Smart Contracts are not Databases

If it is not going to be used by a smart contract, dump it in an event

Alberto Cuesta Cañada

Follow

Coinmonks

--

1

Listen

Share

Anyone that is coding smart contracts as database tables is getting it wrong. That was me as well, until very recently.

Developing blockchain solutions requires you to think differently about data, about control and about privacy. Some of the details are easy to grasp, such as currency being little more than a table of balances and a few methods to operate on them. Some details are more complicated, such as the fact that you are building a solution that you might barely control.

One detail that eluded me until recently is what does exactly mean that data in a public blockchain is, well, public. Looking for a mechanism to easily report on transaction history I stumbled on a very elegant way of providing exactly that while making painfully clear how careful one has to be with data privacy.

On the Ethereum blockchain, you store in state variables only the data that smart contracts are going to use. Any data that you want to archive, you dump it as an event.

In this article, I'll mix a bit of low-levelEthereum architecture, a couple of development principles for the blockchain and a bit of code. With those tools, I'll show you how to track and report on blockchain state changes, how to make your smart contracts simpler, and where you might be making data public without noticing.

There must be a better way to trace data

We needed to provide a complete history of transactions in a decentralized finance platform.

Regulations like MiFID II require that all financial transactions must be reported to regulators on request. That means not only token transfers, but any other actions taken by users that change the state of the blockchain.

My initial research took me to think: "Every state change is recorded in the blockchain, so there must be some way of going through the blocks and retrieving the transaction history". That sounded like a more elegant approach than storing data about all transactions in data structures inside a smart contract. I've learnt to do as little as possible with smart contracts, and as much as possible with external tools.

A database built on transaction history could be rebuilt as many times as needed in as many places as needed, since the source of truth would always be the blockchain. Our smart contracts would not need to store data that is there only as an archive. I asked my good friend Bernardo Vieira to please find me some tool to build such a transaction history database. He knows who is doing what and usually comes up with something we can use.

I was not even thinking about events until Bernardo came back and said: "What about recording the events that were emitted in a blockchain, does this solve your problem?"

It did. It also opened my mind.

Working with events

Until that time I hadn't thought much about Ethereum events. I knew that you are supposed to emit an event instead of returning a value when you change state in the blockchain. I also had learned to capture those events from the frontend to retrieve the result and made peace with the resulting boilerplate. I had no idea what events actually did.

When you emit an event from a smart contractit gets recorded in the blockchain in a semi-structured format. A few fields of the event are always the same, and the variables that you define for the event in the smart contract get appended as additional fields. An event is one of the cheapest operations that you can do in Ethereum in terms of gas. At 4000 gas it is indeed cheap.

Our earliest explorations included setting up TheGraph, which does precisely what we wanted. It captures events from the blockchain and provides a GraphQL interface to query them.

It would eventually be too complex for our use case, but just a couple of queries made me understand that the events in the blockchain really become a write-only database. Some columns are common to all events, and you get a few custom columns if you want to define them.

How we develop software now

Understanding events sounds simple, but it had profound implications on how we coded our solutions from that time onwards. The first change was to adopt a development principle that now we understood:

All state changes must produce an event

By ensuring that all state changes produce an event we don't need to worry about transaction traceability. All state changes will be dumped as events and we will just have the frontend extract information in whichever way it is required.

But what about privacy? Should I be concerned that I'm putting out all the data that is sent to me? Well, the data is recoverable from the transaction data anyway, I'm just making it easier to consume.

All data in a public blockchain is public, remember.

If you have read my other articles, you will have noticed that I always look for the simplest way of accomplishing something. Until now I was happy with this code for a simple document registry:

Simple enough, but it could be simpler and more efficient.

Note that this second contract doesn't explicitly store anything, and doesn't have a function to retrieve data. Yet it works and does the same job

.

We tend to think of events as transitory things, but everything is permanent in the blockchain, including events. If you want to know whether a document is genuine, you just need to compute the hash and run a query on the cache.

The only catch is that you need to keep an external infrastructure to track the events and mimic the blockchain state off-chain. That is not only relatively easy, but it is probably best in terms of performance anyway.

In starter-kit you can find a sample event cache. Please keep an eye on it, because this is a tool that now we use in all projects and should improve rapidly.

Conclusion

The change in mentality is profound, suddenly there is a clear split between information that you need to keep in a structure in a smart contract, and information that you should just emit and forget.

Store data in a state variable only if a smart contract is going to need it, otherwise it goes away in an event.

I used to say that in any blockchain solution only 10% of the code is smart contracts. Doing traceability right it is even less. The result is a much cleaner code.

In the following months, we will continue coding smart contracts, using the principles shown in this article and refining the tools we use. Please keep an eye on this repo for improvements on contracts and tools.

And as always, if you have any feedback, please drop me a line!