose between triggering expansions based on the age of the newest (default) or oldest ticket in an incomplete match. Triggering on the oldest ticket tends to complete matches faster, while triggering on the newest ticket leads to higher match quality. Use the algorithm property `expansionAgeSelection`.

September 17, 2020: Amazon GameLift updates server SDK

Updated SDK versions: Amazon GameLift Server SDK 4.0.1

The new Server SDK contains the following updates:

- C# API version 4.0.1
  - The API operation TerminateGameSession() is no longer supported. Replace with a call to ProcessEnding() to end both a game session and the server process.
  - A known issue with the operation GetInstanceCertificate() is fixed.
  - The operation GetTerminationTime() now returns a value of data type AwsDateTimeOutcome.
- C++ API version 3.4.1
  - The operation TerminateGameSession() is no longer supported. Replace it with a call to ProcessEnding() to end both a game session and the server process.
- Unreal Engine plugin version 3.3.2
  - The operation TerminateGameSession() is no longer supported. Replace it with a call to ProcessEnding() to end both a game session and the server process.
  - The callback operation `OnUpdateGameSession` is added to FProcessParameters to support match backfill.

Download the latest version of the Amazon GameLift Server SDK at Amazon GameLift getting started.

August 27, 2020: Amazon GameLift FleetIQ for game hosting with Amazon EC2 (general availability)

Updated SDK versions: AWS SDK 1.8.36

The Amazon GameLift FleetIQ solution for low-cost, cloud-based game hosting on Amazon EC2 is now generally available. Amazon 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 Amazon 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 Amazon GameLift FleetIQ solution:

- New API operation `DescribeGameServerInstances` returns information, including status, on all active instances for a Amazon GameLift FleetIQ game server group.
- New balancing strategy, `ON_DEMAND_ONLY`, configures a game server group to use On-Demand Instances only. You can update a game server group's balancing strategy at any time, making it possible to switch between using Spot Instances and On-Demand Instances as needed.
- The following preview elements have been dropped for general availability:
  - Use of custom sort keys for game server resources. Game servers can be sorted based on registration timestamp.
  - Tagging for game server resources.

April 16, 2020: Amazon GameLift updates server SDK for Unity and Unreal Engine

Updated SDK versions: Amazon GameLift Server SDK 4.0.0, Amazon GameLift Local 1.0.5

The latest version of the Amazon 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 versions 4.22, 4.23, and 4.24.

- Amazon GameLift Local version 1.0.5 updated to test integrations that use the C# server SDK version 4.0.0.

Download the latest version of the Amazon GameLift Server SDK at Amazon GameLift getting started.

April 2, 2020: Amazon GameLift FleetIQ available for game hosting on EC2 (public preview)

Updated SDK versions: AWS SDK 1.7.310

The Amazon 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 Amazon 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 Amazon GameLift FleetIQ

December 19, 2019: Improved AWS resource management for Amazon GameLift resources

Updated SDK versions: AWS SDK 1.7.249

You can now take advantage of AWS resource management tools with Amazon GameLift resources. In particular, all key Amazon GameLift resources—builds, scripts, fleets, game session queues, matchmaking configurations, and matchmaking rule sets—are now assigned Amazon Resource Name (ARN) values. A resource ARN provides a consistent identifier that is unique across all AWS Regions. They can be used to create resource-specific AWS Identity and Access Management (IAM) permissions policies. Resources are now assigned an ARN and also the pre-existing resource identifier, which is not Region-specific.

In addition, Amazon GameLift resources now support tagging. You can use tags to organize resources, create IAM permissions policies to manage access to groups of resources, customize AWS cost breakdowns, etc. When managing tags for Amazon GameLift resources, use the Amazon 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 AWS CloudFormation templates, updates in China (Beijing) Region

Updated SDK versions: AWS SDK 1.7.210

AWS CloudFormation templates for Amazon GameLift

Amazon GameLift resources can now be created and managed through AWS CloudFormation. The existing AWS CloudFormation build and fleet templates have been updated to align with the current resources, and new templates are now available for scripts, queues, matchmaking configurations, and matchmaking rule sets. 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*
- Manage resources using AWS CloudFormation in the *Amazon GameLift Developer Guide*

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.