How Datadog Ensures Database Reliability with Temporal

This case study is based on an interview with Kevin Devroede who is a DRE leader and one of the earliest Temporal champions at Datadog.

Context

Datadog is a monitoring, analytics, and security platform that helps companies improve the observability of their infrastructure and applications.

As a product, Datadog runs on millions of hosts, and ingests trillions of data points per day. Customers rely on Datadog to determine whether or not their applications are behaving properly, and reliability is of critical importance.

To make all of this possible, a set of domain specific databases power the Datadog product. To deliver reliability and continuity of these databases, Datadog runs a dedicated function for ensuring the excellence of databases, called Database Reliability Engineering (DRE from here on).

The scope of the DRE team is best defined using language from the job description:

- Keep datastores reliable, available and fast.
- Respond to, investigate and fix issues, whether it’s deep in the Database code or in the client application.
- Build tooling to minimize customer-facing downtime, and scale up resources on short notice
- Protect and ensure the consistency of customer data.
- Work with developers to design data models, and choose the correct datastores, to support orders of magnitude more customer data and traffic.

Problem

In the last few years Datadog has seen immense growth, resulting in the need for engineering teams to scale quickly and continuously. For the DRE team, this meant adding and managing more and more databases. Unfortunately, this also historically meant adding a proportional amount of new DRE members to handle the newly added databases. It quickly became clear that it was unsustainable to add new team members every time they created new databases.

> We didn’t want to grow our team size linearly based on the number of datastores that we run.

How was this problem solved before?

DRE is a partially reactive job function, requiring team members to handle dynamic incidents, upgrades and other scenarios. Historically, the DRE team had a collection of remediation, upgrade and scale-out scripts that would be used manually based on the circumstances. The correct sequence of scripts/steps to run in a given scenario was driven through runbooks. This meant that during incidents, the DRE team was required to manually copy and paste remediation steps often with little to no tweaks.

> Although we had very good runbooks about what to do in case of an incident or if there is an operation request, the problem was that we lacked the “glue” to reliably tie things together. So when there was such an incident, we had to consult the runbook, copy a command, tweak a few pointers, paste it and run.

In other words, there had never been an automated approach to handling incidents, so individual members of the team would often solve things in a one-off or unique way.

Change is needed

After some reflection, the DRE team understood that the deficits and gaps in their own processes and internal tools were the biggest contributor to the inability to scale databases without hiring new people. Every time manual intervention was required, it introduced room for human error or unique choices which break the standard model

other team members expect. This led to drift which in turn led to new hires immediately being burdened with an immense amount of scope and complexity.

> That led to a couple of problems, for example handling versions when doing a fix across hundreds of data stores. What happened was that configuration changes to fix databases drifted. So over time we diverged and had almost one version per datastore and that was far from the ideal.

Migration

As a first step in the migration, runbooks were converted to workflows, removing the need for a human to tie together or even execute individual steps. This greatly reduced the chance of drift between different systems and teams. As an example*, if a runbook previously expected a human to run an operation on a Kubernetes cluster, the Temporal Workflow would instead execute an Activity which runs a K8s command on the cluster*

> For most of the runbooks we already had standalone APIs or tools that we could easily leverage within Temporal Workflows and Activities. For example, instead of having the runbook that tells you "Please run this command on this cluster?" What we'll do is write an Activity that will create a Kubernetes job that runs a command. This was quite easy to do.

As things moved forward the DRE team began to realize the hardest part of the migration was building trust in the new system, simply because it was new. While the original runbook approach was messy, the manual nature meant that if something went wrong an operator was already there to detect and react to it. When the team first started running Workflows they felt anxious because of basic questions:

- How do I pause it?
- How do I stop it?
- If things are going wrong how do I detect it?

> Before Temporal, we knew how to do things, albeit manually. If something bad happened, we could react to it, detect it and remediate. But when team members start a workflow (especially for the first few times), there are always these questions of... How do I pause it? How do I stop it? If things are going wrong, how will I detect it?

It took some time for the team to trust the new Temporal solution. For a while they even ran the new system in parallel with the old one. Eventually this created enough trust that the team felt safe completely moving to the Temporal based solution.

Outcomes

The initial goal of the migration was to move from almost completely manual runbooks to automated and highly reliable Workflows. Once the team developed trust in the Temporal solution, they almost immediately started seeing benefits of the switch. Temporal provided DRE the “glue” to tie together all of the critical work they are constantly doing behind the scenes

> We had close to nothing. We had runbooks, bash for loops and what we call toolboxes, which is like a pod running in Kubernetes next to our cluster on which we can run commands. Temporal was filling a giant gap in our operations.

Automating things not only improved reliability and continuity, it also improved productivity since team members no longer needed to sit and wait for long running operations to run. They just let the Workflow handle it.

> Temporal was not just executing all the commands, it was saving time. Some of the operations involve moving data around that can take forever and so that means context-switching for the person calling. Having a computer doing that for you is way better.

But the most interesting outcome was the one that DRE never expected when they started the migration. The team began to realize that Temporal made it possible for them to implement operations and actions that were never previously possible. Database migrations as an example, can take days or even weeks to fully complete. Before Temporal, DRE team members across the globe would need to carefully coordinate to ensure someone was always awake and monitoring the migration. With Temporal, a single Workflow not only replaces the manual steps but also the need for a human to be monitoring it

> We ended up creating new processes because of Temporal. We used to run operations that span multiple days however we could. When you have something like a database migration, the best case is five to 10 days or something like that. You know that even if we have a team based in Europe and a team based in the U.S., it's hard to always follow that migration and at some point everybody is sleeping. It was hard to do but now because we have Temporal, it unlocked a set of possibilities that were previously impossible and we started to define processes based on that.

This is only one example, as time moved on the team continued to find more and more work that was previously impossible without Temporal

> We've done things that we likely wouldn't have done without Temporal. Large migrations. Operations, we could previously handle, but something like fleet-wide migration wouldn’t be possible without Temporal because it would have taken five times longer. ex: we have a migration of a database between multiple Kubernetes clusters so, we want to migrate from one to the other and that involves creating a new node and a new data center with the data, wait for it to be in sync…

> Versions are another relevant topic. The fact that we were mostly upgrading things in bash, now we can do regular deployments fleet-wide that are spanning multiple days and more complex than they previously could be.

> Take one cluster, change a version on GitHub, do a build, do a deployment. Wait for it to be deployed, make sure no monitors are firing, and rinse and repeat a lot of times. Obviously it's doable manually, but how long would it have taken? I think it wouldn't be possible to do that really. These use cases were unblocked by Temporal.

How do you feel about your decision post migration?

Excitement and investment in Temporal at Datadog has only grown stronger since the initial migration. The DRE team has immensely improved their processes with the Temporal solution leading to higher productivity and reliability. After the first use case went into production, usage rapidly expanded to over 100 users across dozens of teams within just one year, using Temporal for everything from CI/CD to Human metadata aggregation to the datastore operations critical to DRE’s core function.

> It's quite impressive to realize, less than a year ago we were using Temporal in production for the first time. Fast forward 10 months or something, we have many, many teams using it. We love to see it going in that direction. It definitely has a lot of interest.

Areas to improve

Nothing is perfect, and that includes Temporal. One of the wonderful benefits of working with partners like Datadog is the wealth of constructive feedback that helps drive our product forward.

> One area of improvement (especially those who aren’t familiar with debugging Temporal Workflows) is to actually go through the history

> of Activities and understand what the work we're doing is. We have some Workflows that are orchestrating a lot of Activities and events. Searching by Activity type or event type or something like that would be quite useful to be able to do.

> Another area to improve is CRON Workflows. We’re hoping it's going to be revamped to have more options. That would be super nice. In my team we've started to look more into scheduled Workflows and we would really like to see some of the options that were mentioned such as the ability to configure the behavior when a schedule overruns. It currently works, but this would be much better.