Medical imaging you can count on with Zebra Medical Vision and Temporal

Problem

Existing software for medical imaging devices forces doctors and diagnosticians to manually examine every chart, which is time consuming and error prone. Zebra Medical Vision (ZMV) provides a highly integrated solution, powered by machine learning to automatically identify medical conditions. The result is doctors spending more differentiated time with patients, providing them confidence that nothing critical was missed during diagnosis.

There is no room for error in ZMV's system as the output will be directly used by doctors to make a diagnosis. To provide a consistent and safe analysis, ZMV is built on a complex data processing and machine learning pipeline which runs patient scans through a litany of proprietary algorithms. Any given step within this pipeline can be lengthy and may depend on dozens of external services.

How was this problem solved before?

ZMV's pipeline was originally a coordinated series of microservices written in Python. The pipeline receives files (medical scans) and processes them, eventually spitting a series of labels out into a well understood table format. The pipeline was developed organically, with individual teams implementing each microservice using the tooling and style of their choice.

While this worked great initially, it quickly led to serious disorganization. Data would enter the pipeline and would occasionally not emerge from the other end. Debugging high level issues with the pipeline was overwhelming, as there was no consistency in tooling or design from stage (microservice) to stage.

Before long, data analysts who consumed the pipeline output began to lose faith. Instead of starting a run and getting back to work, they would anxiously supervise the process to ensure its integrity.

> It was a mix of Redis and Cron jobs. The whole pipeline required a lot of manual intervention. When it was failing, you had to massage it, check it every x times and move it from one stage to another manually. You didn’t have a clear picture of what happened to a single element and why. An element that had started the pipeline did not always make it through the end of the pipeline and the only option was to re-run this element through the pipeline again from the beginning. The pipeline consumers (data analysts) were very frustrated about the time it would take to process the data and the outcome that they would receive - so they didn’t trust the pipeline. They had to check the data themselves against the original source to feel confident.

It became clear that the pipeline architecture wasn't just frustrating for ZMV developers; it was also affecting the business (failing to provide reliable data in a timely manner caused delays in the development of new algorithms).

Cleaning up the mess

ZMV developers had known early on that the initial pipeline architecture wasn't going to cut it long term. As the reliability and visibility of the pipeline started to degrade, they were already researching alternatives. Their experience helped them generate a wish list of capabilities for the V2:

- Out of the box visibility
- Can scale to 10's of millions of data points
- Traceable
- Highly reliable
- Supports automatic retries

- Provides throttling controls
- Long running operations

At first they discussed using a more traditional solution like Airflow. After some research they believed that the scale required to process the images would be far too much for Airflow to handle. Conductor was also a contender, but was ruled out immediately due to its configuration-driven nature.

> We also compared [Temporal] with Netflix Conductor and [other alternatives]. It was a lot more convenient and intuitive to write the business logic in code, instead of by configuration. Therefore, Temporal felt like the most natural solution.

A handful of ZMV developers previously had a great experience working with Temporal's predecessor at Uber. It was clear to these developers that Temporal would deliver the visibility, reliability and traceability needed to provide a highly consistent experience to their users and service consumers.

Temporal migration

To understand the initial migration approach ZMV used, you first need to understand some basic concepts in medical imaging. When scans are done for a patient, multiple images are taken to ensure a comprehensive set of data is collected. This set of scans, taken together, is known as a **series**. Unfortunately, a series is not enough to represent imaging for a patient, as multiple series of different types (X-Ray, CT, etc.) may be needed to form a complete picture. The name for multiple series taken with a common goal is a **study**.

Before Temporal, teams across ZMV had no standard for how they processed medical images. Some services had been implemented to work with series, while others had chosen to work at the study granularity. The ZMV team decided that standardizing this API across the company was a great use case for Temporal.

The problem was that many teams were already storing data in the series format. It wasn't enough to change the API; the data also needed to be migrated. Orchestrating this migration seemed like a great first task for Temporal.

> For each study, we needed to download all of its series, un-archive them, repackage them into one archive that represents the study, and upload them back through the pipeline. In addition, we discovered metadata inconsistencies inside the images, which we had to deal with during the migration. Using Temporal allowed us to easily discover this and add validation to the business logic.

The result was not only a successful migration, but also immediate visibility about the state of the old pipeline and its data. The ZMV team was now confident about the state of data in the system and what had failed to make it through the pipeline.

> Running the new pipeline allowed us to make sure that the studies we have are viable and safe to be used in our algorithms. We did find some errors in several scans (metadata inconsistency inside the scans).

Q&A

1. **How have things improved since migrating?**

> Temporal provides great visibility and a simpler way to write business logic. When something happens, we know exactly what happened and how to fix it, and the fixing process becomes a lot faster.
>
> We used the Grafana templates for Temporal dashboards. It helped us tremendously, because now we could observe the execution of activities and their throughput, which helped us to find the bottlenecks in our pipeline and to fine-tune the configuration. One of the most important things was that you can change the rate limiting

> just by changing an environment variable. We used this feature regularly to fine-tune the pipeline and to find the bottlenecks.

2. **Favorite feature of Temporal**

> The usability of the SDK, mainly. The intuitiveness of it, that you run an Activity, get a call back, and you can retrieve the result at any time. It significantly simplifies the business logic. We didn't use advanced visibility, but I think it's a really cool feature and have made plans to utilize it in the future.

3. **Thoughts on Temporal**

> We demoed the new pipeline and its results to the company. People were very excited with the visibility and reliability Temporal brings to the pipeline. It was very well-received!
>
> Moreover, we received positive feedback about data no longer disappearing. A result of the migration is that we now have structured data. That way we know exactly what data is sent and what data analysts are receiving. We got a lot of positive feedback about that too. Previously, it was all unstructured, and then you had to query some unknown world.

4. **About the Temporal community**

> I think it's really great. I'm a member of both the forums and the slack community, and I see almost impossible responses from the team. I mean, in every hour and every day. It's really amazing, the support that people get, for even naive questions.

> I think that the community is growing very well. I saw that the collaboration between teams from other companies is really great. For example, people from teams at Netflix or Airbnb have discussions there, which makes me feel like Temporal is also helping to connect different companies in the industry around a unified infrastructure.