U061HGP8C : maybe you're missing a channel
U061HGP8C : read file -> channel -> http calls -> channel -> write file ?
U061HGP8C : if that's the general shape and you want to ensure some parallelism in the "calls" step, maybe look at what the `pipeline` functions have to offer in core.async?
U061HGP8C : I would suppose they already handle closing the channel in the right order
U5YA41N3G : Can anyone either give me some thoughts or tell me the best place to ask about some architecture issue I'm trying to sort out?
Long story short, I have a Clojure + Clojurescript web app that needs to extend itself as a desktop app and as a user-owned server (with web admin api) that can have user created functions (think add-ons, plug-ins, arbitrary user code). I'm trying to find the best way to implement these two new pieces that highlights ease of installation, extensibility, minimal system requirements, and shared codebase as much as possible. The desktop app itself is surely going to end up being electron-based. As far as the code, there's a "simulation" engine that I don't want to rewrite in multiple languages unless from scratch to be shared, and is currently in .cljs files.

My ideas thus far are -
Option 1: Write the user-run server in Clojurescript + Node.js (running on node.js as the server) with user-defined JavaScript functions that get called from Clojurescript/JavaScript, and expose the Clojurescript API via externs
Option 2: Write the user-run server in Clojure and run a v8 instance where I eval anything written by users in pure JavaScript, returning and sending json to communicate (at the cost of some speed and cruft).

Any other ideas or thoughts? Or anywhere someone can point me? Thanks.

U056QFNM5 : <@U5YA41N3G> - Don't have time to think deeply about this now, but your question strikes me as one that could be asked at the Clojure or Clojurescript subreddit. That way it's less likely to be buried by other conversations while people think about it.
U5QCSK76C : Any news on when clojure.spec become part of core?
U09LZR36F : It won't. It's been split into its own library.
U051SS2EU : wasn't it going to be merged in later, on a "when it's done" kind of schedule (so after 1.9) ?
U054BUGT4 : There's a good discussion about this on the latest defn podcast (<https://defn.audio>) - with Stuart Halloway - it's not on the website yet but it's episode 23 - it's in the RSS
U380J7PAQ : Hi, I'm trying to figure out how to do something that's embarrassingly simple, but im just having no luck lol. Basically, I need something that does the following```(check #{1 2} [1 2 3]) => true
(check #{1 2 4} [1 2 3]) => false
```

for check i've been through variations of some, every? not-any?, etc. But still can't quite get what I'm looking for . I just need to flag the fact that there's something in the first collection that's not in the second

U5XMV6DQT : convert 2nd arg to set?
U11BV7MTK : are you looking for set equality or set subset?
U380J7PAQ : not equality, just that "a has something that b doesn't" ,it's ok if b has somethign that a doesnt
U380J7PAQ : yeah I guess set/difference would work too
U380J7PAQ : the problem is that 'set b' may be large
U11BV7MTK : subset sounds like it?
U11BV7MTK : oh, you need one to be a proper subset of the other
U380J7PAQ : so was wondering about the cost of converting from a list/vector to a set to do the comparison
U380J7PAQ : but i think I'll just go with that for now
U5XMV6DQT : set uses transients, so don't think it would be a big overhead
U380J7PAQ : ok wasnt sure how effcient it was in that scenario
U5XMV6DQT : ```(defn set
  "Returns a set of the distinct elements of coll."
  {:added "1.0"
   :static true}
  [coll]
  (if (set? coll)
    (with-meta coll nil)
    (if (instance? clojure.lang.IReduceInit coll)
      (persistent! (.reduce ^clojure.lang.IReduceInit coll conj! (transient #{})))
      (persistent! (reduce1 conj! (transient #{}) coll)))))
```

U380J7PAQ : ok will give that a shot if it behaves then i'm good
U380J7PAQ : cool
U5XMV6DQT : I just jumped to definition in my emacs/cider (M-.) :slightly_smiling_face:
U11BV7MTK : so i guess it would be `(and (subset my-set (set coll)) (not-empty? (remove my-set coll))`
U064X3EF3 : the true answer is: we have not yet decided
U064X3EF3 : it is effectively part of core now as Clojure depends on spec.alpha. thus users of Clojure get both. Whether and when the two units of code are co-located will depend on how things unfold (but should not actually matter to users)
U061HGP8C : `(not-empty? (set/difference (set a) (set b)))` ?
U11BV7MTK : that wouldn't ensure that a is a subset of b though
U051SS2EU : if the difference is empty, a is a subset of b