USING BARREL TO BUILD YOUR OWN P2P DATA PLATFORM
- benoît chesneau
- craftsman working on P2P and custom data endpoints solution
- **enki multimedia**: the corporate interface
- member of the **Erlang Industrial User Group**
micromobile services
1. Does my service do only one thing?
2. Is my service autonomous?
3. Does this service own its own data?

a good micro-service?
- **standard solution**: client call a webservices to query and update the data

- **problem**: if connection is slow or absent the microservice stops

**sharing data**
sharing data

- **local storage** replicated always available
- eventually consistent

cloud storage

synchronize

update and query

microservice
barrel

Bring and keep a view of your data near your application
library embedded in your Erlang application(*)

available as a micro-service via HTTP(1,2)

operation friendly (Wombat, OpenCensus)

(*) including elixir or life
a database focusing on simplicity

document oriented

Automatic indexing

push/pull replication

stream changes

Focusing on simplicity
Document oriented

{ "id": "someid", "field1": "fieldvalue1", "_rev": "1-..." }
```javascript
1> Doc = `{ 
1>  "id" => "someid", 
1>  "field1" => "fieldvalue1"
1> },
1> Barrel = "test",
1> barrel_store_sup:start_store(default, barrel_memory_storage, {}),
1> barrel:create_barrel(barrel, {}),
1> {ok, DocId, RevId} = barrel:save_doc(barrel, Doc).
06:04:24.498 [info] start db: ref="test", type=create, options={} 
06:04:24.523 [info] update doc id="someid" rev=>>, deleted=false revtree=>> 
{ok,<<"someid">>, 
   "1-3385e5507c1a801ee9df79400f3688437c7ad5cd41fe0c7e3a23f12f3aa045c5"}>>
2> barrel:fetch_doc(barrel, DocId, {}).
{ok,#{"_rev" => 
   "1-3385e5507c1a801ee9df79400f3688437c7ad5cd41fe0c7e3a23f12f3aa045c5"},
   "field1" => "fieldvalue1", "id" => "someid"}}
```
<table>
<thead>
<tr>
<th><strong>Databases</strong></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GET</strong> /dbs</td>
<td></td>
<td>Get list of available databases</td>
</tr>
<tr>
<td><strong>POST</strong> /dbs</td>
<td></td>
<td>Create a new database</td>
</tr>
<tr>
<td><strong>DELETE</strong> /dbs{database}</td>
<td></td>
<td>Delete a database</td>
</tr>
<tr>
<td><strong>GET</strong> /dbs{database}</td>
<td></td>
<td>Get the database informations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Documents</strong></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GET</strong> /dbs{database}/docs</td>
<td></td>
<td>Get list of all available documents</td>
</tr>
<tr>
<td><strong>POST</strong> /dbs{database}/docs</td>
<td></td>
<td>Create a new document</td>
</tr>
<tr>
<td><strong>DELETE</strong> /dbs{database}/docs{docid}</td>
<td></td>
<td>Delete a document</td>
</tr>
<tr>
<td><strong>GET</strong> /dbs{database}/docs{docid}</td>
<td></td>
<td>Get a document</td>
</tr>
<tr>
<td><strong>PUT</strong> /dbs{database}/docs{docid}</td>
<td></td>
<td>Update a document</td>
</tr>
<tr>
<td><strong>POST</strong> /dbs{database}/revsdiff</td>
<td></td>
<td>Check for differences in documents history</td>
</tr>
</tbody>
</table>

**simple api**
- simple API (P1 version):
  - `save_doc(s)`, `fetch_doc(s)`, `delete_doc(s)`, ....

- multi storage backend:
  - ordered Key/Value stores
  - with snapshot support

multi-backend
MVCC over ETS

- InternalKey(user_key="Key1", seqno=10, Type=Put) | Value = "KEY1_VAL2"
- InternalKey(user_key="Key1", seqno=9, Type=Put)  | Value = "KEY1_VAL1"
- InternalKey(user_key="Key2", seqno=16, Type=Put)  | Value = "KEY2_VAL2"
- InternalKey(user_key="Key2", seqno=15, Type=Delete)| Value = "KEY2_VAL1"
- InternalKey(user_key="Key3", seqno=7, Type=Delete) | Value = "KEY3_VAL1"
- InternalKey(user_key="Key4", seqno=5, Type=Put)    | Value = "KEY4_VAL1"

- writes are done via a single process
- garbage collection is handled in background via a process

- https://gitlab.com/barrel-db/memstore
- persisted on disk using rocksdb
- dirty NIF
- https://gitlab.com/barrel-db/erlang-rocksdb.git
- pure Erlang disk storage over leveled (https://github.com/martinsumner/leveled)?
- {"headquarters": "belgium" } has path /headquarters/belgium

- {"exports": [{"city": "Moscow"}, {"city": "Athens"}]} has path /exports/[]/city/Moscow and /exports/[]/city/Athens.
indexed as a tree
{ "id": "someid", "field1": "fieldvalue1", "_rev": "1-..." }
409
Conflict
a revision tree: conflict is stored on a new branch (based on hash histories https://dl.acm.org/citation.cfm?id=851951 )

- revision id : (level)-hash(Doc, Parent, Deleted)

- a winning when the leaf is not a deleted tombstone

- longest branch is the winner

- all revisions are replicated

multiple revisions
List or subscribe on document changes

Changes are received as events:
{changes, Stream, Changes, LastSeq}

Changes are received by batch at an interval set during subscription

filter changes

changes stream
change document

\[
\{ 
\text{"id"} : \text{"someid"}, \\
\text{"seq"} : 1, \\
\text{"rev"} : \text{"1-..."}, \\
\text{"doc"} : \#\}
\]

document Id
revision of the document
change sequence
p2p
- replicate the data
- aggregate the data
- discover & find the data

a p2p data platform
push/pull (master-master) replication
replicate all the document revisions between nodes
based on the changes stream
- every peers fork the master, updates are offline
- peers pull and merge from the main server
- peers keeps a partial view of the database (filtered replication)

replication scenarios
- released in Q2
- platform tested with partisan (but not only)
- nat traversal support: https://github.com/benoitc/erlang-nat
- https://barrel-db.org
- https://gitlab.com/barrel-db
- contact me @benoitc