Embrace Failure: Build a High-Availability Streaming Architecture

Authors:
Karin Schade – Senior Enterprise Architect, Global Consulting Services, Akamai Technologies
Jason Lane – Enterprise Architect, Global Consulting Services, Akamai Technologies
Akamai is no stranger to high-profile events. We have streamed Olympics, Super Bowls, FIFA, March Madness, and more to millions of viewers around the globe. Our mission is to provide the highest-quality video experience with zero downtime for all viewers.

On the other hand, the ever-increasing complexity in the process of preparing, publishing, and delivering live linear streaming increases the chance of unexpected errors and outages. Therefore, it is important to improve the resilience of each component inside the content distribution workflow and guarantee event uptime by anticipating unexpected issues.

In this article, you will learn best practices for designing a highly resilient delivery architecture, quickly bypassing SPOF, and readily applying multiple tested configurations to allow continuous, uninterrupted event streaming.

**Component Resilience**

In any streaming event, there are several components that must all stay up and perform well to continue delivery to your viewers. Read on to learn how to design a resilient architecture that is able to compensate for a failure at any one of these components.

**Ingest Architecture**

In every live stream architecture, it is imperative to have both a primary and backup stream. If one stream goes down, the player can switch to the backup stream and continue seamlessly.

The goal of the following recommendations is to ensure that primary and backup ingest are completely separate, so that any issue affecting one does not affect the other.

<table>
<thead>
<tr>
<th>Level of Resilience</th>
<th>Backup Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal/Recommended</td>
<td>Primary and backup streams ingested from different physical locations</td>
</tr>
<tr>
<td>Median</td>
<td>Primary and backup streams in same location, but on different networks</td>
</tr>
<tr>
<td>Minimal</td>
<td>Primary and backup streams in same location and network, but on different hardware</td>
</tr>
</tbody>
</table>

Any setup that does not meet the best practice above must implement a worst-case disaster recovery architecture. Should your single data center go down, you are clearly at risk of a total outage. If you are forced to utilize a single data center for ingest/publicing, it is quite likely you will need to utilize your disaster recovery setup.

For maximum performance and resilience, the ingest server should be as close as possible to the encoder. RTT has a significant impact on ingest performance, and it is imperative to optimize this with a short route.

<table>
<thead>
<tr>
<th>RTT</th>
<th>Ingest Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;20ms</td>
<td>Best Possible</td>
</tr>
<tr>
<td>20–40ms</td>
<td>Still good, no more than 40ms</td>
</tr>
<tr>
<td>41ms–80ms</td>
<td>Impact seen in stream latency</td>
</tr>
<tr>
<td>&gt;80ms</td>
<td>Definite impact to client rebuffering</td>
</tr>
</tbody>
</table>
If your encoder cannot be close to your ingest server, or your RTT is not within 20-40ms, you should be implementing Akamai Ingest Acceleration (IAS). Akamai IAS can optimize ingest and keep your ingest within the desired range.

**Example In Action**
A recent large sporting event required ingest from Russia and delivery to viewers in Mexico. In this case, the closest ingest server to the encoder in Russia was in Sweden. This issue was complicated further by the fact that it was a 4K experience being streamed. Without IAS, we experienced ingest interruptions that negatively impacted playback performance with a lot of rebuffering. With IAS accelerating the ingest of the 4K stream, the customer was able to deliver a great 4K experience for one of its largest events.

**A Note on Testing**
There are more and more components in each streaming event. CMS, ad stitching, encoding, sourcing, and more must be working together to deliver a successful event. Thorough, collaborative testing of the entire workflow is required to execute flawlessly.

<table>
<thead>
<tr>
<th>Testing Guidelines</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedule several end-to-end (E2E) tests with all vendors</td>
<td>Easier to identify ownership in case of any failures</td>
</tr>
<tr>
<td>Functional tests should be done on a smaller scale with collaboration between the customer and the Akamai account team</td>
<td>Individual components can be tested by smaller teams in preparation for E2E tests</td>
</tr>
<tr>
<td>Ensure client-side logging on all test devices/emulators</td>
<td>Charles logs and Wireshark</td>
</tr>
<tr>
<td>If possible, perform a load test against all components to ensure every link in the chain is scalable</td>
<td></td>
</tr>
</tbody>
</table>

*Always include DR Architecture in the same tests.*

**Mid-Tier Architecture**
The goal of any design for the mid tier of your streaming experience should be a 0% error rate sent to the client. This means removing all bottlenecks and always having a fallback for any component in the architecture.

**Cache Hierarchy (Cache-H)**
In any event, there will be a Cache Hierarchy in place. With a streaming event, we want to make sure that we can always get to the origin, so we want to have both a primary and backup Cache Hierarchy map. There should always be a fallback to the backup Cache Hierarchy map in case of failure to the primary.
Performance
If you have a long distance between the media server system and client, try to beat latency and Internet issues by using SureRoute between the caching layer and media server system.

Implementing prefetching is another great way to gain performance. Using prefetching allows the next segment to be put in the cache of the edge server before the client requests it. This must be thoroughly and carefully tested against a customer origin with prefetching against custom origin, because the additional prefetching load could overwhelm the origin.

Failover Strategies
If a connection is failing to origin server (or in the case of MSL4 Origin Shield, instead of multiple retries), stop after the first retry and do a failover to try a different route. This accomplishes two things:

- bypasses Internet connection issues
- bypasses single-server instance issues

Disaster Recovery (DR)
It is imperative to have a DR architecture completely set up and fully tested going into any event. Unforeseen Internet issues can and will happen. The more robustly designed the primary architecture, the less likely you are to need the DR architecture. The DR architecture must also be able to handle the same load as the primary, if it is to be able to recover any viewers.

It is not recommended to do automatic failover between primary and DR architecture. There are always cases that struggle to play back the stream, and that could be because of a bad client Internet connection or a suboptimal client player, etc. You don’t want these cases to failover to the DR system, because it will not solve the problems.

Furthermore, manual failover to DR architecture should be only used when you see that the primary system totally fails, such as when the primary and backup encoders are in the same location, the data center goes down, and it is not possible to bring the encoders back to live in a short time.
Try to identify the root cause of an outage or issues. Don’t failover when you see, for example, a large ISP has an outage. Even if you fall back to DR, you will have the same issues as with the primary.

If you do automatic failover to a backup recovery system, you will have more stress with the monitoring because you have to support and monitor both systems at once. Instead, reduce your monitoring resources, and focus.

It is advised to not implement a server-side failover on segment level. To get that working, both streams from primary and backup systems need to be 100% in sync, which is almost impossible.

The preferred option is to leave it to the customer to update the video CMS system with the stream URL that is given to the player.

There are some options that may be implemented by your Professional Services team through Akamai Professional Services.

**Human Factor**

When supporting a high-profile event, it is important to let cooler heads prevail. Don’t panic if an issue occurs. Try to stay objective and analyze the issue to come up with valuable solutions. Many times, Akamai has solved an issue by adding workarounds into configurations — sometimes in less than an hour. That only works when you stay calm.

It is important for the whole team that you stay positive and expect the unexpected. Events are long running and can get quite intense in a short amount of time. It is important to prepare for the worst, with the hope that you don’t have to use it, because you designed the primary architecture as a bulletproof system.

Finally, we all make mistakes — so use it to your advantage. Mistakes happen, but it is important to use them as a chance to learn and improve things for the next time.
Conclusion

Your live event can be successful, but it does not happen by accident. Thorough planning, designing your architecture to accept and handle failure, and collaboration between teams are all critical to the flawless execution of your strategy. You can go into your event with confidence knowing that if someone unplugs a router at your primary data center, you’ve planned for and successfully tested this exact scenario, and you have a plan in place to failover to keep the event up and running.

Contact us at consulting@akamai.com to explore how you can achieve a high QoE and create immersive, innovative, and differentiating experiences.

About the Authors

Karin Schade
Karin has 13 years of experience as a software developer, IT project manager, and IT architect. Before Akamai, Karin worked in various roles in Germany on multimillion-dollar budget software projects, and developed and designed software in different computer languages and frameworks. At Akamai she specializes in digital media and is working on the biggest streaming events. Karin is a German engineer that likes to be challenged with complex requirements or problems.

Jason Lane
Jason is an enterprise architect at Akamai, focused on successfully executing some of the world's largest streaming events. During his time at Akamai, Jason has worked on several record-breaking streaming events, was responsible for assessing risk and performance of Site and Stream for those events, and has designed and implemented solutions to address these topics.