U08FM7RL1 : okay, this is what I'm looking for
U08FM7RL1 : please explain
U08FM7RL1 : if you would be so kind
U050MP39D : I think this thread should explain it for you
<https://groups.google.com/forum/#!topic/clojure/OeXZOJYydAs>
U08FM7RL1 : very helpful, thank you <@U050MP39D> !
U050MP39D : no worries, I remember finding that consequence pretty interesting myself
U050N5T47 : Anybody have tips or know of a write up or library for implementing fine grained role based permissions in a ring/compojure web app? I mean something like check that this particular user has a role with permissions to delete/modify this particular record.
U0K0TFQLW : <@U050N5T47> friend can help with that: <https://github.com/cemerick/friend>
U050N5T47 : Thanks <@U0K0TFQLW> I want something like the authorize macro but more focused and easier to use (with docs) to recommend to another developer.
U050N5T47 : Also, that doesn't depend on friend specific attributes (like *identity* being bound, hierarchical keywords, etc.)
U3QUAHZJ6 : is it possible to update 2 keys at once inside a map using clojure's `update` function? something like```
(update {:a 20 :b 10} :a inc :b dec)
```

U0K0TFQLW : all you really need to do is have some ring middleware that populates a session key containing a user-id in the ring request. then your controllers can take that user-id and build whatever authz (role-based, fine-grained permissions based on models, etc) that you want
U0K0TFQLW : there are a bunch of helpers for doing things like working with encrypted+signed session cookies
U050MP39D : <@U3QUAHZJ6> not with update afaict, reduce works though (reduce #(update %1 %2 inc) [:a :b])
U5ZAJ15P0 : <@U3QUAHZJ6> what about this?```
(merge-with #(%2 %1) {:a 20 :b 10} {:a inc :b dec})
```

U050MP39D : ^that's pretty bloody cool
U051SS2EU : or `(-&gt; m (update :a inc) (update :b dec))`
U051SS2EU : which I think tends to be the version that is easy to read and refactor
U09FEH8GN : So how do you provide high availability? Right now in prod we just have on JVM instance that handles our web traffic, datomic peer, and background jobs. It's handling the load fine but we don't want it to be a single point of failure. Also it's been nice to reason about things by just having one process.
I figured we could take two approaches. The Datomic transactor approach, have a backup instance that just sits idle and takes over if the main instance dies. The other is to have a pool of instances that can operate independently, possible separate web workers from background job workers. Sort of like the way Rails does it.

U050ECB92 : ^ that GenericVersionScheme already proved useful
U050ECB92 : you can't override the broken spec.alpha reference from 0.1.94 to 0.1.123 right now, because the comparator is busted
U07TDTQNL : <@U09FEH8GN> most systems I've worked on use a load balancer and more than one web server. If one server goes down the loadbalancer stops sending requests to it
U09FEH8GN : <@U07TDTQNL> do you see anything inherently flawed about the Datomic transactor approach for web servers? Just because we built our whole architecture assuming one instance.
U07TDTQNL : It's really hard to implement