Building resilient infrastructure with CouchDB

Tim Perry

*Tech Lead & Open-Source Champion at **Softwire***

tim-perry.co.uk **@pimterry** **github.com/pimterry**


softwire

CouchDB

relax

Document Store

```
{
  "_id": "my-document-example",
  "_rev": "21-qwe123asd",

  "some-content": {
    "a": 1,
    "b": 2
  },

  "a list!": [3, 4, 5]
}
```

HTTP API

```
$ curl -X GET http://couchdb:5984/my-db/a-doc-id

{"_id": "a-doc-id"
 "_rev": "4-9812eojawd"
 "data": [1, 2, 3]}
```

HTTP API

```
$ curl -X PUT http://couchdb:5984/my-db/another-id \
       -H 'Content-Type: application/json' \
       -d '{ "other data": 4 }'

{"ok":true,
 "id":"another-id",
 "rev":"1-2902191555"}
```

Replication

```
# Pull from B > A
$ curl -X POST http://couchdb-A:5984/_replicator \
       -H 'Content-Type: application/json' \
       -d '{ "source": "http://couchdb-B:5984/demo-db",
             "target": "demo-db",
             "continuous": true }'

# Pull from A -> B
$ curl -X POST http://couchdb-B:5984/_replicator \
       -H 'Content-Type: application/json' \
       -d '{ "source": "http://couchdb-A:5984/demo-db",
             "target": "demo-db",
             "continuous": true }'
```

Indexed Views

Incremental Map/Reduce

ACID (locally)

Erlang-based

Web UI

Show Functions

Filters

Validation

Resilient Infrastructure

Let's break everything!

```
while true
do
  curl -X POST 'http://couchdb-A:5984/demo-db' \
       -H "content-type: application/json" \
       -d '{ "created_at": "'"`date`"'" }' \
       --max-time 0.1

  curl -X POST 'http://couchdb-B:5984/demo-db' \
       -H "content-type: application/json" \
       -d '{ "created_at": "'"`date`"'" }' \
       --max-time 0.1
done
```

```
while true
do
  vagrant halt couchdb-a --force
  sleep 30
  vagrant up couchdb-a --no-provision
  sleep 30

  vagrant halt couchdb-b --force
  sleep 30
  vagrant up couchdb-b --no-provision
  sleep 30
done
```

(Some console logging omitted)

Is this useful?

Real World Example

(Anonymized)

B2B SaaS product, with strict SLAs

Millions of paying daily users

3,000 servers across 25 datacentres

50,000 requests per second, average

Highly latency sensitive

Every request needs the (readonly) user session

Bonus Challenges

Struggling network infrastructure

Frequent loss of connection to datacentres

Occasional power outages in datacentres

Users can and do roam, worldwide

Server failover is always to a different datacentre

Data centres have hub & spoke connectivity only (through London)

Previous Solution

Hold all user sessions in memory on every server

Announce new sessions to every server with a central message queue

Canonical store kept in a single RDBMS (for server initialisation)

Real World Problems

Memory usage doesn't scale

Network and server failures are big problems

Message queue failures are catastrophic problems

CouchDB Solution

Small LRU cache in every server

CouchDB in every datacentre

CouchDB in the central datacentre

Hub & spoke replication

Servers query local CouchDB by default, or fall back to central CouchDB

Real World Improvements

No single point of failure

Scales horizontally easily

Major memory savings

Some Challenges

Ops ramp-up

Support service setup

Disk usage

Hoodie

http://hood.ie

Hoodie

No Backend

Offline-First

Hoodie

Save data

```
$('.addTask .submit').click(function () {
    var desc = $('.addTask .desc').val();
    hoodie.store.add('task', { desc: desc });
});
```

Hoodie

Handle new data

```
hoodie.store.on('add:task', function (task) {
    $('.taskList').append('<li>'+task.desc+'</li>');
});
```

Hoodie

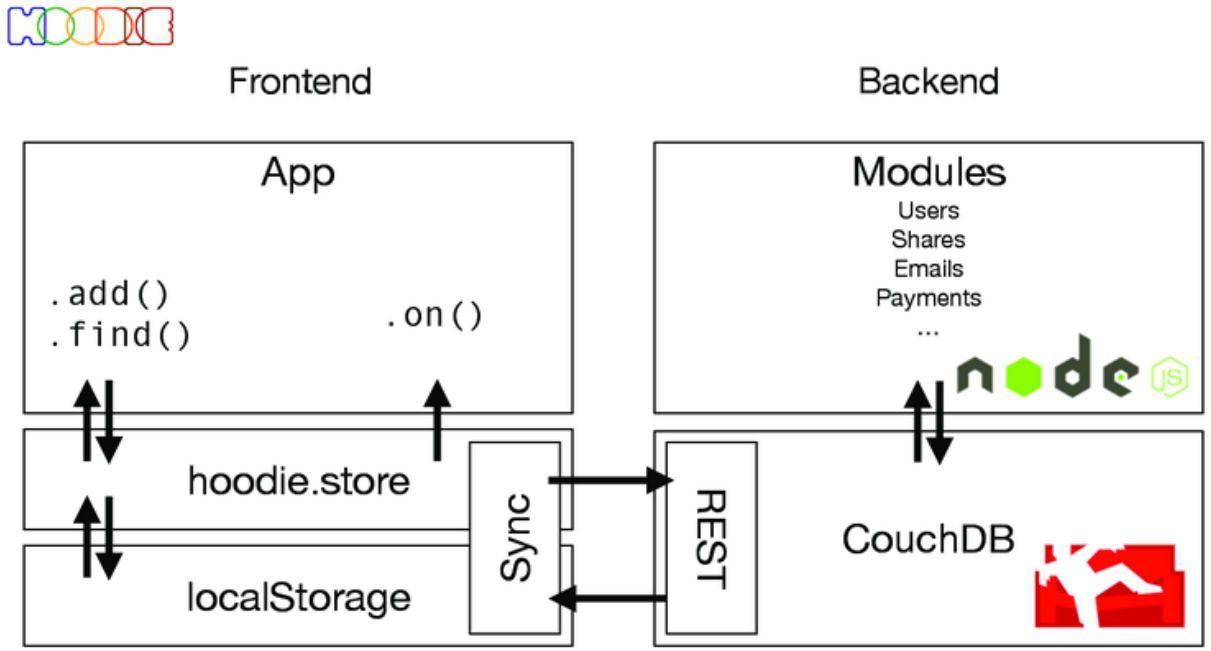
Log in users

```
$('.login').click(function () {
    var username = $(".username").val();
    var password = $(".password").val();

    hoodie.account.signIn(username, password)
                  .done(loginSuccessful);
});
```

Hoodie

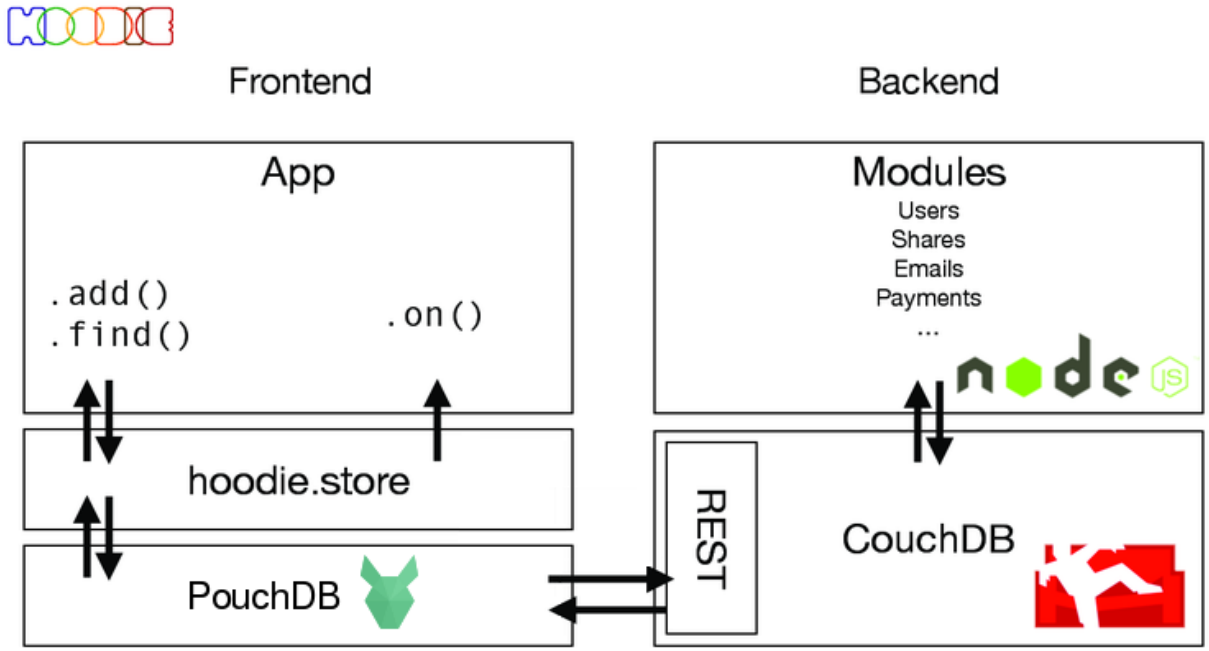
Architecture



(From the Hoodie team at http://hood.ie/intro#magic, CC-BY-SA-NC)

Hoodie

Future Architecture (Probably)



(Modified, from the Hoodie team's diagram at http://hood.ie/intro#magic, CC-BY-SA-NC)

Why does any of this work?

Reliable Replication

Multiversion Concurrency Control

(or MVCC)

Reliable Replication

The Changes Feed

```
$ curl -X GET http://couchdb:5984/my-db/_changes?since=1

{ "results": [
  {"seq":2,"id":"my-doc","changes":[{"rev":"1-128qw99"}]},
  {"seq":3,"id":"my-doc","changes":[{"rev":"2-98s9123"}]},
], last_seq: 3}
```

Reliable Replication

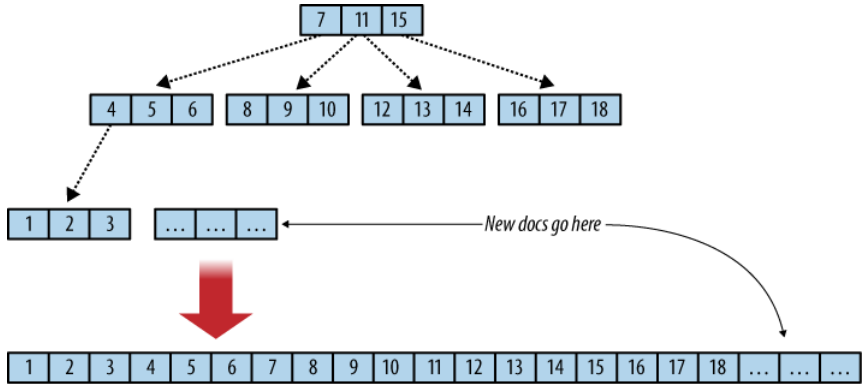
Replication Process

1. Track the source's sequence number in a local-only metadata document in the target DB, unique to this replication, set to 0 initially
2. Read the changes from the source, since the sequence id stored in the local document in the target
3. Read any missing document revisions from the source DB
4. Write these updates to the target DB
5. Update the sequence number tracked in the target
6. Go to 2

(Paraphrased from http://replication.io)

Append-Only B+ Trees

Append-Only B+ Trees



(From 'CouchDB: The Definitive Guide', CC-BY)

Did we break everything?

```
while true
do
  curl -X POST 'http://couchdb-A:5984/demo-db' \
       -H "content-type: application/json" \
       -d '{ "created_at": "'"`date`"'" }' \
       --max-time 0.1

  curl -X POST 'http://couchdb-B:5984/demo-db' \
       -H "content-type: application/json" \
       -d '{ "created_at": "'"`date`"'" }' \
       --max-time 0.1
done
```

```
while true
do
  vagrant halt couchdb-a --force
  sleep 30
  vagrant up couchdb-a --no-provision
  sleep 30

  vagrant halt couchdb-b --force
  sleep 30
  vagrant up couchdb-b --no-provision
  sleep 30
done
```

(Some console logging omitted)

Phew.

CouchDB is not perfect

But '*always* available' is a great superpower

Any questions?

Tim Perry

*Tech Lead & Open-Source Champion at **Softwire***

tim-perry.co.uk **@pimterry** **github.com/pimterry**