Building a Video Annotation Platform Part 2

How to provide the scale video requires

//ALEGION

Building a Video Annotation Platform Part 2

How we built the expertise we needed to identify and solve for key problems with video annotation (VA)

In Part 1 of this series, we talked about the explosion of use cases for video in the computer vision (CV) space, some of the big challenges of video annotation, and our initial quest to build a super-responsive user experience for video annotation (VA), one that operates across numerous annotators with potentially unstable internet connections and scales on the back-end.

About Alegion : We provide an open training data platform to power enterprise machine learning (ML) initiatives. Our offering operates at massive scale, combining a data and task management platform with a global network of trained data labeling specialists to handle use cases ranging from the straightforward to the highly complex.

Part 2 Outline:

- **Design Goals :** 7 ambitious design goals for a better video annotation experience
- **How We Got "Generations Ahead" of the VA Competition :** The importance of market feedback
- **Building Expertise through Experimentation :** Benefits of a naive implementation
- **From Problems to Solutions and Results :** How we solved for front-end and back-end problems

Our design goals were clear

We wanted to create a solution that enabled:

- **Long videos -** minimum of an hour in duration
- **High resolution video -** full smooth streaming of videos with a minimum resolution of 4K or more
- **Large number of annotations -** 500k minimum
- **Responsive UI -** no perceptible degradation regardless of video length or annotation count
- **Efficient UI -** great U/X even with thousands of objects
- **No video pre-processing -** support what the browser supports while detecting and managing underlying video issues
- **Scalability -** support over 1,000 simultaneous annotators without performance degradation

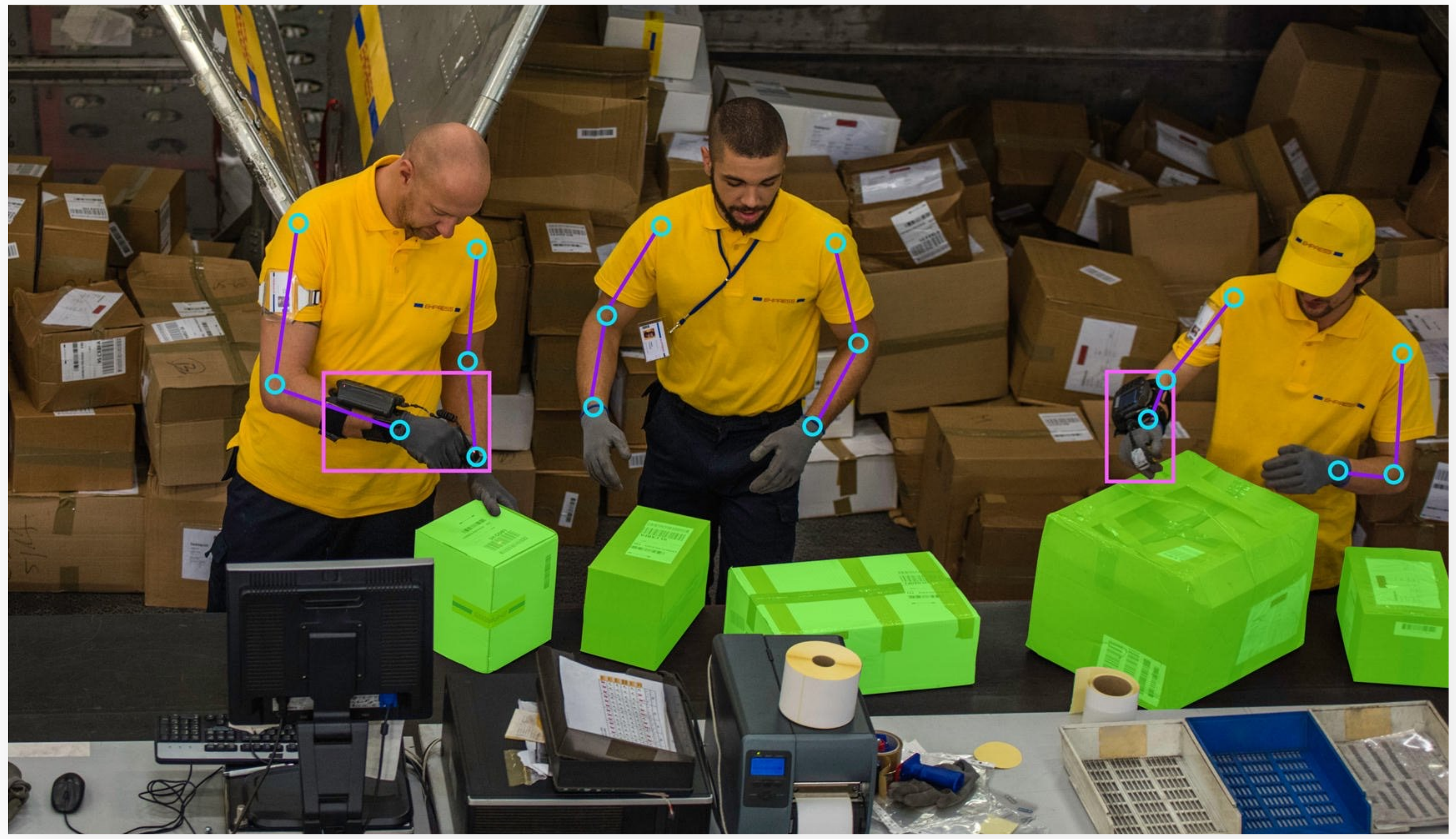
Solving for these design goals meant creating solutions on both the front-end and back-end that flawlessly scaled regardless of the annotation content or load on the system, saving customers time and money.

How we got “generations ahead” of the VA competition

We launched Alegion Control, our self-service video annotation offering in November 2020 to great acclaim. One partner described Control as “generations ahead” of the competition. Our ability to produce solutions that advance the field is the result of Alegion’s company culture of experimentation and our commitment to solve for our customers, both for their present needs and in anticipation of future needs.

The launch of Alegion Control was informed by over a year of learnings built upon real world customer engagements. While we have an amazing engineering team working on the product, it's critical that they also had the leeway to experiment and iterate. This was only possible because of our great collaborative relationships, both internally and with our customers. The empathy, coordination, and patience across Alegion’s Engineering, Product, and Customer Success teams allowed Alegion’s development teams to create the best-in-market solution we offer today, while continuing to deliver success for ongoing projects.

We knew that if we could meet our design goals for VA, the benefits would be tremendous for all of our customers, across use cases and verticals. **Let’s jump in and explore how we created a video solution that has the flexibility and sophistication to handle the most difficult use cases.**



Building expertise through experimentation

Like any good startup, we optimize our efforts around learning first and then using those lessons to guide our implementation of new capabilities. As we saw interest rise in video annotation, we obviously wanted to start saying **"Yes!"** to this growing stream of business, but at that time we didn't support video as a data type in the platform.

As a data labeling company with both a self-service, traditional SaaS offering, and full-service offerings where Alegion handles all aspects of the process, we have the opportunity to start delivering new solutions without having built out full platform capabilities. This “full service” capability gives us flexibility to experiment and time to learn.

Benefits of A Naive Implementation

In reality, tooling and infrastructure are just two pieces of the puzzle of delivering ultra-high quality annotations. You need to build expertise, and the best way to do that is through a naive implementation.

We moved to quickly stand up open-source tooling outside of our core platform and started delivering high-quality production work. Our platform already handled workflows and workforces; we needed to build a naive video implementation inside of the platform that was just good enough, incorporating our learnings from our open-source experience, to let us see how difficult scalability was going to be. The focus for this initial implementation was to be able to handle slightly more complex annotation work while giving us exposure (time) to learn from a varied set of new use cases (broader experience).

The naive implementation...was naive. So much new pain. We saw the number of classes, classifications, and the length of videos soar. We encountered use cases involving multi-hour surgical procedures with a handful of classes as well as five-minute aerial monitoring videos with more than 1000 objects, each requiring multiple sub-classifications.

From problems to solutions and results - the front-end

UX/UI and high speed rendering

Problem:

Our naive implementation involved overlaying SVGs on top of an HTML5 video element. This approach made development easy but performance slowed down with lots of annotations simultaneously on the screen at once. During use the painting of the annotations lagged the video, creating a jerky playback and scrubbing experience. This was especially fatiguing for quality control (QC) work or reviewing an annotated video.

Solution:

The basic explanation is that we experimented with different JS libraries. We clearly needed a new approach so we investigated three different options, ultimately selecting one that gave us WebGL performance, played nice with our React/Redux front-end, used a familiar syntax (TSX), and supported our preferred approach to testing.

Result:

In our final iteration, regardless of the count (tested to 1K), we were always able to paint all annotations in under 10ms. This allowed us to have a very smooth playback experience up to 100fps. This helped us meet design goals of scalability, and a responsive and efficient UI.

In addition to looking and feeling great, the annotation experience needs to be the same regardless of the complexity of the use case. Alegion's Head of Design covered much of the design process involved in the annotation U/X in a **separate article**. One of the biggest challenges here is to create a smooth playback and review experience.

From problems to solutions and results - the front-end

Memory management

Problem:

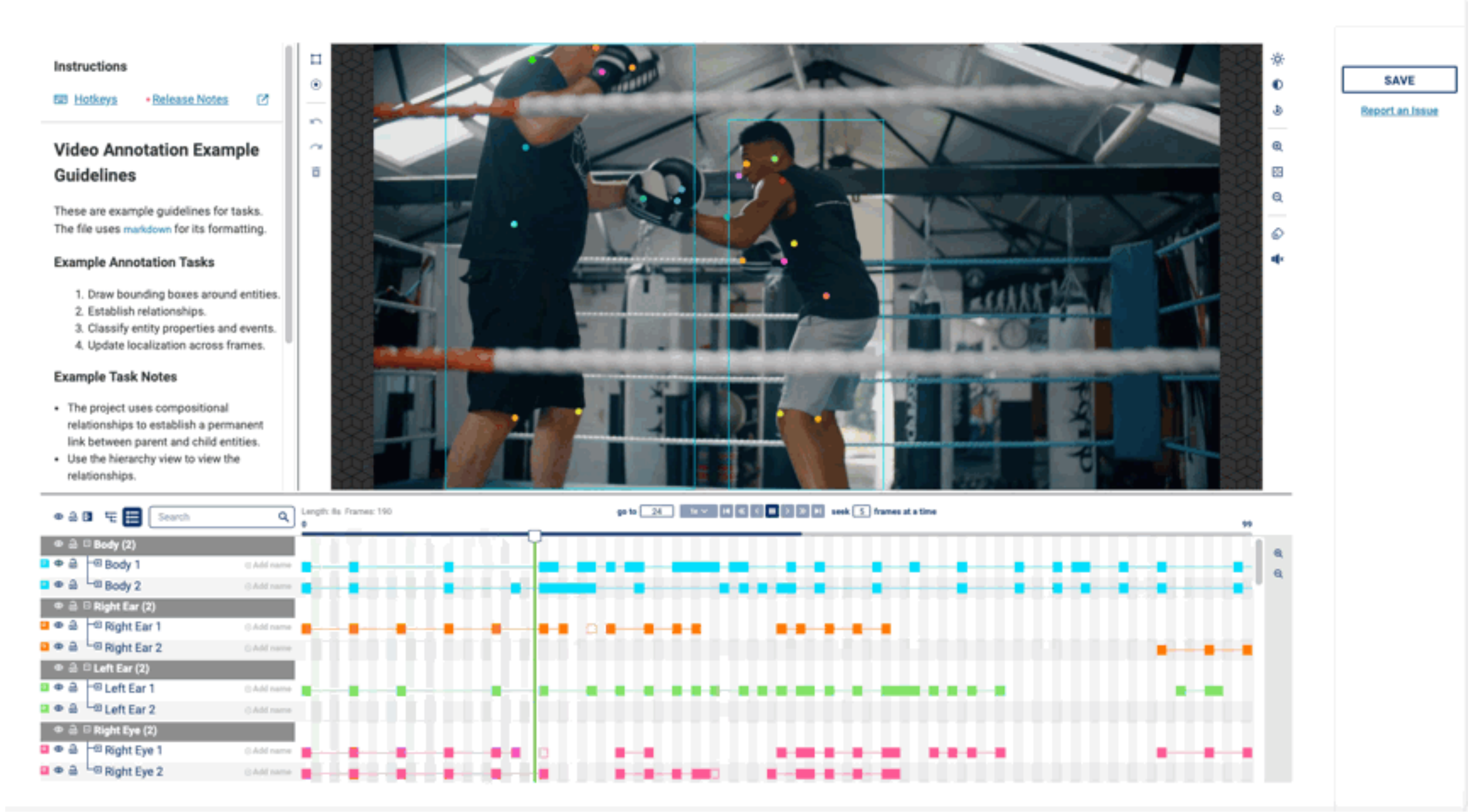
Many solutions, including our naive implementation, simply store all the annotations for every frame in memory. It’s straightforward, but memory usage scales with the annotation count and clip length leading to unpredictable browser performance or even losing hours of work due to a browser crash.

Solution:

Without going into too much detail, we solved this problem by taking a functional approach to storing annotation data, essentially just storing keyframes. The addition of a caching layer between the browser and back-end allows us to minimize the amount of annotation data we have to keep in the browser and ensures that data is always being pushed to persistent storage.

Result:

We figured out how to keep the UI responsive while enabling support for long videos and very high annotation counts helping us meet almost all of our design goals from video size to no video pre-processing.



Video Annotation Example Guidelines

These are example guidelines for tasks. The file uses markdown for its formatting.

Example Annotation Tasks

1. Draw bounding boxes around entities.
2. Establish relationships.
3. Classify entity properties and events.
4. Update localization across frames.

Example Task Notes

- The project uses compositional relationships to establish a permanent link between parent and child entities.
- Use the hierarchy view to view the relationships.

From problems to solutions and results - the back-end

Stable and scalable infrastructure

Problem:

We needed to figure out how to support thousands of simultaneous annotators without performance degradation or work replication. Users are often not able to label an entire video in one session and needed an easy way to query the state previous sessions by all annotators, including themselves.

Solution:

In theIn the back-end, we use dedicated video annotation pods in our Kubernetes clusters to allow us to scale up and down based on utilization. We built a new microservice for video annotation whose primary purpose is to store and retrieve keyframes for users as they are annotating videos. We chose Amazon Aurora for our database which gives us great scalability in addition to being based on the ubiquitous Postgres. We broke up the data logically for each “iteration” that is performed by one or more users.

Result:

This dynamic scalability allows us to handle thousands of simultaneous annotators. Our implementation allows us to easily query the state of each video annotation session while including any previous sessions performed for the same video.

From problems to solutions and results - the back-end

Final output generation

Problem:

If you recall, we switched the front-end to use a functional approach for determining the values for each keyframe (see section on Memory Management). This works great during the annotation process but we still needed to generate a final output file with the values for every frame of the video. We also wanted to guarantee that outputs matched the same results as the browser showed the users.

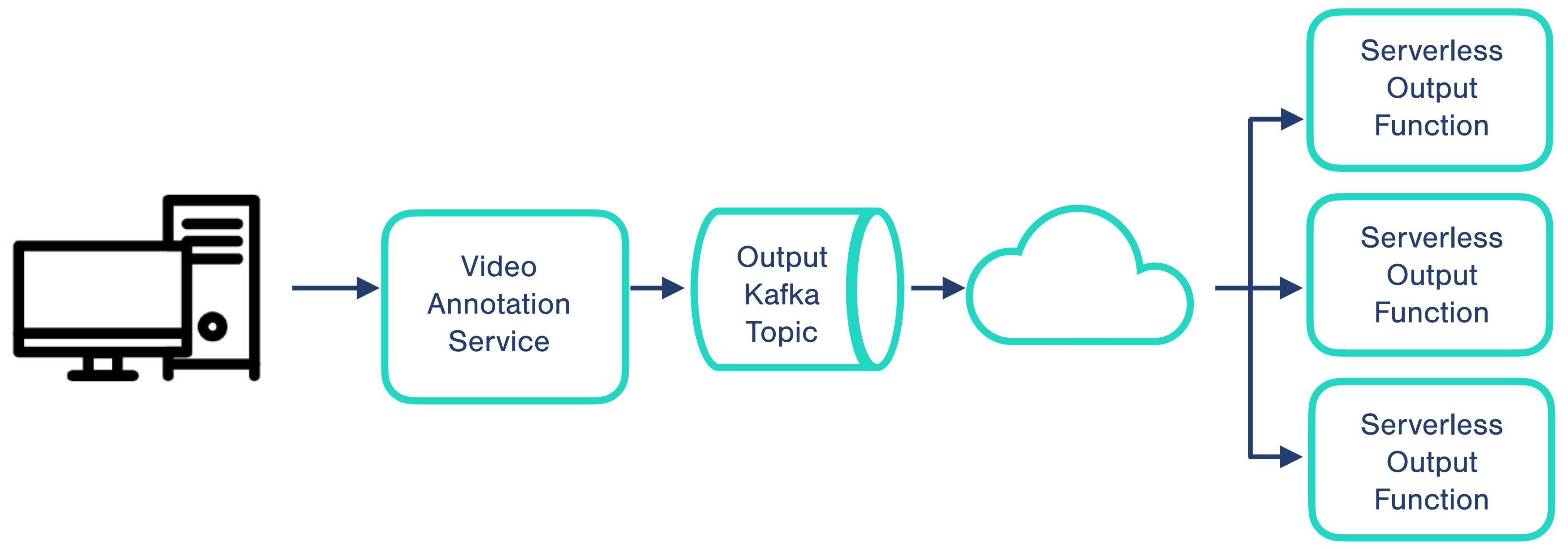
Solution:

To accomplish this we decided to move the generation of the output file to a OpenFaaS serverless solution. The design allows us to publish to a Kafka topic after submission takes place which asynchronously generates the full output file for the video.

Result:

This design scales very well because OpenFaaS manages a queue of messages for a Kafka Topic. It ensures the cluster does not get overloaded by making sure tasks flow out of the queue in a sustainable manner.

Because OpenFaaS offers language flexibility, it allows us to share JavaScript code from the front-end using a Node.js serverless function. Importantly, this *guarantees* the same results as the browser showed the users. Sharing front-end and back-end code when possible allows us to maximize code, and equally important, test reuse.


Video Annotation Service
Output Kafka Topic
Serverless Output Function
Serverless Output Function
Serverless Output Function

Excellence is what we achieved

By listening to our customers we created the <u>leading VA platform</u> solving for their pain points

- **Long videos -** minimum of an hour length time
- **High resolution video -** minimum resolution of 4K
- **Large number of annotations -** 500k minimum
- **Responsive UI -** no perceptible degradation regardless of video length or annotation count
- **Efficient UI -** great U/X even with thousands of objects
- **No video pre-processing -** support what the browser supports while detecting and managing underlying video issues
- **Scalability -** support over 1,000 simultaneous annotators without performance degradation

Every one of these goals was important to us because they represented the biggest needs of our customers. After an intense, exciting year of experimentation and development, we are proud that we have hit these goals and more, creating a solution that makes high quality video annotation not only possible, but efficient, cost effective, and user friendly.

Stay tuned for part 3, where we will discuss the video variables we overcame, focusing on how we created a robust video processing pipeline.

- The expansiveness of the MPEG spec
- Variability of formats, codecs, and encoder/decoders
- The scourge of RTSP (and other streaming encoders)
- Frame rates woes
- Compression artifacts and encoding errors
- The contradictory goals for encoders and decoders

Want to learn more about Alegion's self service annotation platform?

Reach out to solutions@alegion.com

Follow us on