Immutable architecture

An approach for evaluating and building distributed systems from the perspective of immutable objects.

By David Martínez Castañón, Iván Menéndez Mosegui and Jorge Joaquín Gancedo Fernández

Author

Michael L. Perry is a software architect and consultant who specializes in building large-scale distributed systems. He is the author of several books on software architecture and design. The one he talks about in this podcast is "The Art of Immutable Architecture".

He speaks frequently at conferences and events about topics related to modern software development and founded a software development consulting firm called Clear Measure. In addition, his work has been featured in various publications and he is recognized as an expert in his field.

What is immutable architecture?

Immutable architecture is a software design approach to distributed systems that focuses on the use of immutable data structures. It is usually used together with append-only databases that allow new records to be inserted but not modified or deleted. Instead of modifying the existing data, new data structures will be created. This way it is possible to keep track of the changes that made the system be in its current state while its reliability and scalability are improved.

In an immutable architecture, data does not have state. Since changes are modelled as new objects, several threads may work on different copies of the same data without the need of synchronization mechanisms. The risk of data corruption is also reduced, leading to a more efficient caching in which the same data is reused across requests.

However, there are some requirements for using it in a project. Operations must be commutable, that is, their order should not affect the outcome. The system also needs to be able to store and retrieve large amounts of data in an efficient way, and to process the sequences of events.

Differences with immutable infrastructure

On one hand, immutable infrastructure refers to the approach in which services are outright replaced instead of maintained or updated when the underlying configuration must be changed. On the other hand, immutable architecture is entirely different: rather than an approach to how software is built, immutable architecture solves the problem of multithreaded persistence by storing new data.

Therefore, although they seem to be closely related concepts, they are instead similarly named concepts which have entirely different meanings, and the use of one does not mean using the other. An example of immutable infrastructure is the use of Docker images or infrastructure as code like Terraform or Vagrant.

Immutable architecture, the CAP theorem and the two general's problem

The two generals' problem is a known old computer engineering problem. The statement is simple: *there are two allied generals on opposing hills of a valley, in which there is a castle they want to conquer by a predetermined date. If only one of them attacks, then they lose, so they need to coordinate their attack to ensure their win, and to coordinate they dispatch messengers. However, how can one general ensure the other has received his message without the castle finding out about the plan? And if the other general agrees to his plan, how can he be sure his response made it through the valley?*

The problem is, in fact, a simplification of networking and is unsolvable. However, it can be used to present another one: the problem of consistency. If we change the generals for database separated in different machines, then we arrive at another problem: how can we ensure our system is consistent while making it readily available? Because in a real environment, concurrency exists. Here it is where immutable architecture comes to the rescue.

Immutable architecture does not achieve consistency *per se*, but *strong eventual* consistency, so that, in the case of distributed systems that must be always available, we achieve consistency in the long term instead of each time a change is produced, usually without conflicts.

Historical modelling

As defined by Michael L. Perry, "*Historical Modeling is a method of distributed smart client software construction. It is based on a model of software behaviour as a graph of partially ordered facts*".

Event sourcing and historical facts

In event sourcing, we have a totally ordered stream of events, so that given any pair of them we can always tell, no matter what, which one comes first. This way, to reach a certain state of an object, you run the set of events in that specific order, and you ensure that the object will have the desired state. The events do not have knowledge about each other.

However, historical facts know its parent, and we can apply transitivity to know the predecessor of a fact (the predecessor of the parent of a node is also a predecessor of the node). This way, we are able to navigate through the history of changes. They are partially ordered in the sense that there may be cases in which you cannot establish the relation between a pair of facts.

Steps of the process

1. *Identify the changes*

First, we create a list of the user operations that modify the system. Then, for each of them, we assign a name to it, and we give it one or more parameters (they can be simple data, entities or other changes). After that, we represent each change as a node in the graph, with arrows (directed edges) pointing to the changes that it depends on.

2. *Refine changes*

In this phase, we put together changes, pull out entities and register constraints (prerequisites).

3. *Query the model*

After refining the process, the user may ask questions when making changes. So, we must write queries that answer those questions.

4. *Repeat*

After performing the three previous steps, there may be questions that are still to be answered. So, we go over the previous phases as many times as we need until the model can reply to all the proper questions.

Bibliography

- Radio, S. E. (s. f.). *Episode 447: Michael Perry on Immutable Architecture : Software Engineering Radio*. https://www.se-radio.net/2021/02/episode-447-michael-perry-on-immutable-architecture/
- *Historical Modeling*. (s. f.-b). https://historicalmodeling.com/
- Perry, Michael L. (2020). *The Art of Immutable Architecture: Theory and Practice of Data Management in Distributed Systems*. Apress.