Elixir ETS Versus Redis

Last updated: 2017-05-05

by Barry Jones | 5 Comments

Development

Reading Time: 4 minutes

Learning Elixir has a way of challenging everything you know about programming. Redis is becoming an assumed part of many web stacks, in the same breath as your database. But with Elixir, do you need it? Do you even need a database?

The Changing World of Web Application Operation

Here's the standard mode of operation for a web application these days:

- A request comes in;
- travels to one of a few web servers;
- gets some data from a database (or puts some stuff in one);
- does some stuff with it;
- returns a response to the client.

In many cases, we need to either write data at a very high rate or cache data to avoid reprocessing an expensive query. We also need these high-volume cache and write locations to be available to all of our web servers — which is where Redis comes in.

In the world that we're used to living in, that makes sense.

But the world of BEAM/OTP (the Erlang VM that Elixir runs on) is different. In this world, your web servers are connected together as a cluster and talk to each other regularly, even over different datacenters.

In this world, there are no mutex locks needed to coordinate variable-access, write-heavy loads. In this world, your platform comes with three different databases already built into the runtime (ETS, DETS, Mnesia), as well as a number of other options like Agents to handle these circumstances. Even open-source libraries like Riak Core can be implemented to provide an in-server database that naturally grows as your web servers grow.

The level of concurrency is like nothing you've ever experienced before. Because each process is prescheduled, a single processor hogging loop can't prevent the other parts of your system from responding on time as they can with languages like Node and Go.

This changes the game. All of your needs are provided for out of the box in a reliable, fault-tolerant manner that naturally grows as your application grows, without having to maintain separate systems. And that's pretty cool. That means that when you start an Elixir application, you have to ask yourself some questions that you're not used to, like "Do I need *another* database?", "Do I need a caching system at all?", or "Is there another benefit to consider?"

As a side note, having those options built in makes Elixir a really interesting platform for embedded systems (see the Nerves project for more on that).

Comparing Features

When talking about features in a comparison like this, we have to qualify it a little bit. With BEAM/OTP, you're going to have a lot of tools at your disposal that will give you the ability to create features within your application based on its specific needs.

"Specific needs" is the key part here. At every step, we get to challenge whether or not a particular need should be served within BEAM itself or as an outside system/dependency. The table below is going to attempt such a comparison. Keep in mind that the BEAM column is to reflect native functionality that's provided to the entire system (such as ordered message passing to support atomic operations).

Feature	Redis	ETS	DETS	Mnesia	BEAM
Lists	Y	Y (bag)	Y	Y	
Hash	Y	Y (set)	Y	Y	
Sorted Set	Y	Y (ordered_set)	Y	Y	
PubSub	Y				Y
Atomic	Y	Y		Y	Y
Compression		Y	Y	Y	
Concurrency	Y	Y	Y	Y	Y
Disk Persistence	Y		Y	Y	
RAM Only	Y	Y		Y	
Partitioning	Y			Y	Y
LRU Cache	Y				Y
Distributed Lock	Y			Y	
Tables		Y	Y	Y	
Transactions	Y			Y	
Clustered/Fault Tolerant	Y			Y	
GeoSpatial	Y				

You can see from the chart above that most of the features of Redis are available in the built-in options. The biggest

difference is simply deciding which option makes sense within your system. The main differences between the three specified options are:

1. **ETS** – RAM based tables, single node
2. **DETS** – Disk based tables, single node
3. **Mnesia** – RAM/Disk-based transaction RDBMS, transparently distributed across nodes

BEAM is highlighted for certain features because the highly concurrent, distributed, sequential message passing nature of the system makes it possible if not trivial to implement certain functionality (such as an LRU Cache or PubSub) for any solution. There's not a good built-in option for Geospatial, but at the moment, that's just about it.

In Conclusion

This isn't to say you're never going to need Redis. Every Elixir application isn't necessarily deployed to an environment where clustering is possible. For example, if you are using Phoenix Channels and/or Phoenix Presence but deploying to Heroku where clustering isn't possible, there are adapters that leverage Redis as an intermediary just as you would in other languages.

One thing that I've found from talking to a lot of Elixir developers at ElixirConf is that the community overall is very pragmatic. Don't cluster if it doesn't make sense. Don't hot deploy just because you can. In that same vein, don't avoid Redis if you need it just because you can… but at the same time, don't create a dependency out of habit if you don't need it.