Reliable Crypto Transactions at Coinbase

This case study is based on an interview with Anthony Dmitriyev who was a key player in Temporal adoption within Coinbase*

Coinbase has redefined the way that people manage, spend and think about cryptocurrency. Changing the way people think about their money is a hard problem. Expectations are at an all time high whenever a business is responsible for a users hard earned $$$, so finding a model which allows you to innovate without additional risk is critical. This is the exact problem Coinbase was facing when they began re-architecting the system which runs the majority of their transactions.

Problem:

Coinbase processes millions of cryptocurrency transactions every day. From their users' point of view, each transaction is reduced to a binary result of "succeeded" or "failed". Under the hood each of these transactions consists of a series of steps. A simplified example of transaction steps might be:

1. Withdraw foocoin from user A's wallet
2. Withdraw barcoin from user B's wallet
3. Deposit foocoin into user B's wallet
4. Deposit barcoin into user A's wallet

If all of these steps succeed, the user-level transaction succeeds. On the other hand, if a step fails the user-level transaction cannot fail until the steps which were already executed are rolled back. This need to rollback is traditionally accomplished using SAGA - a common pattern for handling rollbacks in distributed transactions.

Until recently Coinbase relied on a custom engine to handle these SAGA needs. The homegrown system was quite reliable and well-suited enough for SAGA support, but when teams began trying to extend the system to other domains things became painful. Each new use case the system needed to support translated into a large amount of plumbing and developer work. Coinbase quickly realized that they were going to need a flexible, general-purpose solution if they wanted to continue scaling and innovating.

Searching for Answers:

A search began for a replacement to their homegrown system. The most important feature was the ability to support the existing SAGA patterns which constitute the bread and butter of Coinbase's business. The contenders were quickly narrowed down to 3 finalists:

- Zeebe
- AWS Step Functions
- Temporal

SAGA workflows were built on each of the contending platforms. In the process, the team realized that both the Zeebe and AWS Step Functions development models forced users to predefine all of their possible execution paths upfront. This is a real problem in the context of SAGAS, since compensation logic is required for each step which can potentially fail. With Zeebe and Step Functions this meant that each step required one or more prebuilt DAGs to handle the potential failures. If your SAGAS consist of a few simple steps, prebuilding DAGs might be feasible, but as complexity increases the overhead becomes enormous.

> Trying to model complex SAGAs using a plain execution graph such as Zeebe and AWS Step Functions provide will make things unnecessarily more complex, plus we want a solution which we could explore for more use cases that don't necessarily use SAGAS. So I think this is where Temporal really shined.

Fortunately Temporal did not come with these drawbacks. Instead of requiring the user to define each possible path upfront using unfriendly DAGs, Temporal enables users to handle failures programmatically. Not only did this make things more manageable, it became possible to share and reuse logic that would have otherwise been isolated.

> Temporal allows us to do SAGAS easily, but is also deep and very extendable.

A slow transformation

When running sensitive workloads at Coinbase scale, it's generally not a great idea to migrate your entire system at once. So once the decision was made to move forward with Temporal, a migration strategy was needed that would enable Coinbase to incrementally shift their critical transaction workload. The team began iteratively replacing each existing component with a Temporal workflow. They relied on Coinbase's open source deployment system Odin to build containers for each of the translated workflows and make them available via blue/green deployments. Each new workflow was put behind a feature flag until it had been running long enough for the team to feel confident putting it center stage. Before long, the majority of existing workflows had been successfully migrated to Temporal.

> For every specific use case we migrated we used a very incremental process. We would take a single use case and translate it to be a Temporal workflow. Then we would put it behind feature flag and do a progressive rollout.

Retrospect:

The migration to Temporal has been a huge win for Coinbase. Temporal maintains the high level of reliability offered by the homegrown system while also providing visibility into running processes.

> Temporal brings visibility into what is happening which was not something we had with our previous system.

Development velocity has also increased as developers can focus exclusively on writing code instead of maintaining a homegrown SAGA solution. Temporal has opened up use cases which weren't even imaginable with the homegrown system.

> Temporal opened up a lot of possibilities of what we could do with the system.

Things that seemed complex before don't really seem that way anymore.

> Things that we thought to be much more complex to implement on top of the old system feel like they are much easier with Temporal.

Afterthoughts:

Teams across Coinbase are excited about Temporal. More and more new features across the business are being prototyped and built with Temporal.

> Temporal has quite a strong "addictiveness" factor to it, once you start using it and figure out the value proposition you immediately see more and more potential use-cases that you can use it for. And getting away from it is a non-trivial effort, as it does make an engineer's life quite a bit easier.

Even though there were reservations about workflows as code initially, Coinbase now sees this as a revolutionary aspect of the Temporal offering. That's because keeping things in code enables developers to make business better.

> I like that you use code to define your workflows. This is definitely a game changer.