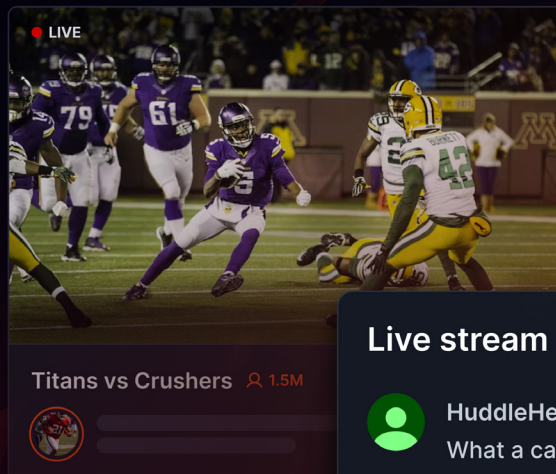


The definitive guide to building livestream chat at scale

Build reliable, cost-efficient chat, no matter how big your moment becomes.









```
import * as Ably from "ably";
import { ChatClient, ErrorCode } from "ably-chat";

async function main() {
  // Initialize Ably Realtime and Chat
  const realtime = new Ably.Realtime({
    key: "YOUR_API_KEY",
    clientId: "football",
  });

  const chat = new ChatClient(realtime);

  // Join a room
```

Live stream chat 👥 1.5M

-  **HuddleHero** 30s ago
What a catch, unreal! 🔥
🔥 52 🙌 42 👍 40 🔄
-  **BlitzBuff23** 28s ago
Comment deleted by admin 🗑️
-  **league_legend** MODERATOR 25s ago
Defense locked in 💪 🙌
👍 38 👍 28 🎉 18 🔄
-  **GridironGlory88** 25s ago
MVP vibes right there 🙌
👍 32 🙌 27 🔄
-  **QBWhisperer** 22s ago
Game of the year 🔥 🙌
🔥 26 👍 18 🔄
-  **Snap2Victory** 18s ago
That drive was unstoppable 🚀
🔥 18 🔄



You've invested in creating the perfect livestream - the content is polished, the production values are top-notch, and the audience is excited. But here's the thing: in today's digital world, livestreaming isn't just about the stream. It's about the shared experience.

When tens of thousands, or even millions, of viewers join your event, they don't just want to watch. They want to cheer, debate, react, and connect with each other in realtime. The chat is the experience. It's what transforms a passive audience into an active, engaged community.

And yet, delivering a truly great livestream chat at scale is deceptively hard.

How do you keep messages flowing smoothly when spikes hit - when the winning goal is scored, when the surprise announcement lands, when the moment everyone's been waiting for finally happens? How do you balance lightning-fast performance with moderation, security, and cost efficiency?

At Ably, we've worked with platforms delivering **500 million+ messages a day** to audiences across the globe. We've seen what works, and what can quietly break your user experience (or your budget) when scale hits.

In the following pages, we'll walk you through the architecture patterns, design decisions, and proven approaches that power dependable, cost-effective livestream chat for some of the world's biggest digital experiences.

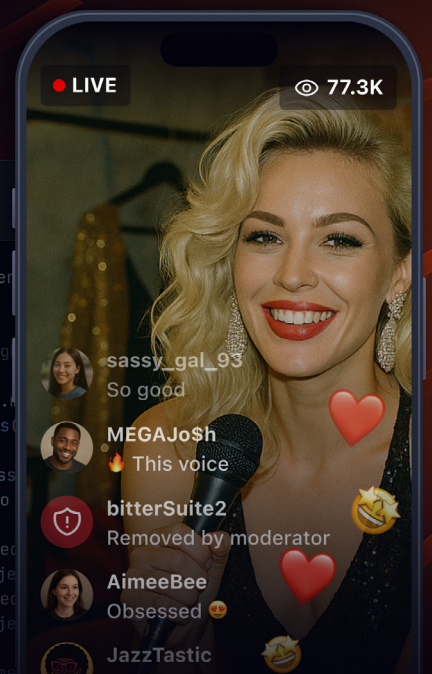
Whether you're building a global sports event, a live shopping app, a virtual concert, or a major conference - this is your playbook for getting it right.



```
Swift JS

print("\n [event.message.client

Try to send a message that mig
{
  let message = try await room.
    params: SendMessageParams(
)
  print("\n✅ Message sent: \(mess
catch let error as ARTErrorInfo
switch error.code {
case ErrorCode.messageRejected
  print("\n❌ Message was reje
case ErrorCode.messageRejected
  print("\n❌ Message was reje
default:
  print("\n❌ Error sending mes
```





01

The challenge: Livestream chat at scale isn't just about sending messages

At first glance, livestream chat might seem straightforward. After all, what's so hard about sending a message from one user to many others? You capture the text, push it out to everyone connected, and you're done, right?

If only it were that simple.

When you're dealing with hundreds of thousands or millions of users, all engaging, reacting, and messaging at once, everything changes. Every message your system handles has to meet demands that are invisible in smaller-scale apps, but absolutely critical when the spotlight is on:



Speed

Each message needs to appear instantly - no lag, no stutter - or you break the sense of connection that makes chat meaningful.



Reliability

Your chat can't buckle when a spike hits. That surprise goal or viral moment will drive a flood of messages, and users won't forgive delays or failures.



Efficiency

High-scale chat generates massive traffic, and that can mean massive cost. Without the right architecture, costs can spiral out of control, fast.



Integrity

Messages must arrive in the right order, with no duplicates, no missing data. At scale, this is harder than it sounds.



Safety

The bigger your audience, the higher the stakes for moderation. Keeping chat positive, inclusive, and compliant at scale isn't something you can leave to chance.



Scalability

Scaling shouldn't mean starting over. Chat should just work, right from your first thousand users up to your first million.

This is the reality behind the scenes of great livestream chat. It's not just about moving messages, it's about doing it at massive scale, with zero compromise on the experience.

And that's where the right architecture, and the right platform, make all the difference.



Key architectural decisions: Designing chat that scales

Scaling livestream chat isn't just about having the right infrastructure, it's about making smart architectural choices that align with your audience, your experience goals, and your cost constraints. The beauty of building on Ably is that it gives you the flexibility to design for your needs, without being boxed in by technical limitations.

Let's explore the key decisions you'll need to make, and how Ably supports each one.

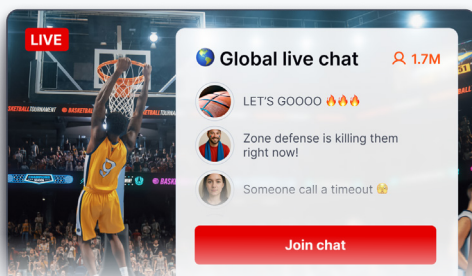
How will you structure your chat rooms?

One of the most important decisions in designing livestream chat is how to structure your rooms, because this shapes how your users experience the conversation.

Do you want everyone together in one massive room, sharing the same moment? Or would your users benefit from smaller, more focused groups where the pace and content of chat can be tailored?

Single global room

Perfect for major events where a unified conversation enhances the sense of community. Ably seamlessly supports millions of users in one room by distributing connections across multiple servers and regions, keeping performance smooth no matter how large the audience.

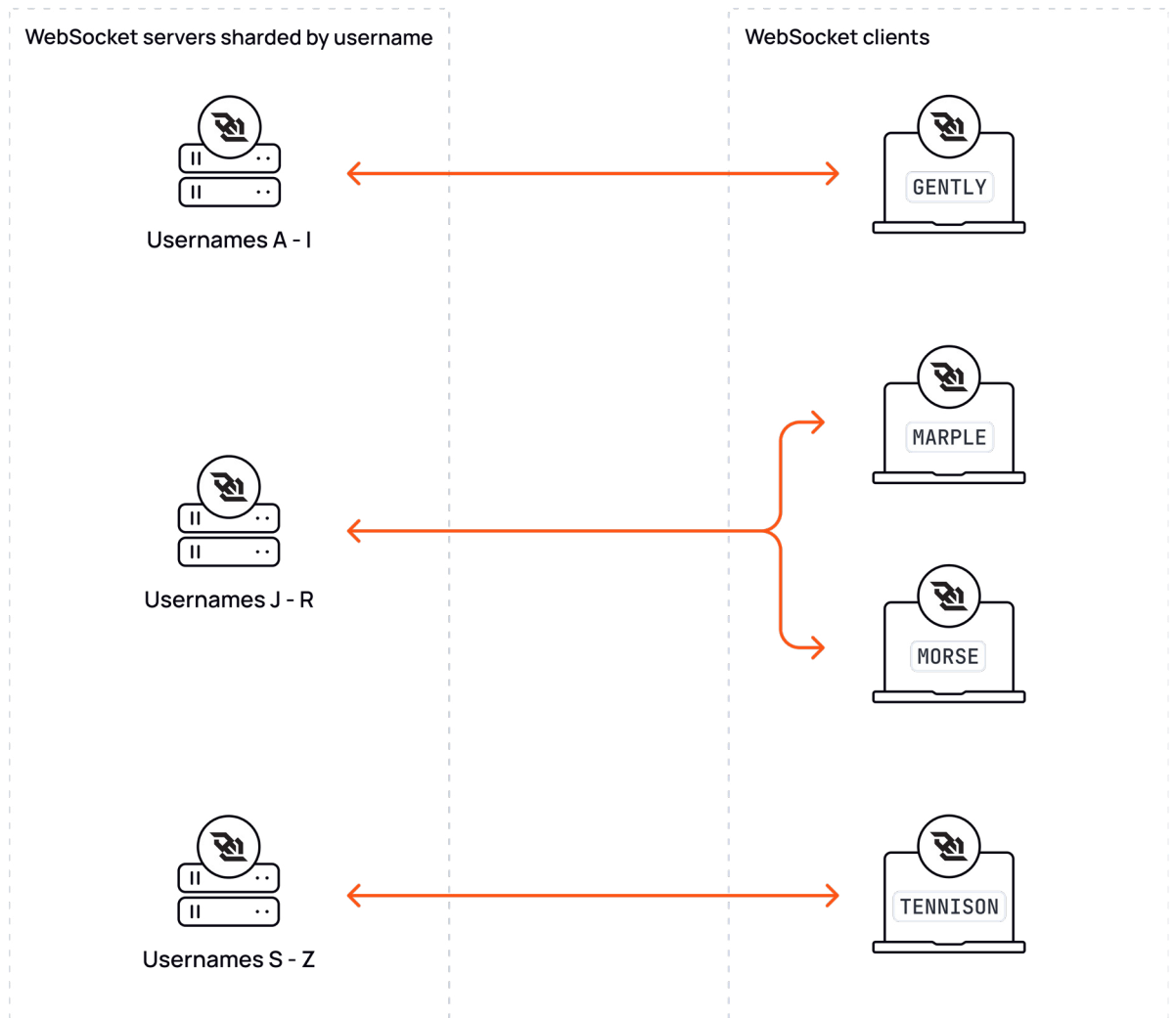


Segmented rooms

Ideal when you want to shape conversations by language, geography, or topic. Ably enables thousands of rooms to run concurrently, with horizontal scaling and consistent hashing to balance load evenly.



And behind the scenes, Aby takes care of intelligently sharding WebSocket connections, for example, by criteria like username or region, so you don't have to manually manage how connections are distributed.



Aby automatically shards WebSocket connections across servers, ensuring balanced load and smooth performance as your user base scales.

If you're supporting multiple languages or regions, you can segment rooms, or connect them to translation services using Aby's extensible architecture. The choice is yours!



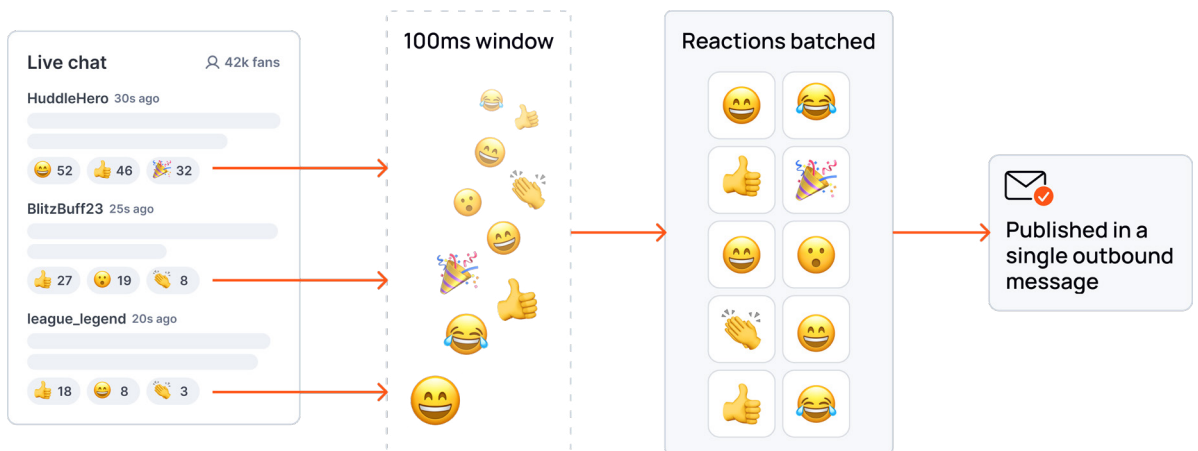
What message rate will deliver the best user experience?

The excitement of a livestream can generate incredible volumes of chat messages, and that's part of what makes these moments special. But too much, too fast, can overwhelm your users. A wall of unreadable messages breaks the connection you're trying to build.

Abyl gives you the tools to manage this elegantly, without dampening the energy of your chat.

- **Rate-limiting message publishing** ensures no one user can flood the chat.
- **Server-side batching** dramatically reduces outbound message fanout, keeping costs predictable and delivery smooth.

For example, Abyl can group reactions and messages sent in quick succession (say, within a 100ms window) and publish them in a single outbound message.



Reactions from thousands of users are batched together within a configured interval (e.g. 100ms or 500ms) and sent as one efficient outbound message, reducing fanout, network overhead, and cost.

For example, let's say you have 10,000 users, each sending one reaction per second. Without batching, that's 10,000 outbound messages per second, or 600,000 per minute. With a 500ms batching interval, you could reduce this to 20 batches per second. That's a 500x reduction in fanout.

Design for the average, plan for the spike. A chat that's easy to follow during normal moments, but ready to handle bursts without degradation gives your users the best of both worlds.

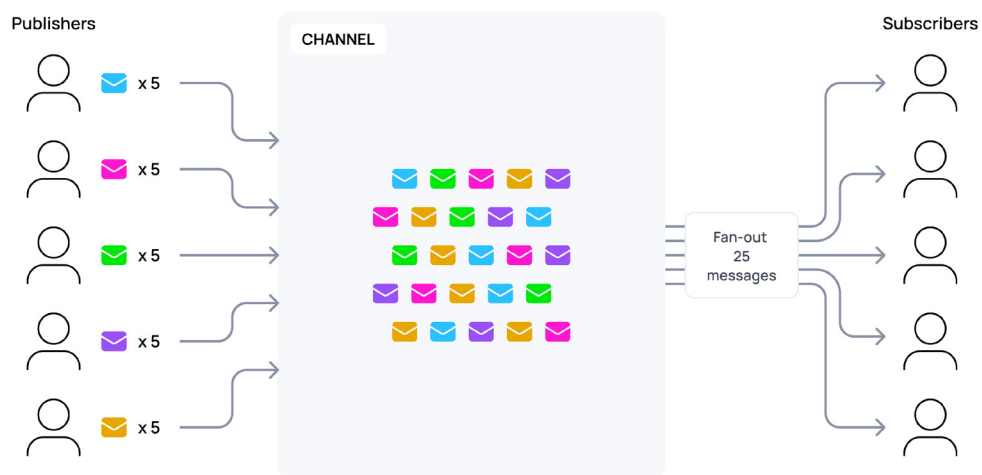


How will you balance cost and latency?

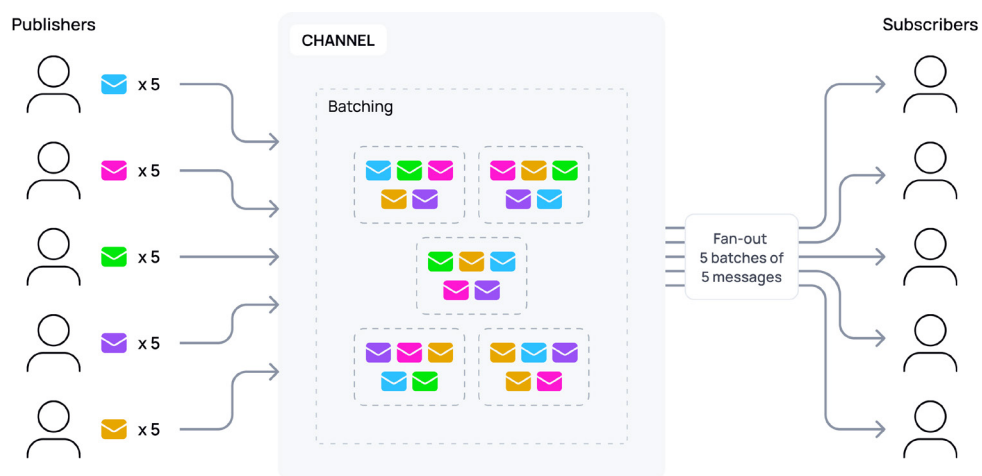
Every design decision shapes both the user experience and the cost of running chat at scale. With Aby, you can [fine-tune this balance without sacrificing quality](#).

Server-side batching is one of the most effective tools for reducing fanout. By grouping multiple messages sent in quick succession into a single outbound publish, you can dramatically lower network overhead while keeping chat responsive.

Without message batching



With message batching



For example, if 10,000 users each send one reaction per second, that's 10,000 outbound messages every second (600,000 per minute). With a 500 ms batching interval, those messages can be grouped into just 20 batches per second - a 500× reduction in fanout and associated costs.

\$ cost per fan per hour



Choosing the right batching interval

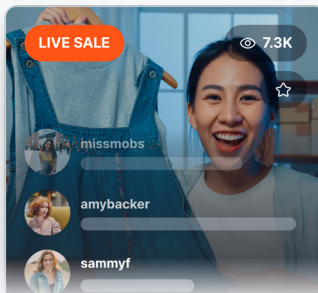
Large rooms

Up to 500 ms keeps chat feeling live while significantly cutting costs.



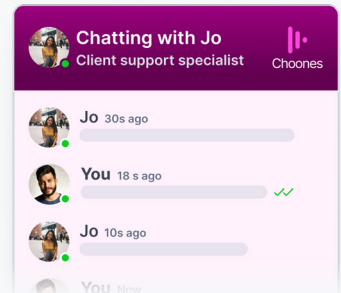
Medium rooms

Between 100–250 ms works well by balancing responsiveness with savings.



Small group/1:1 chat

You can skip batching for maximum immediacy.



Because batching reduces the number of outbound messages, it directly lowers the usage costs in [Aby's consumption-based pricing model](#).

Consumption-based pricing ensures you only pay for what you actually use, rather than for idle connections or over-provisioned infrastructure. Additional connection management features, such as heartbeat tuning and aggressive cleanup of idle connections, help avoid inflated bills from unused sessions.

At the end of the day, balance is key. Use batching and rate controls during peak traffic to control costs, but always design for the responsiveness your audience expects.



How will you keep chat safe?





At scale, moderation isn't optional. A single offensive or harmful message can derail the experience for thousands of viewers, damage brand trust, or even expose you to regulatory risk. The challenge is to keep the conversation safe without slowing down the realtime flow that makes livestream chat special.

First, choose your approach...

Decide whether you will:

- Use a supported moderation provider through Ably's direct integrations, or
- Build your own engine using a serverless function like AWS Lambda that Ably invokes during moderation.

Moderation providers supported by Ably, at a glance

| Provider | Mode support | What it's best at |
|---|----------------|--|
|  Hive (Model-only) | Before publish | Fast, automated text classification with configurable thresholds per class and severity. |
|  Hive (Dashboard) | After publish | Hybrid workflows that mix AI automation with human review in Hive's dashboard. |
|  Tisane | Before publish | NLU-driven text analysis with category thresholds you control. |
|  Bodyguard | Before publish | Contextual analysis tuned for toxicity, harassment, and similar policies. |

Bring your own engine

Prefer full control? Ably supports custom moderation flows via AWS Lambda. Run your own moderation logic before publish, define retry behavior, and control what happens if moderation fails (e.g. block or allow by default).

Whether you need strict controls, light-touch filtering, or custom workflows, Ably lets you build moderation into your chat without slowing it down, even at massive scale.

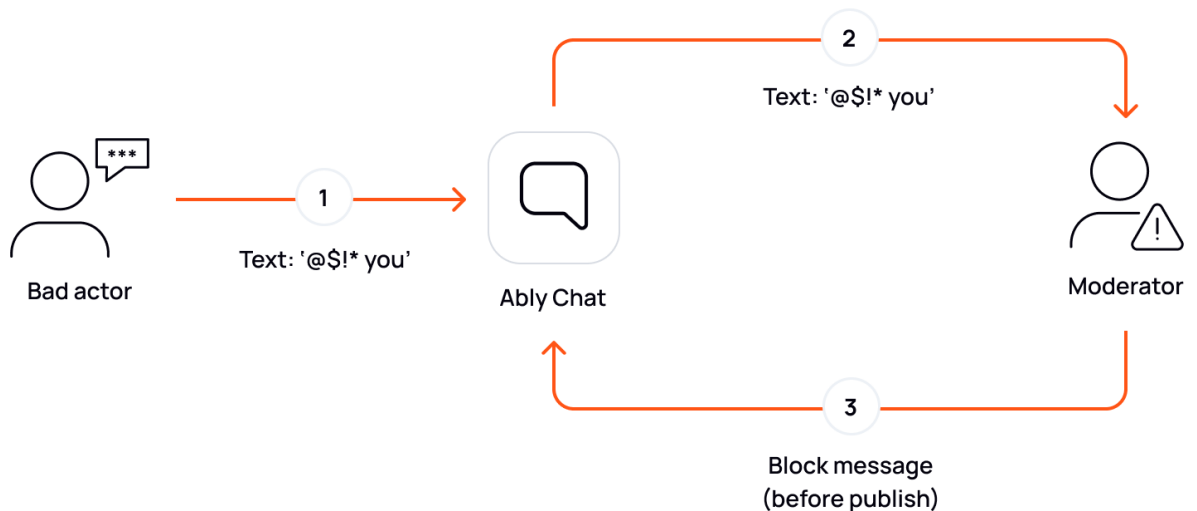
All moderation actions are billed as standard Ably messages and tracked in your usage stats.

Both paths support before-publish and after-publish strategies, so you can tune for safety vs speed per room.

Next, choose your moderation strategy

Before-publish moderation:

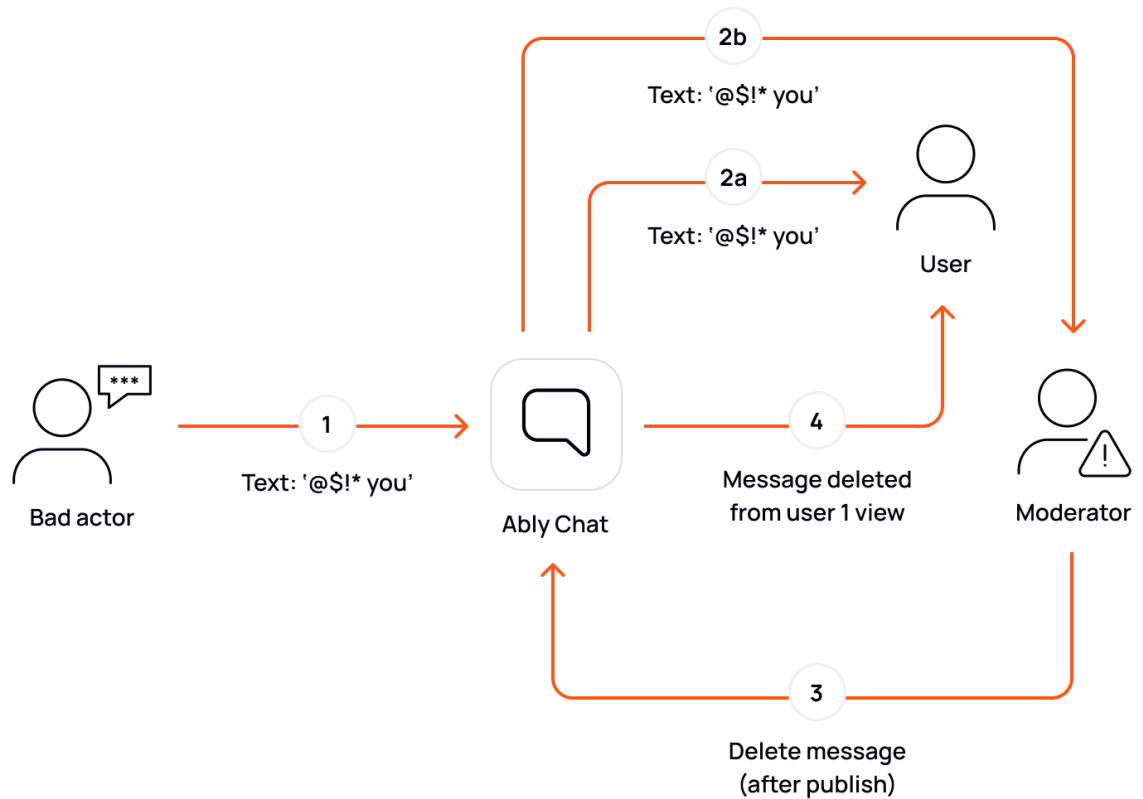
Messages are passed through a moderation engine before they're allowed into the chat room. This ensures inappropriate content is filtered before it ever reaches users. Ideal for sensitive use cases like education, finance, or tightly controlled streams where content must meet strict standards.



Messages flow through your moderation engine first. Only approved messages are delivered.

After-publish moderation:

Messages appear immediately, while moderation runs in parallel. If content is later flagged, it can be edited or deleted using Ably's [edit or delete methods](#). This mode has near-zero impact on latency and is well-suited to high-velocity chats like gaming or sports where speed matters most.



Messages are delivered instantly. Moderation decisions trigger edits or deletions as needed.

With Ably, you can mix and match: apply strict before-publish rules in one room, lightweight after-publish moderation in another, or bring in multiple providers. All moderation events are tracked as standard Ably messages, so usage and costs remain predictable.

Ultimately, the right moderation setup depends on your audience and goals. Some events need strict before-publish filtering, others benefit from the speed of after-publish, and certain use cases call for custom logic. Whether you use a supported provider or build your own engine, Ably lets you mix and match strategies across rooms so every chat gets the right balance of safety and speed.

How to authenticate and authorize users

Whether your chat is open to the world or tied to individual user identities, authentication and authorization are critical to protecting your platform, especially at scale. With Ably, the recommended approach is to use **token-based authentication** for all client connections. These short-lived tokens (such as JWTs) are issued server-side using your API keys, and provide time-limited, revocable access.

This design ensures your API keys stay safely on your server, where they can't be exposed or misused. Tokens can be tied to specific users (via client IDs), can carry exactly the permissions each client needs, and can be revoked or allowed to expire if necessary, giving you precise control without introducing unnecessary risk.

Why token auth at scale?

- Tokens expire automatically, limiting exposure if compromised.
- They support fine-grained permissions via Ably's capabilities system.
- You can revoke tokens at any time to cut off access.
- API keys never need to be shared outside your backend.

When designing your auth flow, follow the **principle of least privilege**: only give clients the capabilities they truly need. For example, a viewer might only be able to subscribe, while a moderator can publish messages and manage chat.

Best practices

- ✓ Use HTTPS for all token issuance and client connections.
- ✓ Set short token lifespans appropriate for your session length (typically minutes to hours).
- ✓ Monitor auth failures to detect anomalies or abuse.
- ✓ Simulate token expiry or revocation in testing to ensure graceful handling.

By putting these patterns in place, you can confidently scale your chat without sacrificing security, no matter how big your audience becomes.

Now that we've explored the architectural decisions involved in building livestream chat at scale - from structuring rooms to managing cost, safety, and performance, let's look at how Ably is purpose-built to help you execute on them, without the usual infrastructure burden.



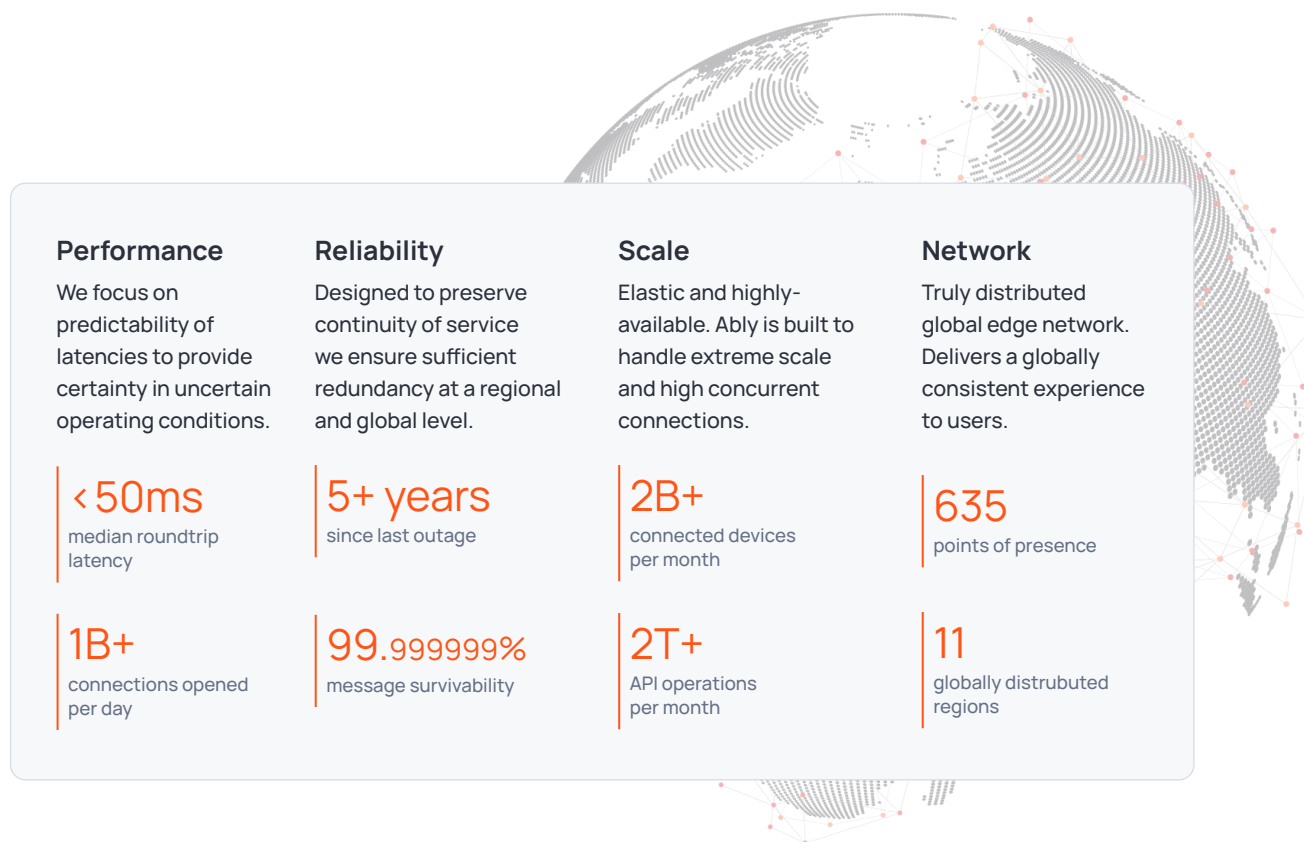
Aby's solution: Built for scale, designed for simplicity

When you're building livestream chat at scale, success hinges on more than just moving messages. You need to deliver them **instantly, reliably, and cost-effectively**, no matter how large your audience or how wild the spikes in activity.

That's exactly what Aby Chat was built for.

Our platform removes the operational headaches that come with scaling realtime infrastructure. No need to manage WebSocket servers, design for failover, or handle complex connection lifecycles, Aby takes care of it all.

At the heart of Aby is a **global edge network** spanning 7 core routing data centers and 600+ edge acceleration points of presence. This ensures your users connect to the nearest point for sub-50ms latency and maximum resilience.



When spikes hit, Aby doesn't flinch. The platform scales horizontally in real time, absorbing massive bursts of activity without manual intervention, so you can focus on delivering unforgettable moments, not fighting fires.



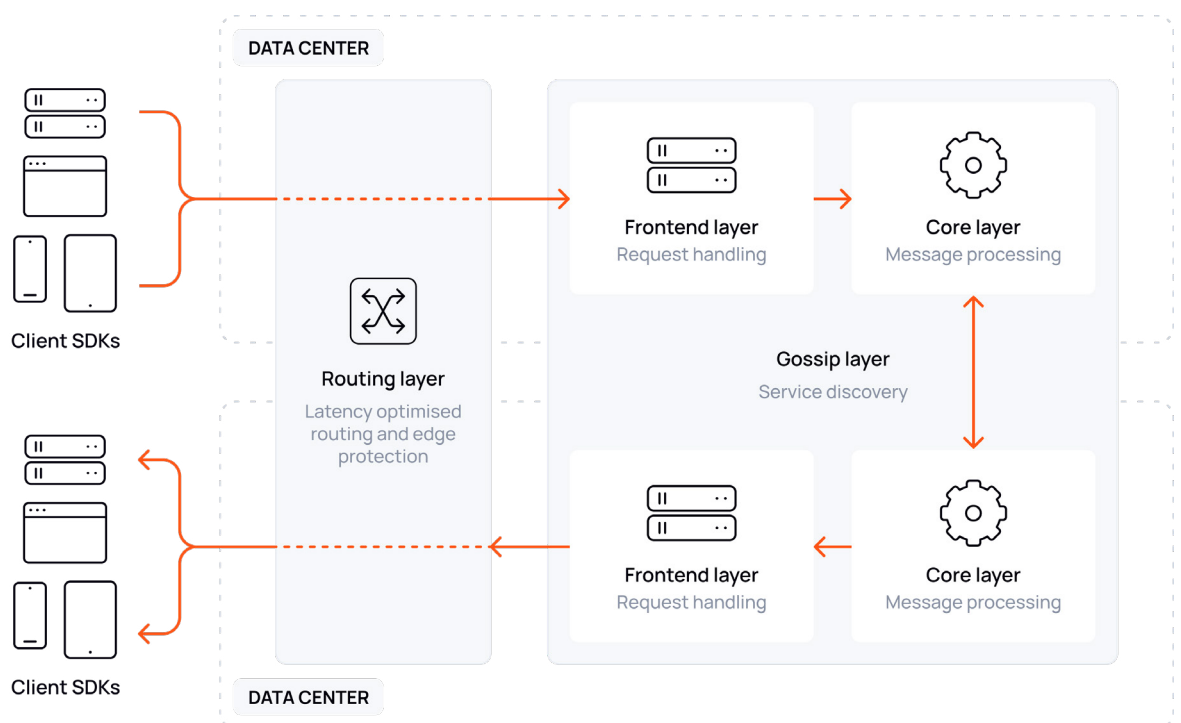
Aby's architecture: Engineered for scale, reliability, and speed

What powers this scale and simplicity is Aby's multi-layer distributed architecture, designed from the ground up to deliver dependable, low-latency messaging at massive scale.

Aby routes, processes, and delivers messages through multiple layers, including latency-optimized routing, load-balanced frontend request handling, and distributed core processing, with service discovery and failover handled automatically by the gossip layer.

Key layers that make the difference:

- ✓ **Routing layer:** Directs client connections to the closest data center and protects against edge failures for optimal latency.
- ✓ **Frontend layer:** Balances incoming connections and handles request processing at scale.
- ✓ **Core layer:** Ensures ordered, reliable message processing and delivery, even during traffic spikes.
- ✓ **Gossip layer:** Powers service discovery and failover so traffic can seamlessly shift if a server or region encounters an issue.



Aby's multi-layer architecture powers low-latency, resilient messaging at scale.

With Ably, you don't need to worry about scaling WebSocket servers, designing complex failover strategies, or handling connection recovery. Ably does it for you, so your chat just works, at any scale.

The result? Livestream chat that scales with your audience, and your ambition.

Ably powers chat for some of the world's most demanding digital experiences, from global sports events with millions of concurrent viewers, to virtual concerts, live shopping launches, and interactive conferences where chat is the main attraction.

- ✓ **Every message delivered with sub-50ms latency**, thanks to our global edge network and optimized routing.
- ✓ **Every conversation is coherent** - no duplicates, no gaps, no confusion, even at massive scale.
- ✓ **Seamless scaling** as your audience grows, with no manual provisioning required.
- ✓ **Resilience that's built-in**, with 99.999% uptime SLA, automatic failover across regions, and seamless reconnections.
- ✓ **Cost efficiency at any size**, with server-side batching, consumption-based pricing, and smart connection management keeping your bill under control.

This is what great livestream chat at scale looks like: always on, always fast, always ready, no matter how big your moment becomes.

Design decisions made simple

The beauty of building on Ably is that these architectural decisions are yours to make, but you're never alone. Our platform is designed to support the patterns that work at scale, and our team is here to help you choose and implement them.

"Ably is now a business-critical part of our organisation-wide infrastructure."

Max Friert
Product Group Lead

HubSpot

"With Ably, we delivered a high quality outcome in an impressive timeframe."

Andy Camera
Head of Technology (C2C)

sportsbet.com.au

"Partnering with Ably was a game-changer... ensuring a top-tier experience for our users."

Andy Tseng
Senior Engineering Director

i7LIVE



Your implementation checklist: Go live with confidence

Before your livestream chat goes live, it's worth pausing to make sure everything is in place, not just to handle the expected, but to stay rock-solid when the unexpected happens. This is your last line of defense against surprises during the big moment.

Here's what we recommend you verify:

- ✓ **Authentication and permissions:** Make sure you're using token authentication (such as JWTs) for all client communication, no hardcoded keys in your frontend. Check that client permissions follow the principle of least privilege: only the actions a client truly needs, nothing more.
- ✓ **Room architecture:** Confirm your chat room design fits your audience: whether that's one global room or many segmented rooms for regions, languages, or topics. If you're sharding, test that load is balanced evenly and that moderation policies apply as expected across rooms.
- ✓ **Throughput and cost configuration:** Validate your message batching intervals and rate limits. Are you balancing cost efficiency with the latency your users expect? Run load tests at expected and peak traffic levels, so you know how your setup behaves under stress.
- ✓ **Moderation:** Integrate and test your moderation tooling (whether pre-publish, post-publish, or hybrid). Simulate failure cases, for example, what happens if your moderation provider is unavailable?
- ✓ **Monitoring and metrics:** Set up dashboards to track occupancy (number of active users), message delivery rates, and error rates in realtime. Ensure alerting is configured so your team knows immediately if something isn't right.
- ✓ **Resilience and failover:** Check that your app handles reconnect scenarios gracefully, your clients should reconnect automatically, resume streams, and backfill history where needed. Review your failover setup to confirm that your chat will seamlessly reroute traffic if a region or data center goes down.

If in doubt, [our team is here to help](#) review your architecture or assist with load testing. The goal isn't just to go live, it's to stay confident that your chat will delight your users, no matter how big the moment becomes.

Power livestream experiences your audience will remember

Designing and operating livestream chat at scale doesn't have to mean wrestling with infrastructure, firefighting performance issues, or watching costs spiral out of control.

With Ably, you have a platform that's purpose-built to handle the complexity for you, so you can stay focused on creating experiences your users will remember. From the first line of code to your biggest livestream moments, we help you deliver chat that's fast, reliable, cost-efficient, and ready for whatever comes next.

Your audience is ready for the big moment. With Ably, so are you.

Ready to build?

- [Explore our developer documentation](#)
- [Try Ably for free](#)
- [Book a consultation](#)