Building a cloud with Erlang and SmartOS

How hard could it possibly be?
Spoiler
Spoiler

Quite hard!
Who am I

- Writing Project FiFo
- Twitter: @heinz_gies
- Github: https://github.com/Licenser & https://github.com/project-fifo
- IRC: Licenser
Disclaimer

• This is time travel! Situations might have changed by today.

• This is about my experience not the total truth - yes there is a chance I was double wrong!

• I don't want to shame any technology, it is just about my experience on applying them to a specific problem.

• No dogs were harmed in the making!
• What is FiFo? - Open Source Cloud orchestration
  
  • For SmartOS: ZFS, DTrace, Crossbow, Zones, …
  
  • In Erlang: Distributed, fault tolerant, fun to write, …
The fail of Clojure Script
What was done

• CLJS app in GZ

• HTTP API
Reason

• existing client for the API

• node.js was on the GZ (looked like additional deps).

• Wanted to try Clojure Script.

• No idea of what Project FiFo would become.
The problem

• lots of dependencies (version conflicts, missing libraries).

• at that time very hard to debug (no source maps etc., lack of visibility/horrible stack traces).

• Everything in the Global Zone. (big footprint)

• Only one system
What I learned

• Try to plan what you do before you do it.

• Rewriting is no shame!

• What seems easy in the beginning is not always the right thing.
The fail of a single host
What changed

• Added wiggle, API endpoint over multiple cljs application

• running in a zone

• Allow more then 1 hypervisor!
Reason

- Needed good abstract over the existing code.
- A web interface for the clojurescript code.
- Wanted to work with Erlang.
The problem

- HTTP between wiggle and cljs-app.
- Single point of failure.
- Did not simplify the code on the hypervisor; it just forwarded.
- Still not enough separation.
- Authentication handled downstream in cljs.
  - Synchronization is a pain.
What I learned

• HTTP is not the silver bullet.
• Split out applications.
• Modularize (not only in code, but in applications).
• Handle things like authentication as high up as possible.
• Remove work from leaves that should be handled in a different layer.
The fail of distribution
What changed

• Split out authentication -> snarl

• Split out most logic -> sniffle

• Reduced GZ footprint -> scrap cljs replace by minimal erlang app
Reason

- Erlang apps are wonderfully self-contained (releases)
- Distributing systems protects against SPOF
- Separating concerns
  - management on system
  - authentication
  - API
  - Management of hypervisors
Problem

• Synchronization is really hard

• 1st try: gproc had problems with multiple nodes

• 2nd try: wrapper around grpoc -> had a SPOF

• lots of configuration needed with connecting all the systems
What I learned

• distributed systems are hard, who would have thought that!

• managing configuration is annoying, especially in a multi-node environment.

• in Erlang land there are great libraries for distribution.

• riak_core rocks!
The fail of storing JSON

```json
{
    "dataset": "4b6c9c1e-ab43-11e3-b6af-0799fb0203af",
    "description": "Graphite Instance with Carbon Cache and Webinterface",
    "image_size": 0,
    "imported": 1,
    "name": "graphite",
    "networks": [
        {
            "description": "public",
            "name": "net0"
        }
    ],
    "os": "smartos",
    "status": "imported",
    "type": "zone",
    "users": [
        {
            "name": "root"
        },
        {
            "name": "admin"
        }
    ],
    "version": "13.2.1"
}
```
Reason

• It’s “easy”, no schema, good library support for serializing and deserializing

• The fronted/UI used it anyway

• everyone uses JSON, so it must be good right?
Problem

• Choice based on popularity not common sense
• No Pattern matching
• No good libraries to manipulating JSX-JSON
• Verbose and ‘big’
• hard to represent data in Erlang (esp. maps/objects)
• Hard to synchronize/merge (state box[^2] is only a partial solution)
What I learned

• Model data around the backend not the front-end
• JSON is no silver bullet, it has the same problem XML had, it is used for the sake of being used
• CRDT’s are a lovely thing\(^4\)
• Records are not perfect but a very nice storage for structured information
The fail of CAP
Reason

• riak_core really rocks!

• Eventual consistency is a very tempting concept

• Availability is more important than consistency when managing a cloud
Problem

• Expect when it is not, like IP assignment, memory constraints on server :( 

• Globally locking those things would break availability

• Not beating CAP anytime soon [3] :( 

What I learned

• The more control you have over your data the further you can push the ‘eventual’ in eventual consistency

• Locks don’t have to be global need to just cover enough to ensure consistency

• The locks location matters:
  • Hypervisor memory on the hypervisor itself
  • IP’s ‘sharded’ over the ring
Links

• https://project-fifo.net
• https://docs.project-fifo.net