# Proposal: An object store backend for Irmin

Date: 12-01-2022

## Summary

This proposal is to add another backend (or, alternatively, storage layer) to Irmin based on an "object store". Possibly we can then replace some parts of irmin-pack with this layer (but most of irmin-pack will remain untouched). There are many advantages to doing this. However, in the near future the main advantage is that the object store can provide direct support for garbage collection.

## What is an object store?

An object store is essentially a persistent key-value map, where the values are "marshalled representations of objects", and the keys are "object identifiers" (or references). An object store typically implements an interface similar to:

```
type t
type key = string
type value = string

val open_store: flags:flags -> filename:string -> t
val close: t -> unit

val add: t -> key -> value -> unit
val delete: t -> key -> unit
val find_opt: t -> key -> value option
```

The values here are strings, and it is the responsibility of the next layer to decide how these strings are interpreted to represent objects. Keys and values are not limited in size, but typically keys will be small (say, 8 bytes), whereas values will typically range in size from very small (a few bytes perhaps) to potentially several MB or more. In this interface, keys are freely chosen by the caller.

Aside: A more complicated interface exposes inter-object references, allowing (for example) functions related to reachability to be implemented by the object store. This can be built on top of this basic object store layer if required.

NOTE An important point about the above interface is that the "delete" function not only removes the object from the store, it reclaims the space that was taken by that entry (although reclamation may happen asynchronously).

## Relevance to Irmin backends

Irmin backends do not typically have a "delete" function. However, when building some kind of GC on top of an Irmin store (as we currently envisage for the Tezos use case), this functionality can be extremely useful. For example, if we use this in-built functionality, we do not have to worry so much about **free space tracking, object relocation, compaction, etc. etc. That is all provided by the object store.** We must still decide which objects we want to delete (say, by object graph traversal) but the basic storage management is already in place.

## Poor performance of existing key/value backends for the Tezos usecase

We already know that even well-respected key/value stores such as LMDB do not perform well enough for the Tezos usecase where the number of objects that are indexed is potentially huge. Similarly for databases such as SQLite. In short, it appears difficult to find an existing database or key/value store which has performance good enough for the Tezos usecase.

## An object store with good performance

In earlier work we developed a replacement for Irmin's current index, called kv-hash.This has various "good" properties for the Tezos use case: it supports single-writer-multiple-readers for example. We proposed kv-hash as a replacement for Irmin's index.

**In fact, kv-hash is (almost) a generic object store.**

It does not currently support "delete" and space reclamation, but that would not be too hard to implement (say, a week of work). Thus, in a short time we could make available a high performance object store suited to the Tezos usecase.

On top of this, we already know from preliminary testing that kv-hash performs well under the Tezos workload (although that used kv-hash only as an index; now we propose to use kv-hash as a full object store).

# Proposal

We believe that the functionality provided by a generic object store, particularly the "delete" function, can be extremely useful when developing new features for Irmin, such as garbage collection.

Although off-the-shelf object stores are not suitable, we do have an object store implementation that is fairly close to being directly usable as an Irmin backend, can support "delete" and disk space reclamation, and we believe will perform well on the Tezos usecase.

**The proposal, then, is to use the kv-hash object store as the lowest storage layer on which to build irmin-pack's GC features.** The advantage is that implementing GC at that point should be much simpler.

[Aside: Because all objects in kv-hash are effectively indexed, it is quite easy to move them around on disk; this is used when implementing kv-hash's own garbage collection for instance. This feature can be used to move frequently used objects so that they reside in a sequential part of the store, thus potentially also providing a way to deal with the archive store issues. In comparison to the existing irmin-pack *external* index, kv-hash has an *internal* index which improves performance significantly.]

# Risks

There are many risks. We focus on the main risks as we perceive them.

**Performance:** There are many tradeoffs when designing storage systems. kv-hash assumes that the stored data is huge (many times larger than main memory) and that the active set is also huge (much larger than main memory). As such, kv-hash functions with an assumption that "each read or write will definitely incur a disk IO", that is, kv-hash assumes that caching disk blocks in memory is pointless because the active set of blocks is so large that a cached block will not be revisited before it is expelled from the cache. Now, the OS can cache blocks of course, and kv-hash will perform faster as a result. But kv-hash doesn't rely on this caching. The expected case is that each read requires a block IO. On the flip side, this means that the performance is reasonably consistent: each access pays the cost of one random read from storage, and SSD random reads are relatively fast. This puts a strong lower bound on performance. [Aside: kv-hash will not perform well on harddisks.]

**Feasibility:** It may turn out that implementing a full object store is infeasibly hard. It should be possible to determine whether this is the case by spending a few days fleshing out the design of the kv-hash delete implementation.

**Integration with existing irmin-pack code:** Although I am confident in the kv-hash codebase, I am not so familiar with the irmin-pack codebase. In particular, I am not sure how much work there would be to reorient the existing code to use the kv-hash object store. I would hope that the work would not be too much, because the object store is similar to storing objects in a file (which is what we currently do), but with the advantage of the "delete" function (which is hard to implement on top of a raw file of course). Thus, the rest of the team would have to assess whether this proposal is feasible.

**Only solves part of the problem:** Even if we adopt kv-hash as the storage layer, we still need to implement irmin garbage collection. A naive GC (object traversal, followed by deletion of dead objects using the object store "delete" function) would be possible, but we believe we can further adapt kv-hash to support more efficient GC, for example, by introducing a notion of "GC period" into kv-hash, and garbage collection based on which objects have not been visited during the period. From the Irmin side, the period would be signalled by a call to kv-hash; then live objects would be visited; then the end of the period would be signalled. At this point, kv-hash could internally (and asynchronously, in parallel) complete the garbage collection. The point is: there are various strategies that could be considered, and then implemented within kv-hash, without disrupting too much the existing irmin-pack codebase.