Apache Kafka is not for Event Sourcing

Jesper Hammarbäck Follow
Jan 22, 2018 · 3 min read

Apache Kafka is a cool product, but if you are thinking about using it for *event sourcing* you should think again.

Kafka is a great tool for delivering messages between producers and consumers and the optional topic durability allows you to store your messages permanently. Forever if you'd like.

So, if your messages are **events** you can use Kafka as an *event store* or an *event log*, but it really isn't a suitable tool for *event sourcing*.

Here's why.

Loading current state

When a service receives a command requesting a state change of an event sourced entity, we first need to recreate the current state of that object. We do that by loading all previous events for that particular entity ID from our event store and then re-apply them on our object, fast-forwarding it into its current state.

Loading events for a particular entity like this is not easy in Kafka. Topics are usually centered around and partitioned by entity types like "*Orders*", "*Payments*" or "*Notifications*" so we would have to go through *all* events for *all* "*Orders*" and filter them by their ID to load up a single "*Order"* entity. Although this is POSSIBLE, it's not very practical.

One alternative would be to have one topic per entity but then we would probably end up in a situation where we have thousands of topics and on top of that, the subscribing services downstreams need a way to automatically discover the newly created topics for each new entity. Again—not very practical.

Consistent writes

When our entity's state has been recreated, it's time to execute the business logic requested by the incoming command. If the business logic fails we return an error to the client but if it succeeds a new event is emitted. In that case we must be able to save the new event to our event store with a guarantee that no other event has been stored for this particular entity ID in the meantime, or we would risk breaking the consistency of our domain objects.

OCC to the rescue

One way to guarantee write consistency is by utilizing the event store's *optimistic concurrency control*. A proper event store provides a way for the user to say "save this event only if the version of the entity is still x". Kafka does not support this and the suggested workaround from the experts in the field seems to be to put a "database" in front of it all to provide a consistency point. While this suggestion might be a viable solution in some cases, choosing a tool more tailored for the specific problem is probably wiser in the long run.

Single-writer

Another way to get consistency is to assure serialized writes, i.e using the *single-writer principle*, meaning we make sure all writes concerning a particular entity ID occur on a single thread. It is possible we could make this scenario work by making our producer a consumer of its own events and effectively block until the event is committed and available in the other end of Kafka. This kind of design would have severe impact on performance, especially considering the limited loading capabilities described above.

Does it fit at all?

So, is there a place for Kafka in an event sourced architecture? Maybe. Probably. It might be a good complement to your event store as a way of transporting events to downstream query services or read models.

However, we should always be careful when adding large and complex pieces of infrastructure to our systems —everything comes with a cost so make sure it's a good enough fit for your problem at hand!

If you want to get started with event sourcing, value the simplicity of HTTP and appreciate hosted services, take a look at what we did at Serialized IO
— Happy sourcing!