E-Book

VOLKAN CIVELEK | INDUSTRY SOLUTIONS ARCHITECT @ REDIS

The Game Developer’s Guide to Matchmaking

Matchmaking with Redis Enterprise on AWS

redis

Table of Contents

Introduction

When you Google “screen time,” the majority of the results that come up are related to parenting and how to limit screen time for kids. But anxious parents aren’t the only people obsessing over screen time. Both the gaming and media industries think about screen time a lot—mainly, how to maximize the screen time of their active users. These companies are competing for end-users’ valuable attention. The more consumer screen time they win, the more revenue they earn. And the gaming industry has pulled ahead in this competition—its annual revenue surpasses that of the movie and music industries combined.

While elements such as gameplay, storyline, and design are all undoubtedly important elements of a successful game, user engagement is just as critical. Unlike consumable media such as movies and TV shows, gamers become active participants of the game experience, and they often are very social with other players on the gaming platform.

This increased engagement creates a snowball effect, as players discuss the game with their peers, create online buzz, and attract more players to join. And all this excitement leads to better player retention, more monthly active users (MAUs), a stronger gamer community, and increased revenue.

In the world of gaming, however, higher engagement requires real-time interactions. Real-time performance is what gamers expect—anything less will see them drop the game and move on to other games that can deliver it. Because real-time performance is so critical to gaming, Redis Enterprise and AWS have become the database and cloud platform of choice for game developers for such use cases as matchmaking, leaderboards, personalization, session management, and content caching.

Let’s dive deeper into one of those key real-time gaming use cases: matchmaking.

What is matchmaking?

Matchmaking is the process of connecting players for online play sessions. It is an important gaming element, as matchmaking can ensure that players are paired with others of a similar skill level, located in close proximity, or with similar interests.

Matchmaking is akin to leaderboards. Leaderboards track player progress and to encourage them to keep playing. Matchmaking can also encourage players to keep playing, as it provides them with a sense of accomplishment and the urge to compete against higher-ranked gamers.

Designing an effective matchmaking strategy, programmatically, requires developers to address several technical issues. Game developers strive to match players based on these three factors:

1. Latency

Latency and ping are the two terms players use to describe the gaming experience in performance terms. In multiplayer games, the geographic region from which the player joins the game is the primary factor that contributes to latency. That is because connecting to a server in a different region isn't ideal for online gaming. If a player in the United States connects to a server in Europe, the data sent between them travels a large distance. This takes time, causing a lot of delay and resulting in high ping—the last thing you want in gaming. Ensuring gamers are playing on geographically-close servers is crucial to improving latency.

2. Preference / skill level

Skill-based matchmaking (SBMM) has always been a hot topic among gamers. SBMM works by trying to match players together who are at a similar skill level, with the goal of keeping games fair and competitive. Game developers use techniques to periodically calculate the percentiles of players' skill levels to minimize wait times.

But the way skill level is determined differs from game to game. Games such as "Call of Duty" and "League of Legends" have skill rank queues. However, in other games, a player's level isn't necessarily a depiction of their skill but rather how much time they have invested in the game.

3. Wait time

The concept of matchmaking is not only in gaming but in dating. It's also known in the business world, such as in B2B matchmaking or investor matchmaking. There is one significant difference. In gaming, matchmaking needs to happen within seconds.

A short wait time is as important as compatibility when it comes to game matchmaking. Although it's dependent upon complex rules among many players, matchmaking must happen instantly.

It's not always possible for all three of these matchmaking factors to have equal priority. For example, if a player has been waiting longer than what the game deems reasonable, the other two factors (location and skill level) get less priority. This means a player may end up playing with people outside their skill level or on a distant server, which causes the game to lag.

Data requirements of matchmaking

For decades, the answer to "How do I manage data?" was "a relational database." However, a one-size-fits-all database doesn't fit anymore. Gaming developers need to select the best tool to solve the problem – and for this problem, relational databases are no longer the best answer.

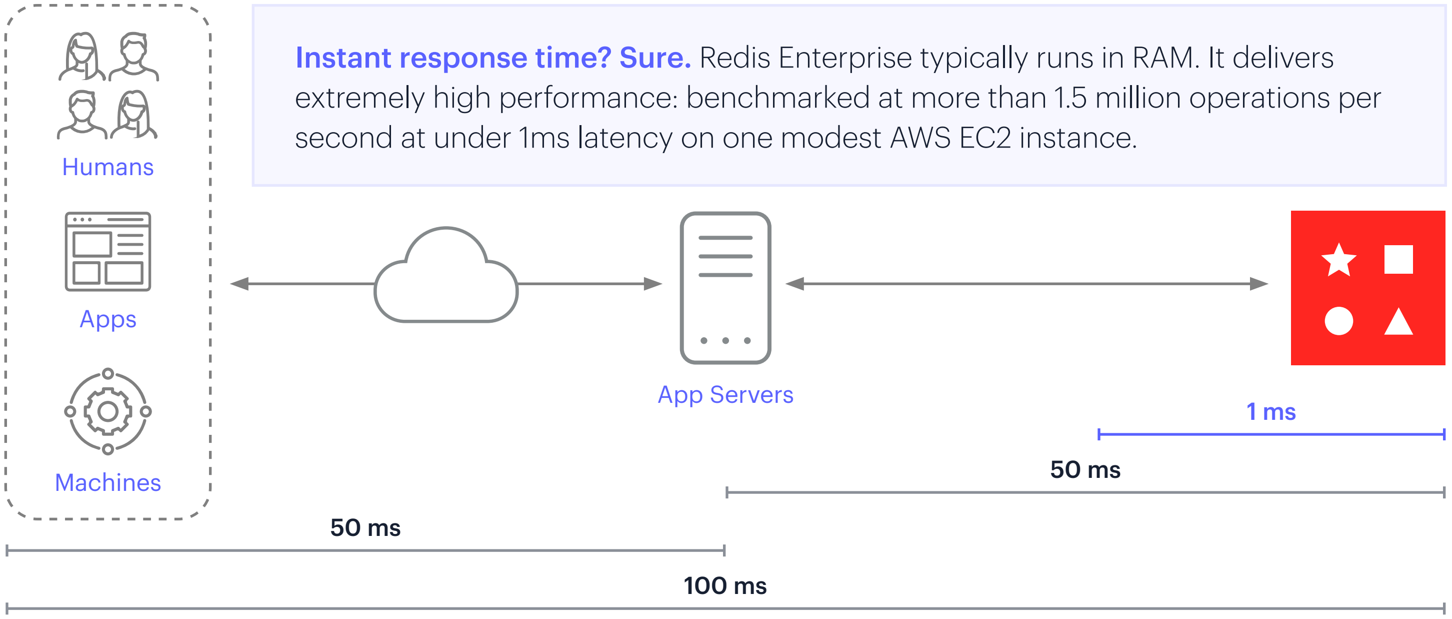
The data layer of a function needs to be diligently articulated from ground up so that it can serve up the data to its performance requirements such as real time and high availability. Matchmaking functionality has specific performance, scalability, and availability requirements. Failing on these requirements not only jeopardizes success of the matchmaking functionality but endangers the entire game itself. Let's visit the data requirements of matchmaking in more detail.

Real-time database

In order for humans to perceive an experience as instantaneous, it must occur in 100 milliseconds or less.

In a client/server world, data does not come right back to us directly from the database. Rather, it's processed and travels through servers. Network time itself can easily take up to 50ms and servers can take up another 50ms. This leaves the database response time somewhere between 0ms and 1ms in order to produce an experience perceived by gamers as instantaneous. So to drive down from the requirements, a real-time database should serve up the data in 1ms or less.

Instant response time? Sure. Redis Enterprise typically runs in RAM. It delivers extremely high performance: benchmarked at more than 1.5 million operations per second at under 1ms latency on one modest AWS EC2 instance.



Will it scale? Linearly.
Redis Enterprise is benchmarked at 50 million ops/second under 1 millisecond, in as little as 26 EC2 nodes.

High concurrency

In game matchmaking, a ticket usually means a join request to play a game.. Every submitted ticket is enriched with the player's data and graph relationships from the database. In addition to these queries, the player's data is constantly updated with scores, ranks, and percentiles.

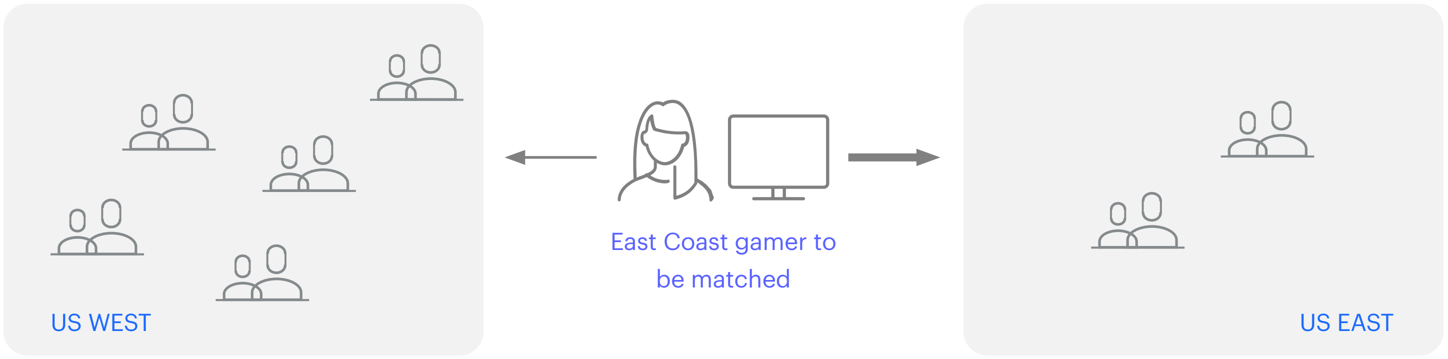
Join requests exponentially increase during peak hours. A database needs to be highly concurrent and conflict-free to support this level of read and write operations. Industry economics requires that this performance is delivered in the most cost-effective manner.

Play anywhere, at top speed
With Redis Enterprise Active-Active Geo-Distribution, you can have a database spread across multiple regions. All replicas are writable and all replicas are consistent via Conflict-Free Replicated Data Type (CRDT) technology. Active-Active provides 99.999% (five-nines) availability, which means about five minutes of downtime in a year.

Global availability

It's not enough for a gaming matchmaking service to be fast. It also needs to be reliably available whenever (and wherever) the players are online, which means the database also needs high availability while running on a global scale. Specifically, every region requires its own rules to optimize its own regional matchmaking.

As a Plan B to reduce gaming wait times, developers may choose to matchmake across geographical regions if the gamer's preferences allow for it. Larger player pools permit quicker matching and shorter wait times. In these circumstances, a conflict-free, geo-replicated database is a must.

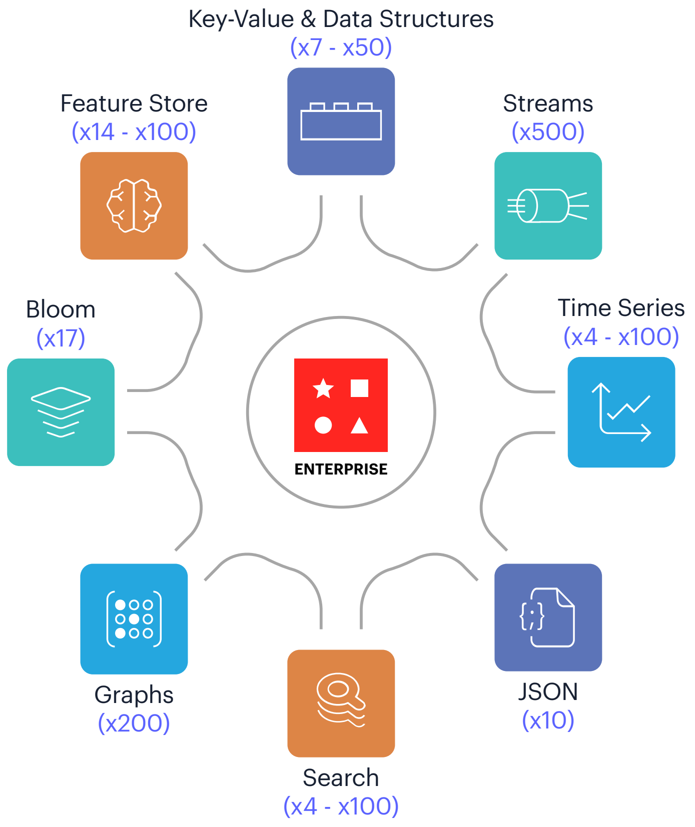


Right data model for the right use case

At its core, Redis supports popular data types, such as list, set, sorted set, and hashes. This is extremely useful to many features in gaming, such as leaderboards. Implementing leaderboards is as easy as issuing the ZINCRBY command to increment the score of any player—all the data kept ordered in memory by default, no full table scans required to retrieve it. Issuing a ZRANGE command will suffice to read the ordered data in O(log(N)+M) time, N being the number of elements in the sorted set and M the number of elements returned.

Rich data model support

There are cascading levels of technical complexity in game development. Key factors of complexity are the way players interact with each other, the hours they play, and any unexpected spikes in demand. Yet despite this complexity, it is vital to serve games at real-time speeds. All of this makes data modeling of utmost importance for gaming. Storing gamers' profiles, with all their interactions with the game itself and also with other gamers, requires rich data modeling so that the application can perform ultra-fast reads and writes.



On top of the native data structures, Redis Enterprise also supports:

- **RedisJSON** to store user profiles and access them fast.
- **RediSearch**, a module for querying, secondary indexing, and full-text search.
- **RedisTimeSeries** to add time series data structure to Redis.
- **Redis Streams**, to enable asynchronous, event-based data enrichment and queuing to Redis.
- **RedisGraph**, which is the full implementation of a property graph database with Redis.

And with every module, the performance difference against other common alternatives is measured in orders of magnitude.

Rich query language / search

Game matchmaking is based on a number of rules such as geo-proximity, matchmaking rating (MMR), skills percentile difference, and/or applying user preferences such as custom blacklists. All these rules require an efficient way to query the database in order to satisfy fast matchmaking.

Game matchmaking in action: a RediSearch example

The **RediSearch** module includes a query language that can perform text searches as well as complex structured queries on top of Redis Enterprise's set of commands.

Let's explore some of the search commands in action below. These commands are queries against the Redis Enterprise database that can be used in a matchmaking solution.

```
// Find the closest network edge POP to the player

FT.AGGREGATE cities * LOAD 2 city location
  APPLY geodistance(@location, 134.811767,51.6716705) AS dist
  SORTBY 2 @dist ASC LIMIT 0 1

// Find players on closest server

FT.SEARCH Game-x @pop:Miami LIMIT 0 4 RETURN 3 gamer_id mmr pop

// Find players on the closest server within +/- 2.5% of MMR

FT.SEARCH Game-x @pop:Miami @mmr:[2731 2872]

// Closest players within +/- 2.5% of MMR who are not blacklisted

FT.SEARCH Game-x @pop:Miami @mmr:[2731 2872]
  -@gamer_id:(2112|4343)

// Match people in my groups with a similar play style
…
~@group_tags:{thistle_community|olive_club}
~@play_style_tags:{med_mobile|sprayer}
```

Streams

We mentioned tickets, and defined them as a request to play a game. But where do we submit these requests?

There needs to be a mechanism when a gamer submits a ticket, and "matchmaker" is triggered. Since a player can submit a ticket anytime, this mechanism is called a stream.

For effective matchmaking, the stream needs to be:

- **Fast:** The stream's consumers should not fall behind the pace of the ticket production.
- **Reliable:** The stream consumers need to recover from failures. They cannot reprocess what has been processed, nor leave behind an unprocessed ticket. In other words, the code should consume the tickets exactly once.
- **Concurrent:** Gaming companies host multiple games and the popularity of these games differ. The streaming mechanism needs to effectively allocate streaming resources for all the available games and seamlessly scale when needed.

Streaming commands: An example

Redis Enterprise is well known for its speed. On top of that, with specific streaming commands such as **XACK** and **XAUTOCLAIM**, it's easy to develop a reliable, exactly-once consumer function. For concurrent stream processing, **XGROUP CREATECONSUMER** is the command to use.

Let's examine some of the streaming commands in action below:

```
// Create the stream and consumer groups

XGROUP CREATE tickets tickets-group $ MKSTREAM
XGROUP CREATECONSUMER tickets tickets-group worker-1
XGROUP CREATECONSUMER tickets tickets-group worker-2

// Almost always use * to indicate auto generation

XADD tickets 10-0 game_name "Game-X" gamer_id 1
XADD tickets 20-0 game_name "Game-Y" gamer_id 2
XADD tickets 30-0 game_name "Game-Z" gamer_id 3
```

```
//worker-1 reads the message 10-0 and acknowledges it

XREADGROUP GROUP tickets-group worker-1 COUNT 1 STREAMS tickets >
XACK tickets tickets-group 10-0

// Claim any messages that have been pending longer than 10 seconds

XAUTOCLAIM tickets tickets-group worker-2 10000 0-0

// Monitor the groups and the consumers

XINFO GROUPS tickets
XINFO CONSUMERS tickets tickets-group

// Redis keeps track of what the last delivered message to a stream group is
```

If you want to store JSON in a stream, you'll need to stringify it first like below:

```
const ticket = JSON.stringify({
  "game_name": "GAME-x",
  "gamer_id": "RANDINT",
  "network": {
    "ip": "1.2.3.4",
    "platform": "iOS",
    "ping": "5.4",
    "enabled": True
  },
  “location”:{
    “pop”: “Cary”,
    “coordinates”: “35.7915,78.7811”
  },
  “pref_override”: {
    “level”: “easy”,
    “max_wait”: 120,
    “blacklist_tags” : []
  }
})
```

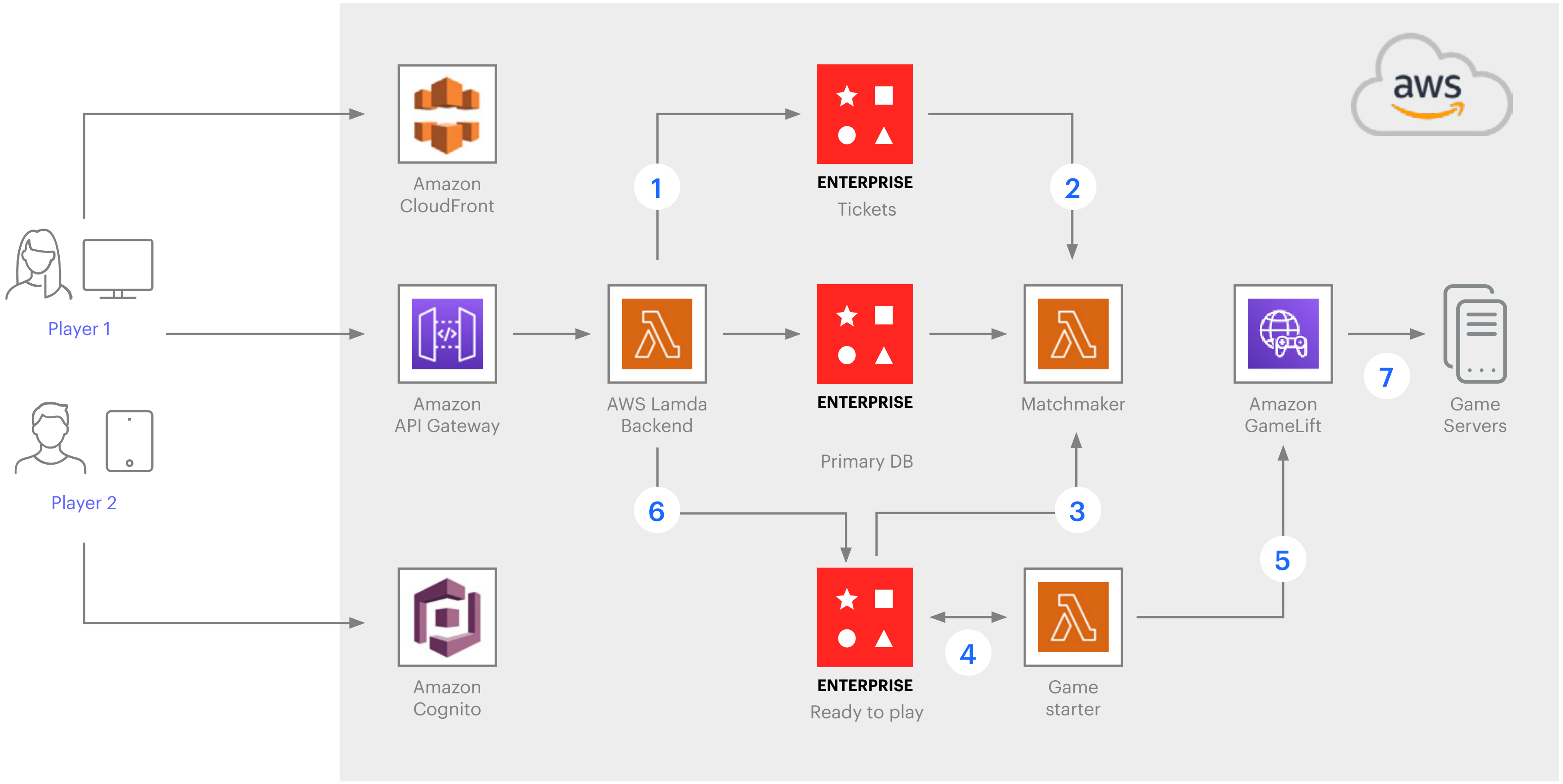
Then you can add it to the stream.

```
XADD tickets * json ticket
```

Integration with Amazon Web Services

Redis Enterprise meets gamers wherever they are, with support for multi-cloud and hybrid cloud deployment models, including on AWS.

The architecture below depicts the matchmaking functionality of a multiplayer game on AWS.



Amazon CloudFront is used as a global CDN to serve static content globally. In this example, Amazon Cognito is used for identity and Amazon API Gateway is the global load balancer to distribute traffic within a region and between regions. Once Amazon API Gateway invokes the AWS Lambda for a ticket request:

1. The AWS Lambda function acts as a producer. It adds the join request to the "Tickets" stream.
2. The matchmaker Lambda function continuously pulls the "Tickets" stream and enriches the join requests by pulling user data from the primary database.
3. The system establishes a match, based on the rules and user preferences. Then the matchmaker Lambda function adds a message to the "Ready to play" stream.
4. "Game starter" Lambda function consumes these messages and consults Amazon GameLift for a server assignment. Once it gets a response, the Lambda function updates the message with the provided session information.
5. As noted in step 4, the AWS Lambda function consults Amazon GameLift for a game server assignment.
6. Amazon GameLift reserves a game server and session for matched players.
7. After initial join request and player code starts periodically checking whether the match is ready. When it's ready, the player directly connects to the game server via the provided game server information such as IP, port, and custom player session ID.

All of these services provide pay-as-you-go pricing on AWS Marketplace, with no upfront costs or long-term commitment pricing. For customers who wish to leverage their AWS EDP, Redis Enterprise Cloud annual commitments are also available through the AWS Marketplace. The gaming companies enjoy Redis Enterprise on unified AWS billing and can consume credits towards their AWS contractual commit.

Game on

Game economics are changing. They are no longer single-player, offline puzzles. Most gaming companies are transitioning to a "games as a service" model, using a monthly subscription fee or microtransactions with in-game currency that creates stickiness for gamers.

Modern games have unique challenges, such as usage spikes when millions of players concurrently play and interact. In fact, gaming software company Unity has reported that 96% of multiplayer game developers believe scalability is critical to the success of a game. The popular game "Fortnite" has more than 350 million registered players, and it has been reported that on average 3 to 4 million people play the game, with spikes of up to 8 million concurrent players at certain times.

User acquisition costs are high, so it behooves gaming companies to do everything possible to keep players on the platform for as long as possible. This requires that games have to be engaging and fun whenever players join.

Accurate, compatible, and ultra-fast matchmaking is critical to player engagement and to any online video game's success. Game developers know this well, so they use the best tools to suit their needs, which is why Redis is so popular among game developers. The ability to handle a variety of data models at very high speeds for efficient matchmaking is what sets Redis Enterprise apart, and hosting on AWS easily and automatically prepares the game for unknown demand.

Etermax maximizes throughput for its game servers

With more than 10 million players for its Trivia Crack game, Etermax turned to Redis Enterprise to maximize its game servers' availability and throughput while reducing AWS infrastructure costs by 30%.

Read more >

MyTeam11 builds a low latency fantasy sports platform

MyTeam11's fantasy sports gaming platform has more than 15 million users. Using Redis Enterprise, the company easily handles traffic peaks when users set their fantasy sports rosters during the critical times when starting lineups are announced.

Read more >

Next steps

Learn more about the ways that Redis can improve gaming software development

- Build better games with real-time data
- Level up your Gametech with a Real-Time Database
- Pizza-Tribes - a demo game
- Fully managed Redis Enterprise Cloud on AWS
- AWS for Games

Why wait? Get started now with Redis Enterprise on AWS Marketplace.

About Redis

Data is the lifeline of every business, and Redis helps organizations reimagine how fast they can process, analyze, make predictions, and take action on the data they generate. Redis provides a competitive edge to any business by delivering open source and enterprise-grade data platforms to power applications that drive real-time experiences at any scale. Developers rely on Redis to build performance, scalability, reliability, and security into their applications.

Born in the cloud-native era, Redis uniquely enables users to unify data across multi-cloud, hybrid, and global applications to maximize business potential. Learn how Redis can give you this edge at redis.com