



Architecture Diagrams Game Tech

Multiplayer Session-based Game Hosting on AWS



Multiplayer Session-based Game Hosting on AWS: Architecture Diagrams Game Tech

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

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

Table of Contents

Multiplayer game hosting **i**

Multiplayer Session-based Game Hosting on AWS Diagram 1

Further reading 2

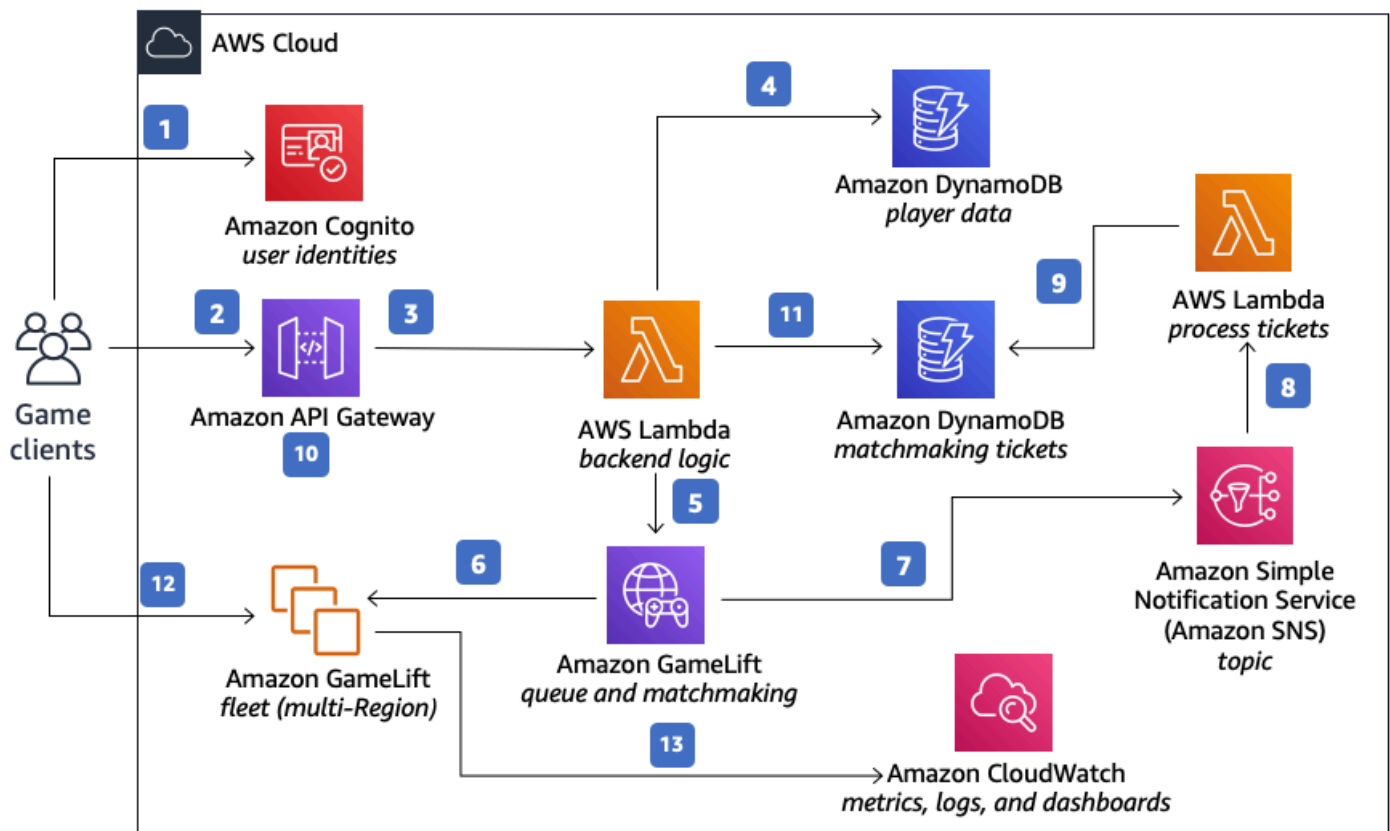
Diagram history 2

Multiplayer Session-based Game Hosting on AWS

Publication date: **September 1, 2022** ([Diagram history](#))

This architecture enables you to use Amazon GameLift Servers multi-Region fleets and a serverless backend solution to host a session-based multiplayer game.

Multiplayer Session-based Game Hosting on AWS Diagram



1. The game client requests an **Amazon Cognito** identity and temporary AWS credentials.
2. The client signs a matchmaking request to **API Gateway** with the temporary credentials. The request includes client latency information to supported AWS Regions.
3. **API Gateway** calls an **AWS Lambda** function with player identity information.
4. The **Lambda** function gets the player skill level from a **DynamoDB** table.
5. The **Lambda** function requests matchmaking from **Amazon GameLift Servers FlexMatch** with player skill and latency data.

6. **Amazon GameLift Servers FlexMatch** creates a match with multiple players, and an **Amazon GameLift Servers** queue allocates a session in an **Amazon GameLift Servers** fleet location based on the latency data.
7. **Amazon GameLift Servers FlexMatch** publishes an event to **Amazon SNS** on matchmaking success.
8. **Amazon SNS** triggers a subscribed **Lambda** function for ticket processing.
9. The **Lambda** function stores the ticket result in a **DynamoDB** table.
10. The game client polls for matchmaking success on a defined interval from **API Gateway**.
11. The **Lambda** function checks matchmaking information from the **DynamoDB** table and informs the client of a successful match by returning server IP, port, and player session ID.
12. The client connects directly to the server and sends the player session ID. The **Amazon GameLift Servers Server SDK** is used to validate the player session.
13. Game servers send logs and metrics to **Amazon CloudWatch** with the **CloudWatch** agent.

Further reading

For additional information, see the following resources:

- [AWS Architecture Icons](#)
- [AWS Architecture Center](#)
- [AWS Well-Architected](#)

Diagram history

To be notified about updates to this reference architecture diagram, subscribe to the RSS feed.

Change	Description	Date
Initial publication	Reference architecture diagram first published.	September 1, 2021

 Note

To subscribe to RSS updates, you must have an RSS plugin enabled for the browser you are using.