U5ZAJ15P0 : <@U0BKWMG5B> thanks for taking the time to explain this out! I'll read up on Duct/Integrant. The theory sounds great; let's see if it holds up in practice :stuck_out_tongue:
U0BKWMG5B : No problem - let me know what you think
U0BKWMG5B : The two blog posts I linked, especially the later API-focused one, should give you a good idea of how to start.
U087U9YG3 : If you want to evaluate mount/component/integrant, I'd recommend starting by just reading them
U087U9YG3 : they're all in the 300-500 LoC range
U5ZAJ15P0 : <@U0BKWMG5B> I think I am going to have some fun with `clojure.core/derive` itself too. Such a neat little function
U0BKWMG5B : I wouldn't say it's a function that should commonly be used; it's got niche functionality. But I also think it's underused. Not many people seem to have heard of it.
U087U9YG3 : I mean there are things you can only learn by using them in a large project for a long time, or talking to someone who has, but I think reading them is a reasonable starting point
U5ZAJ15P0 : Yep that's what I intend to do now that I've seen they're pretty concise. Thanks for the advice!
U050MP39D : if I'm understanding your example correctly, component also wires everything together in one file like you prefer
U050MP39D : generally you have a 'system' or whatever ns with a big map that looks like
```
(defn example-system [config-options]
  (let [{:keys [host port]} config-options]
    (component/system-map
      :db (new-database host port)
      :scheduler (new-scheduler)
      :app (component/using
             (example-component config-options)
             {:database  :db
              :scheduler :scheduler}))))
```

U3SG7RX7A : Aaand don't forget the new-new kid on the block:
`deferst`:<https://github.com/employeerepublic/deferst>

U2PGHFU5U : Yes that is what I am missing from Mount :slightly_smiling_face:
U2PGHFU5U : Your `example-system` is a Composition Root
U2PGHFU5U : <http://blog.ploeh.dk/2011/07/28/CompositionRoot/>
U5ZAJ15P0 : <@U0BKWMG5B> Out of curiosity, why did you make the choice of using multimethods for Integrant?
U5ZAJ15P0 : It seems to me (as a clojure beginner) that it introduces a form of global that makes stubbing a bit harder
U5ZAJ15P0 : e.g. wouldn't it be nicer to pass a second argument to `ig/init`: a map from keys to implementations?
U5ZAJ15P0 : I am sure you had a good reason for using multimethods and not the approach I'm suggesting, but I would like to understand it
U5ZAJ15P0 : <@U0BKWMG5B> is it related to hot-reloading?
U11BV7MTK : do clojure files have to have a namespace?
U11BV7MTK : and if so, should tooling not assume that there will be a namespace form at the top?
U0NCTKEV8 : they don't
U0NCTKEV8 : and you can put other things before the namespace
U0NCTKEV8 : (the ns form)
U11BV7MTK : ok
U11BV7MTK : never really know where to go to find the hard requirements on these things so thanks for the info
U0NCTKEV8 : it is super common to have the ns form be the first thing, so some tooling does assume it
U11BV7MTK : ok. i was not positive if it was convention or spec
U0NCTKEV8 : so some tooling requires it, but clojure does not
U11BV7MTK : well, if possible, i'd like to conform to clojure requirements and not some subset
U11BV7MTK : there were some namespace cache changes in CIDER that seems to affect files without a ns form so wanted to see what the requirements were. thanks a bunch
U0567Q30W : @puredanger Are your deps slides available anywhere, until the talk is online?
U0567Q30W : I mean <@U064X3EF3> ^^
U61KCTX8S : hi all, good morning
U61KCTX8S : how do I check wether a process is alive?
U61KCTX8S : i mean i launched a subprocess
U61KCTX8S : whether with sh or conch
U61KCTX8S : and i'd like to know whether it's finished or not

U064X3EF3 : I'll put some stuff out early next week - slides, code, and some more info
U1WMJ5CQ2 : <@U61KCTX8S> `sh` is a blocking call:
```
(sh "ls")
(prn "finished")
```

U064X3EF3 : want to have a little more context than just dumping the slides. haven't had a chance to do all that stuff due to the conf and today's travel home.
U61KCTX8S : i did it like:
U61KCTX8S : (defmacro try-noooo "Returns the result of evaluating e, or :noooo if it throws an exception." [e] `(try ~e (catch java.lang.Exception _# :noooo)))(defn is-proc-running[p] (if (= (try-noooo (.exitValue p))   :noooo) true false))
U61KCTX8S : sorry about the formatting, havent figured this out yet on slack
U61KCTX8S : the .exit-Value call throws an exception if the process is still running
U61KCTX8S : that's the trick
U1WMJ5CQ2 : <@U61KCTX8S> If you use the built-in future to handle async, you can do this:
```
(def process-future
  (future (sh "./my-script.sh")))

(when-not (future-done? process-future)
  (prn "still running...."))
```

U0BKWMG5B : hmaurer: A lot of the time stubbing isn't necessary; only for keys that connect to an external I/O source. Also, polymorphism is more convenient than a lookup table, and it encourages using the same key for the same behaviour.
U0BKWMG5B : It's important for Duct/Integrant's design around the idea of a vocabulary that keywords have the same meaning.