Is AWS ready to provide serverless WebSockets at scale?

medium.com/dazn-tech/aws-serverless-websockets-at-scale-8a79cd5a9f3b

AppSync vs WebSockets for API Gateway

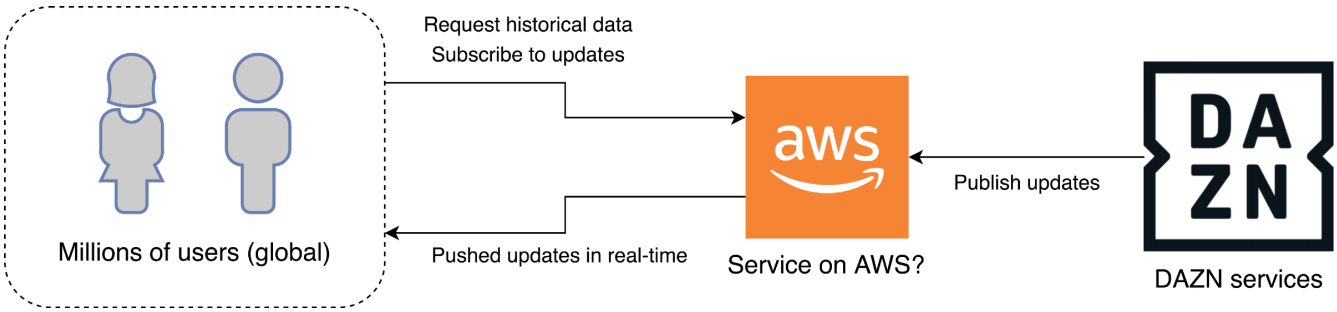
DAZN streams sports to millions of loyal fans worldwide. Our user base demand real-time updates on scores, goals and knockouts. The classic solution is to build a WebSockets service, creating a persistent connection between the user's browser and the server. But, if the server goes down the user has to reconnect. This carries risks of bringing a thundering herd of reconnections, crashing your servers down like dominoes.

Having dealt with such complexities around high-scale WebSockets services before, we wanted to try a different approach. Could we use AWS managed services to reduce our operational overhead? Could we take up Werner Vogels' offer to do our "undifferentiated heavy lifting"?

Our specific requirements

Let's start with our requirements for what the solution needs to do:

- Publish updates to users, rather than have the users poll
- Retrieve previous messages and subscribe to new messages
- Perform with high-traffic on a global scale



These requirements could easily apply to many organisations, the specific considerations for DAZN are outlined further below.

Publish updates to users, rather than have the users poll

We want all users to receive updates on key events without the need to poll—this could be the start of a live match, a goal or a red card. We want to avoid polling as it is inefficient for both servers and clients. Making

repeated HTTP requests can cause stuttering while playing live videos, especially on CPU-restricted devices such as TVs.

Retrieve previous messages and subscribe to new messages

We need to display a feed of key events including both historical events and live updates—for example a live feed of goals scored. The query for old messages is an ideal use-case for caching, but this carries the risk of race-conditions. For example if a match score changes whilst the query is cached, it risks the user missing the update—so, any solution must avoid this.

Perform with high-traffic on a global scale

The solution must scale with reasonable latency and regional failover. Millions of users come online just before a major sporting event kicks off, which leads to incredibly bursty traffic. So, it must be highly elastic and able to handle our traffic patterns. Of course, the costs at this scale need to be justifiable too.

With these requirements in mind, the two AWS products which look most suited to our needs are AWS AppSync and WebSockets for API Gateway.

Both services are relatively new—AppSync was released in April 2018 and WebSockets for API Gateway was announced at AWS re:Invent 2018 and only made available in December 2018.

AWS AppSync review

How it's sold

> AWS AppSync is a serverless back-end for mobile, web, and enterprise applications.

> Uses GraphQL, an API query language designed to build client applications by providing an intuitive and flexible syntax for describing their data requirement.

> Virtually unlimited throughput and storage, that scale according to your business needs.

> Enables real-time subscriptions across millions of devices

It couldn't sound any better! A flexible, serverless, massively-scalable, real-time, GraphQL-supporting managed service which doesn't cost the earth. So does it live up to the claims?

What's good about it

GraphQL is incredibly flexible and can easily cover many different use-cases.

Fully serverless solution with the ability to integrate with many other services.

Where it falls down

Limited authentication

Although multiple methods are available, you can only use a single authentication method at any given time.

Our front-end clients must be able to query data and subscribe for updates, but only our internal services should be able to publish data. Ideally, we would use API Key authentication for the front-end clients and a more-secure IAM authentication for our back-end services.

As AppSync only permits a single method, a work-around would be to use a Lambda authoriser. However, our bursty traffic would hit the hard-limits imposed on automatic scaling of 500 new concurrent executions per minute.

The only viable solution at this scale is to use API Key authentication, with one key for front-end and another for back-end. But then we'd need to hard-code those keys in the GraphQL resolvers to enforce access-control, which presents a significant security risk.

Requires two requests

Two requests are required to retrieve existing data and subscribe to updates. This has the potential to cause race-conditions — the client must subscribe *before* making the GET request to ensure the user doesn't miss

any messages. This also requires the GET request to return a strongly-consistent copy of the data, removing the viability of caching.

Works best with DynamoDB

DynamoDB has limited capacity per partition. Due to our need to retrieve existing data, each new client must send a query to AppSync—which could result in millions of requests to a single partition. This can't be cached due to the race-conditions mentioned above, so we'd hit the hard limit of 3,000 DynamoDB read-units per second for a single partition. This causes two issues—it could take 1m users at least 5 minutes to retrieve data, and it makes it too expensive to be a viable option.

To resolve this, we considered using DAX (DynamoDB Accelerator) for caching but AppSync doesn't support DAX. Additionally it wouldn't suit our needs of caching a query as it only supports invalidations for individual items—queries are cached separately with a fixed TTL.

The only workable solution is to duplicate the items and distribute them across multiple partitions, spreading the load more evenly. However, this introduces complexity in maintenance and it would still be prohibitively expensive.

WebSockets for API Gateway review

How it's sold

> Historically, building WebSocket APIs required setting up fleets of hosts that were responsible for managing the persistent connections […].

> Now, with API Gateway, this is no longer necessary. API Gateway handles the connections [and] lets you build your business logic using HTTP-based backends such as AWS Lambda, Amazon Kinesis, or any other HTTP endpoint.

In short, instead of terminating WebSocket connections on your hosts, API Gateway can do it for you which allows seamless use of services like Lambda.

Operationally, this sounds fantastic. Holding millions of connections open is the most difficult aspect of scaling WebSockets.

What's good about it

Highly versatile

It has the ability to send messages to each individual connection, making it suitable for many different applications.

Fully serverless

Despite being a lower-level solution compared with AppSync, WebSockets for API Gateway is also fully serverless with many integration options.

Where it falls down

Stored State

Connection metadata needs to be stored in a database, which we don't need for our application. This means that for every connection and disconnection, a Lambda needs to be run. With this limitation we'd hit Lambda's scaling limits. This might be avoidable by using an AWS Integration and have API Gateway call DynamoDB directly, but there would still be a huge amount of unnecessary database usage.

Broadcasting

There's no way to broadcast messages to all connected clients with one API call. In fact, you need to make an API call to AWS for each connection you want to send a message to. Publishing a single update to 1m clients requires fetching all 1m connection IDs out of the database and making 1m API calls, which is a deal-breaker.

New Connection Limits

There is a hard limit of 500 new connections per second per account per region. This means we'd only be able to connect up to 1.8m new connections per hour—another deal-breaker.

The verdict

AppSync provides flexibility through GraphQL, is fully serverless and can integrate with other applications. However, there are serious issues with using it at scale—limitations on authentication creates a security risk and the requirement for two requests prevents caching. Finally, using the preferred database DynamoDB presents further complexity and the solution is too expensive to use at scale.

WebSockets for API Gateway has huge potential through its versatility and could be suitable for a range of software applications. Unfortunately, at the moment it doesn't meet our requirements for scale or broadcasting to millions of clients.

What next?

It became clear that an off-the-shelf AWS solution won't be suitable for our needs—yet. Both AppSync and WebSockets for API Gateway are powerful products with a lot of potential, but they don't work for us right now. As both are fairly new to the market, we can expect lots of fixes and new features soon.

In the meantime, we decided to build our own solution using a combination of many other AWS managed services. Stay tuned for the next post on how we designed and built a custom solution capable of handling millions of users worldwide.