Temporal, a central brain for Box

This case study is based on an interview with Steven Cipolla who lead Temporal adoption within Box

Over the last decade, Box has become the defacto solution for enterprise file sharing and cloud content management. In order to stay on top of the competition, Box continuously searches for ways to improve their product architecture that will enable new features and improve the performance of existing functionality. As Box continued to scale, supporting larger customers with ever-growing storage demands, it needed to rework a critical part of its core architecture in a way that both enhanced the already great experience, and improve velocity for product teams to innovate quickly.

Problem

When you're in the enterprise file sharing business, you end up dealing with some big folders. In Box's case these folders often contain millions of files, each with their own set of permissions and metadata. When files are deleted or updated, changes need to be propagated through all other content in the folder hierarchy. Failing to propagate updates is costly and often results in data corruption or inaccessibility.

> A simple example would be if we accidentally copy a folder from Point A to Point B twice, then you're going to end up with two copies of the folder in the destination folder.

Considering that any file in a given tree could be important, it's critical that the system which handles propagation be strongly consistent and completely reliable.

How was this solved before

Until recently, Box solved this problem with a homegrown orchestration system powered by queues and events. For each file operation, tasks were added to a queue which workers would then pull from, re-queueing new tasks and state as needed. Maintaining consistency with this ad-hoc system was not simple.

> What we had previously was this system where we'd have this worker get a message off of a queue, perform some chunk of work and then go put a message back onto the queue with a pagination marker, to perform some other chunk of work.

Each worker and queue required custom logic and state to handle failures and transient issues. Before long, there was an unmanageable number of databases needed to house all the unique state machines for Box's internal logic. Without a unified, top-down view of what workflows were in flight, it was difficult to observe an operation’s state and progress, not to mention pausing or restarting already running operations. Box realized this, and embarked on a project toward transformative architectural change.

A New Solution

Teams at Box began searching for a high-velocity solution that was strongly consistent and provided comprehensive visibility. This effort wasn't just about addressing the immediate problems, but also finding a scalable long term solution for the business.

> In the back of my mind it's always been, "We need to start looking at a technology that can solve the general workflow pattern." We needed a central brain where we can store state.

As a first step of the search process, teams within Box collaborated on a "template" for a generic file operation that any solution would need to support. Once there was consensus on the template, an internal system was prototyped to solve the problem. The API of the prototype was inspired by AWS Simple Workflow, but did not use the service under the hood. Alternatives such as Netlflix Conductor were considered, but were quickly dismissed as they do not support

writing logic as code, and relied upon lesser-known storage and queueing systems. For a while it seemed like Box would move forward with its own internal solution, but a week before the decision was finalized the team stumbled onto Temporal. Due to Temporal's common roots with Simple Workflow, a minimal mental and programmatic shift was required to move away from Box's internal prototype.

> Temporal is an open source project that has pretty much the exact same data model that we're proposing we build to solve this problem. We're not going to build our own workflow orchestration platform. We're just going to take this one off the shelf. It's working at scale. We just need to figure out how to map our problem to this.

A smooth migration

The value proposition of Temporal was so convincing that Box decided to not only port orchestration logic during the migration, but also the service level business logic. Regardless of the confidence in Temporal, Box was determined to approach the transition in a methodical and scientific way. A gradual transition was planned and two major criteria were chosen to evaluate success:

1. Is the orchestration doing what we think it's doing or at least touching all of the same things that we expect it to touch?
2. Is the logic that the orchestrator is orchestrating actually sound (considering that it's also been reimplemented with Temporal)?

Armed with a framework for judging success, the team started with an in-memory simulation to validate the API and architecture would meet their needs. Once the baseline compatibility was proven, rollout began in a highly controlled and step-wise manner - tenant by tenant. At first deployments were only made to internal tenants, whose results were parity checked against the original solution that was still running the majority of the workload.

Now that Temporal has been validated against production workloads, more and more tenants within Box are being migrated to Temporal.

Retrospect

Since the migration, Box has become aware of a few unexpected benefits of moving to Temporal. Outside of the reliable and consistent orchestration Temporal provides, it also has a tendency to map very well to the architecture of an organization itself.

> A big part of the struggle is, "How do I map my software architecture to my organizational structure so that there isn't a whole bunch of codependency between all of these different teams?". If you get a bunch of really smart engineers together and they can solve any problem. And when you look at any sufficiently sized organization, you realize pretty quickly is that it is not the technical acumen that is holding this team back. Often it's the organizational structure, it's the code ownership boundaries. It's all of these sorts of things that really acts as roadblocks to productivity. And with Temporal we were able to model our problem in a way that solved for this as well.

What's next

Temporal has quickly become a hot topic within Box. An increasing number of teams are using Temporal for new projects and migrations of legacy systems. While the use case for Temporal started out narrow and focused, teams at all levels of the product have begun pushing for adoption.

> We have a data platform team that's working on the systems architecture, a file systems team that's building out the templates for doing traversals. We have a team that deals with file and folder governance, and they may need to traverse a tree in a depth first way for their use case. Well, they just pull the depth first traversal workflow and they plug in their activities, and it just all kind of magically fits together.

Liked most about Temporal

> The idea of workflow as code was one of the main things that drew me to the model. It's a lot easier for our developers to use it. It's the way that it should be. Trying to define workflows as JSON and stuff is just... It's gnarly. Workflow as code, you're just like, "All right. This makes sense.". If I'm an engineer trying to implement some SAGA. It's very easy for me to look at this and be like, "Okay, this is how I'd do it."

> One thing that the API does really well is give you this illusion of single threaded or at least local execution without having to get into the details of trying to orchestrate multiple things. It's pretty amazing.

> And I think one of the things that's really nice about the way the Temporal models everything is it just all kind of bubbles up as sort of the same, in the same way. It's an error, and you got to handle it and that's that. It's not different really from... Let's say you were directly accessing a database in a single threaded monolithic application or you were running some crazy distributed computation that had to go and make an RPC to some service on the other side of your data center. That modeling is exactly the same. And as an implementer, the way that I would handle the errors is almost exactly the same. Of course I have to take some special considerations because I don't know. Maybe my activity did happen and the database transaction over on the other side of the data center did occur and I need to be aware of that I choose to retry it. But the idea of doing things at most once gives you that capability to be

> like, "Okay. It might have happened, it might not have happened. I need to check." It's very explicit about the error states that could possibly happen. And I think that's one of the things that architecturally I really like. I think that's a super sound decision.

> If you describe your workflows in this janky JSON structure. It's tough to understand. It's really hard to test. That's another thing. Testability when you're talking about building a workflow pipeline out of JSON. How do you actually test the logic of that? With Temporal, it's so much easier because it's code. You could use the same tools to test your flow as you do to test any other piece of software. And that's a big deal too. Our workflows have a whole bunch of loops and other if statements and stuff. How would I reasonably test that if I had to describe my workflow in terms of JSON documents or something like that? I don't think I would run any tests for that. I don't even know how I would do it. Can you imagine...