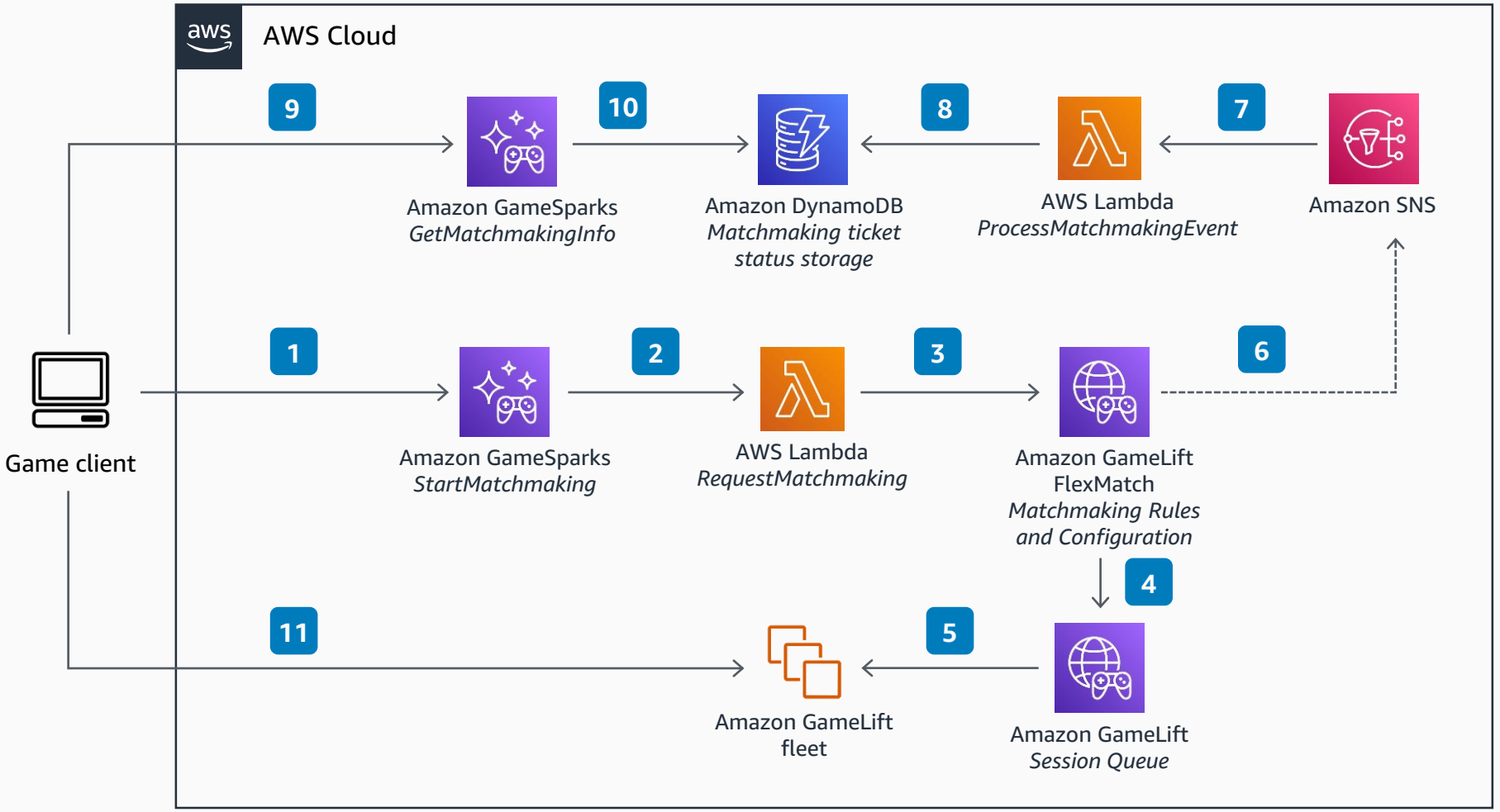
Guidance for Matchmaking with Amazon GameSparks

This architecture demonstrates how to use Amazon GameSparks and Amazon GameLift FlexMatch to perform matchmaking and connect a game client to a game hosted in GameLift.



1. Game client sends a StartMatchmaking request to **Amazon GameSparks**, containing client latency information, supported AWS Regions, and match preferences.
2. The **GameSparks** Cloud Code request handler for the StartMatchmaking request calls an **AWS Lambda** function.
3. The **Lambda** function requests matchmaking from **Amazon GameLift** FlexMatch with player and latency data.
4. **Amazon GameLift** FlexMatch creates a match with multiple players.
5. An **Amazon GameLift** queue allocates a session in an **Amazon GameLift** fleet.
6. **Amazon GameLift** FlexMatch publishes an event to **Amazon Simple Notification Service (Amazon SNS)** when the matchmaking ticket status changes.
7. **Amazon SNS** triggers a subscribed **Lambda** function for ticket processing.
8. The **Lambda** function stores the ticket status in an **Amazon DynamoDB** table.
9. The game client polls for matchmaking updates by sending requests to **GameSparks**.
10. The **GameSparks** Cloud Code request handler retrieves the ticket data from **DynamoDB** and informs the game client of a successful match by returning the server IP, port, and player session ID.
11. The game client connects directly to the game server and sends the player session ID. The **Amazon GameLift** server software development kit (SDK) is used to validate the player session.

Reviewed for technical accuracy March 7, 2023

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS Reference Architecture