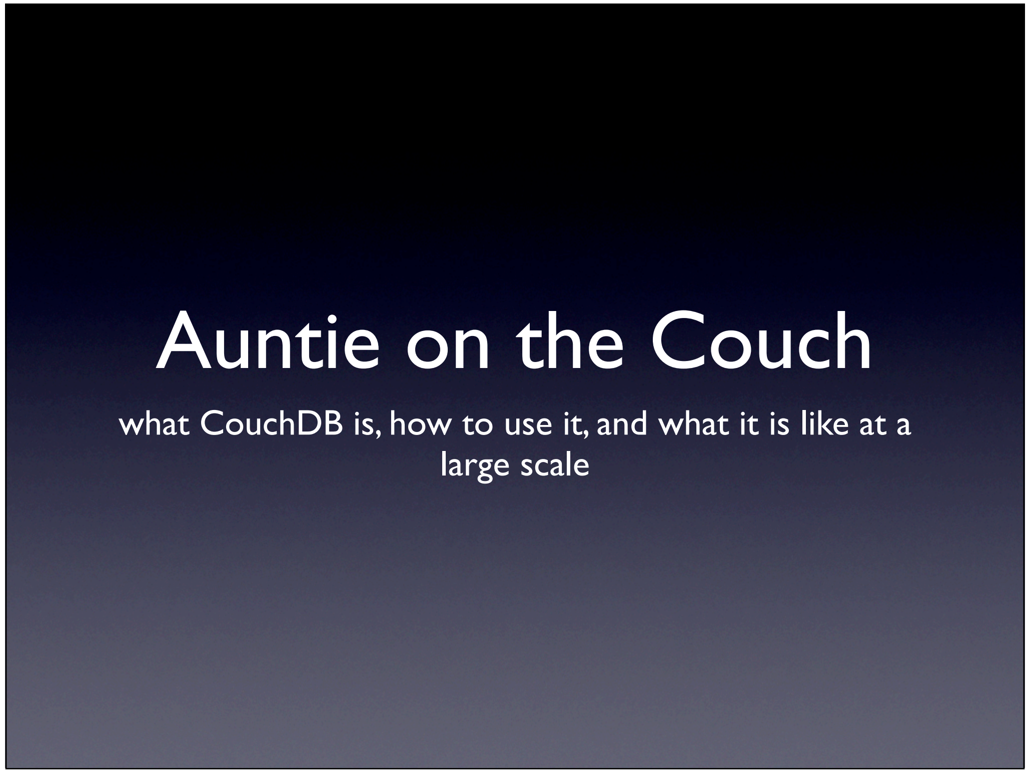
BBC

Enda Farrell

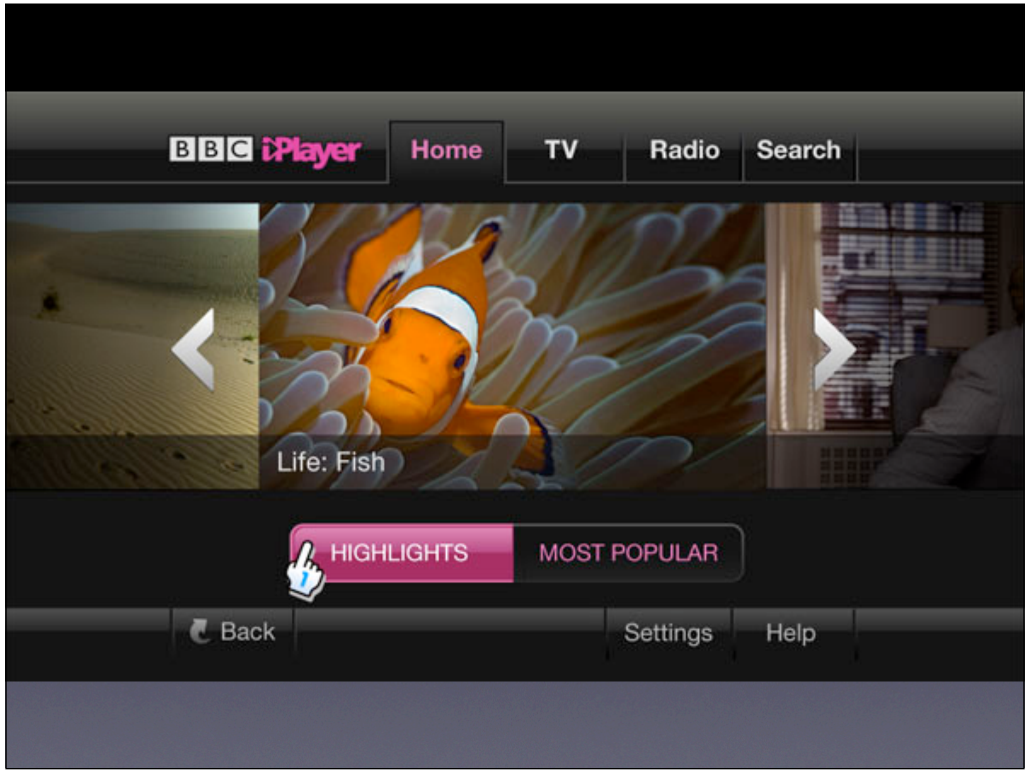
Software architect, product owner and lead developer for the BBC's usage of CouchDB



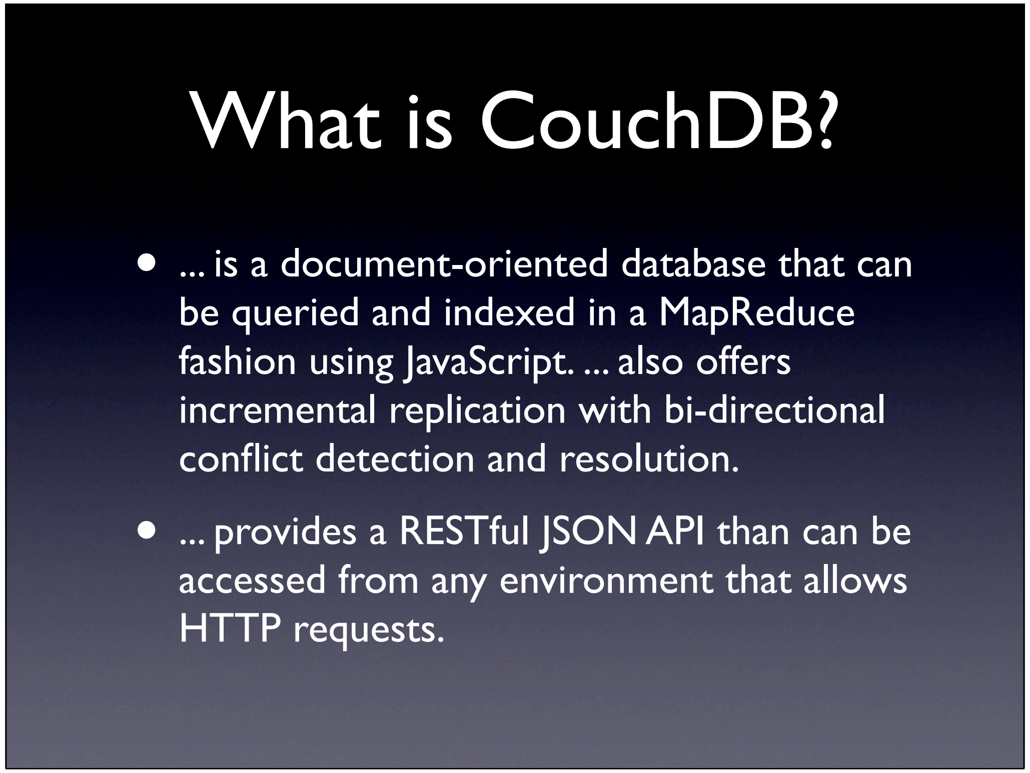
Auntie on the Couch

what CouchDB is, how to use it, and what it is like at a large scale

A little context before I start. I expect most of you have come across the BBC and it’s web site. It’s big, there are popular parts and there are obscure parts. Which are backed in some way by CouchDB?



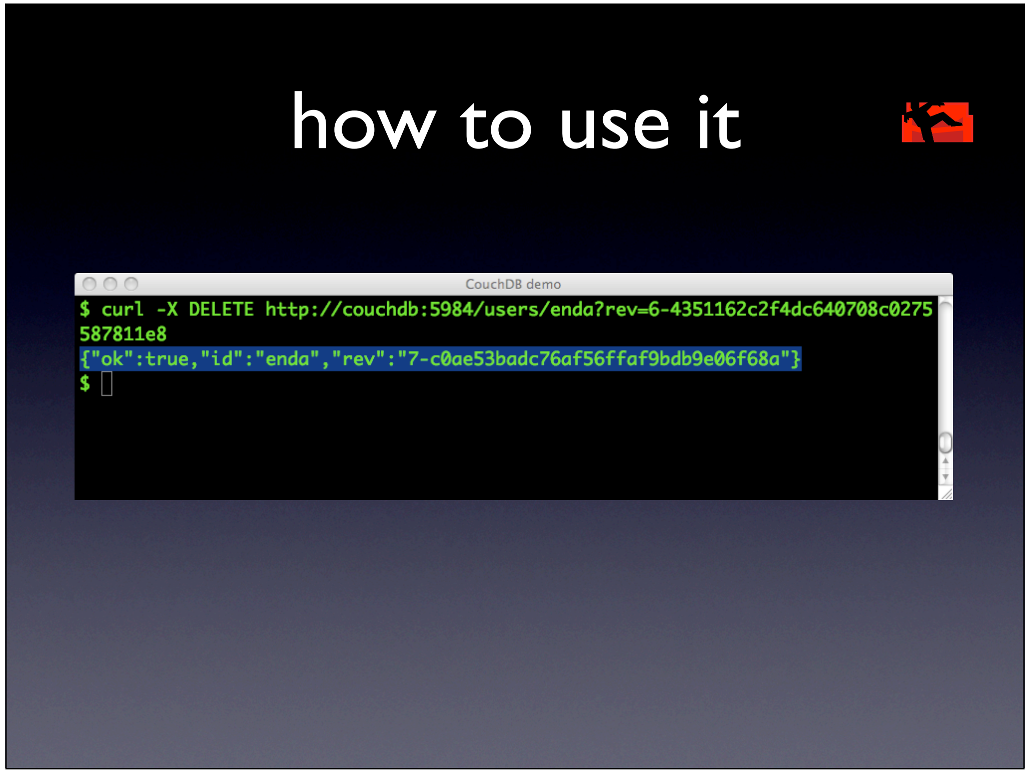
So - which “little” sites are using CouchDB? Might I have ever come across them? Do they matter? Would anyone notice if they disappeared? ;-) Someone might ;-)



What is CouchDB?

- ... is a document-oriented database that can be queried and indexed in a MapReduce fashion using JavaScript. ... also offers incremental replication with bi-directional conflict detection and resolution.
- ... provides a RESTful JSON API than can be accessed from any environment that allows HTTP requests.

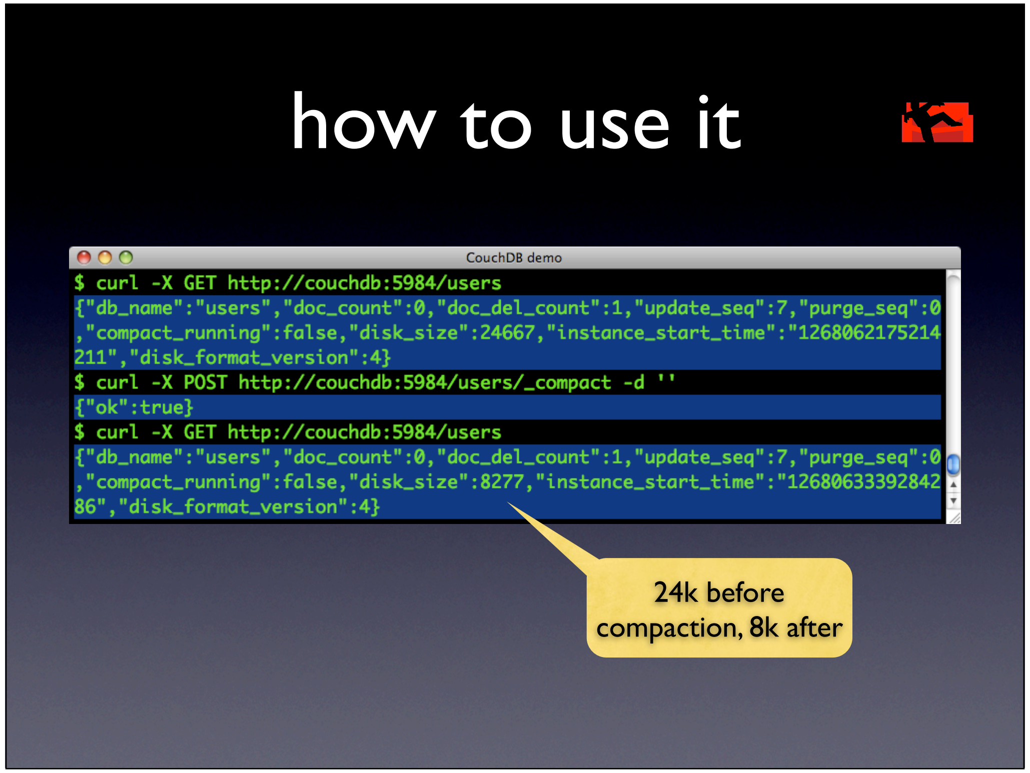
So (almost) goes the introduction from http://couchdb.apache.org/
Let’s skip the text and have a look at CouchDB in action



how to use it

```
$ curl -X DELETE http://couchdb:5984/users/enda?rev=6-4351162c2f4dc640708c0275
587811e8
{"ok":true,"id":"enda","rev":"7-c0ae53badc76af56ffaf9bdb9e06f68a"}
$ 
```

CouchDB uses standard HTTP RESTful commands - GET, PUT, POST and DELETE to access data. It uses a JSON format. Updating an existing document _requires_ having the current revision of that document which stops accidental over-writing of data by clients.



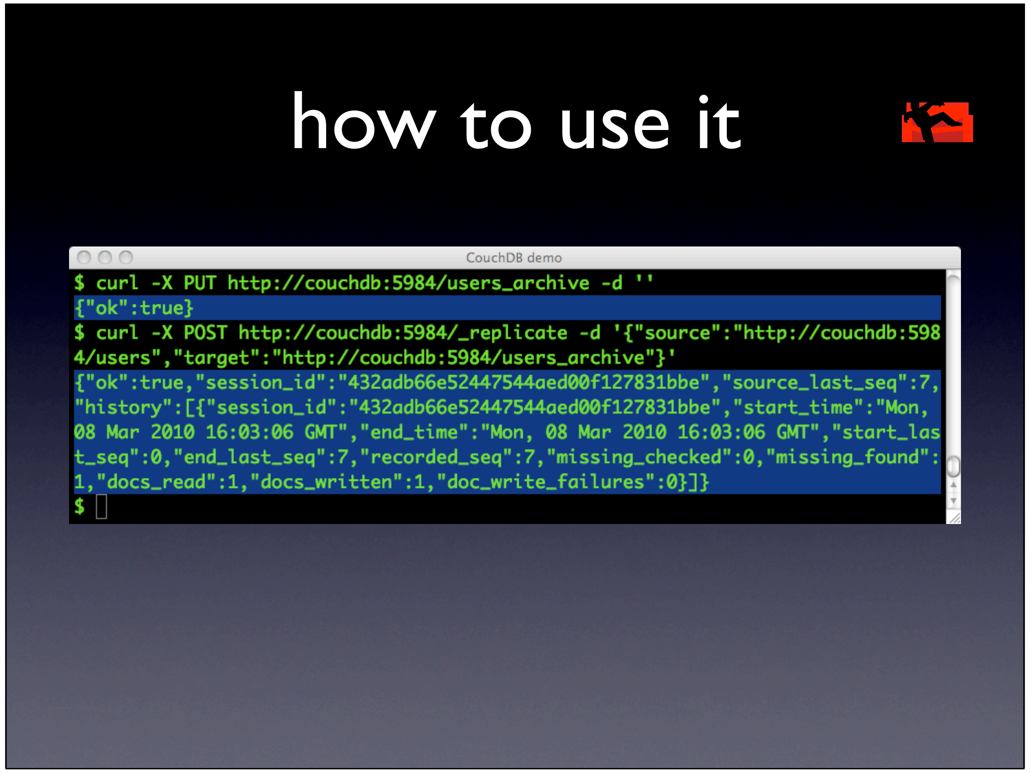
how to use it

CouchDB demo

```
$ curl -X GET http://couchdb:5984/users
{"db_name":"users","doc_count":0,"doc_del_count":1,"update_seq":7,"purge_seq":0
,"compact_running":false,"disk_size":24667,"instance_start_time":"1268062175214
211","disk_format_version":4}
$ curl -X POST http://couchdb:5984/users/_compact -d ''
{"ok":true}
$ curl -X GET http://couchdb:5984/users
{"db_name":"users","doc_count":0,"doc_del_count":1,"update_seq":7,"purge_seq":0
,"compact_running":false,"disk_size":8277,"instance_start_time":"12680633392842
86","disk_format_version":4}
```

24k before compaction, 8k after

Compacting databases removes from disk the old, over-written versions of documents. In our setup, we (a) don’t often care about old versions and (b) we like saving space. This space saving can be significant depending on how many updates are done to documents.



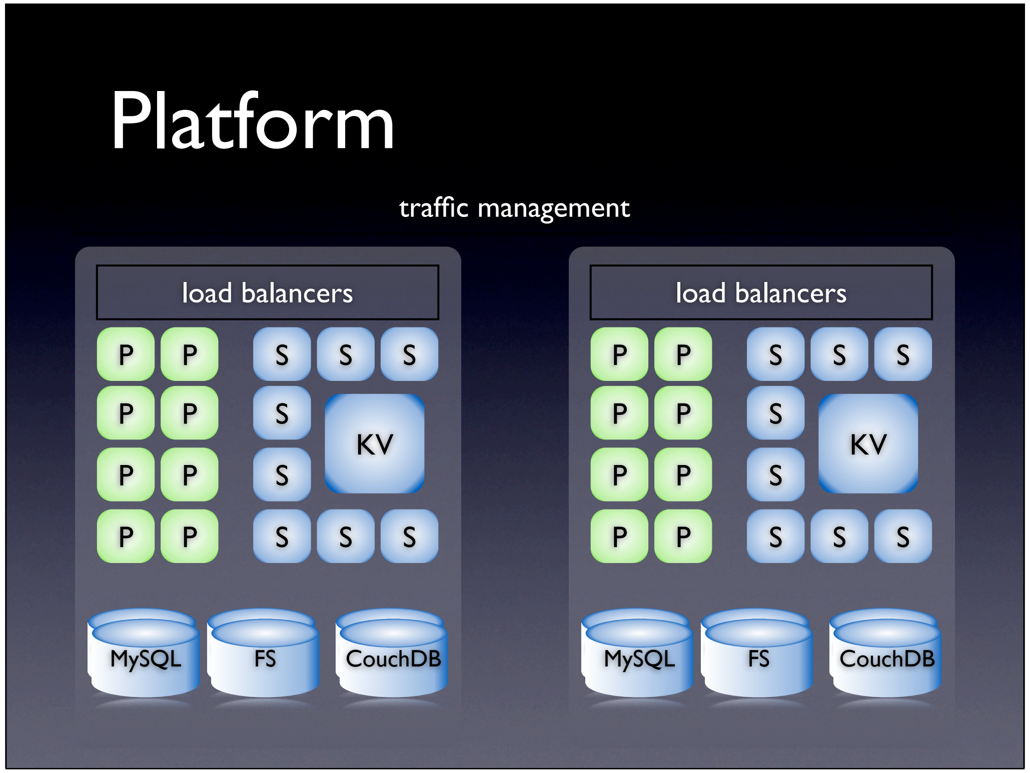
how to use it

```
$ curl -X PUT http://couchdb:5984/users_archive -d ''
{"ok":true}
$ curl -X POST http://couchdb:5984/_replicate -d '{"source":"http://couchdb:598
4/users","target":"http://couchdb:5984/users_archive"}'
{"ok":true,"session_id":"432adb66e52447544aed00f127831bbe","source_last_seq":7,
"history":[{"session_id":"432adb66e52447544aed00f127831bbe","start_time":"Mon,
08 Mar 2010 16:03:06 GMT","end_time":"Mon, 08 Mar 2010 16:03:06 GMT","start_las
t_seq":0,"end_last_seq":7,"recorded_seq":7,"missing_checked":0,"missing_found":
1,"docs_read":1,"docs_written":1,"doc_write_failures":0}]}
$ 
```

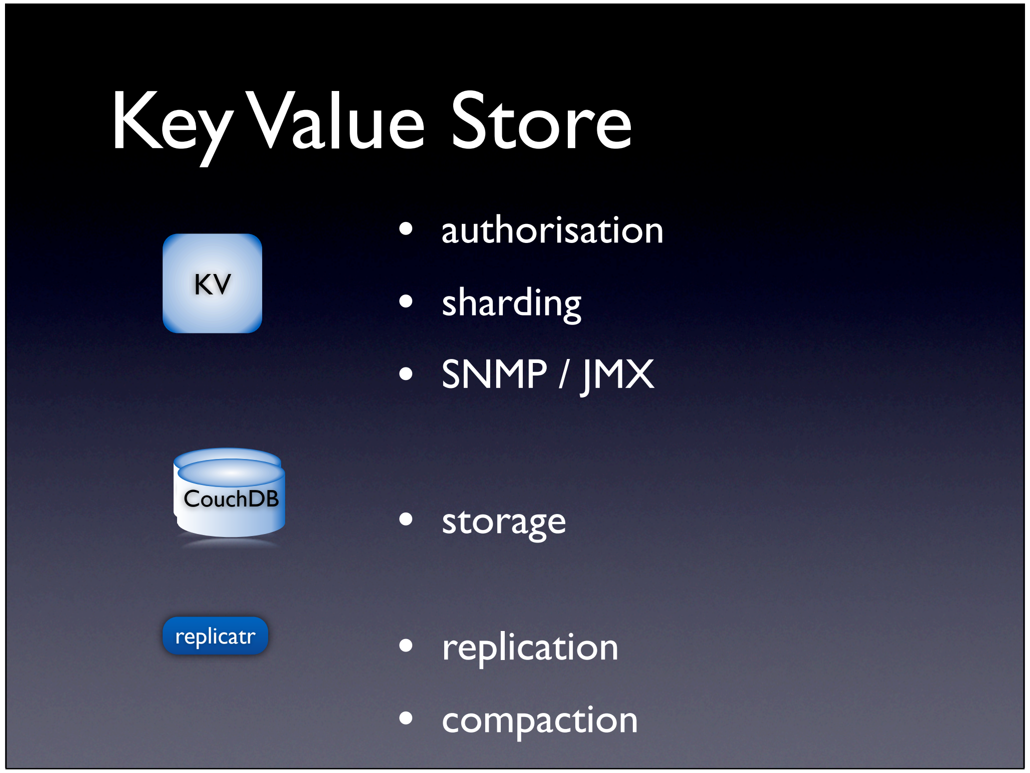
This is the old “trigger” replication which has been improved on in 0.10. Notice that even through CouchDB has an admin UI - _all_ commands to the service - like this “go replicate these” - are RESTful HTTP calls.

what it is like at scale

- Context - one service on a new platform
- Operations
- Replication and compaction
- Some statistics
- How we use it, how we don’t



(mutually authenticating) secure services (with a small “s”) oriented architecture. It’s not the “XML, SOAP, WSDL, UDDI” version of SOA - it is lighter, easier to code to, quicker, easier to scale and easier to manage. “P” are PHP applications assembling data. “S” are JSON/XML service providers.

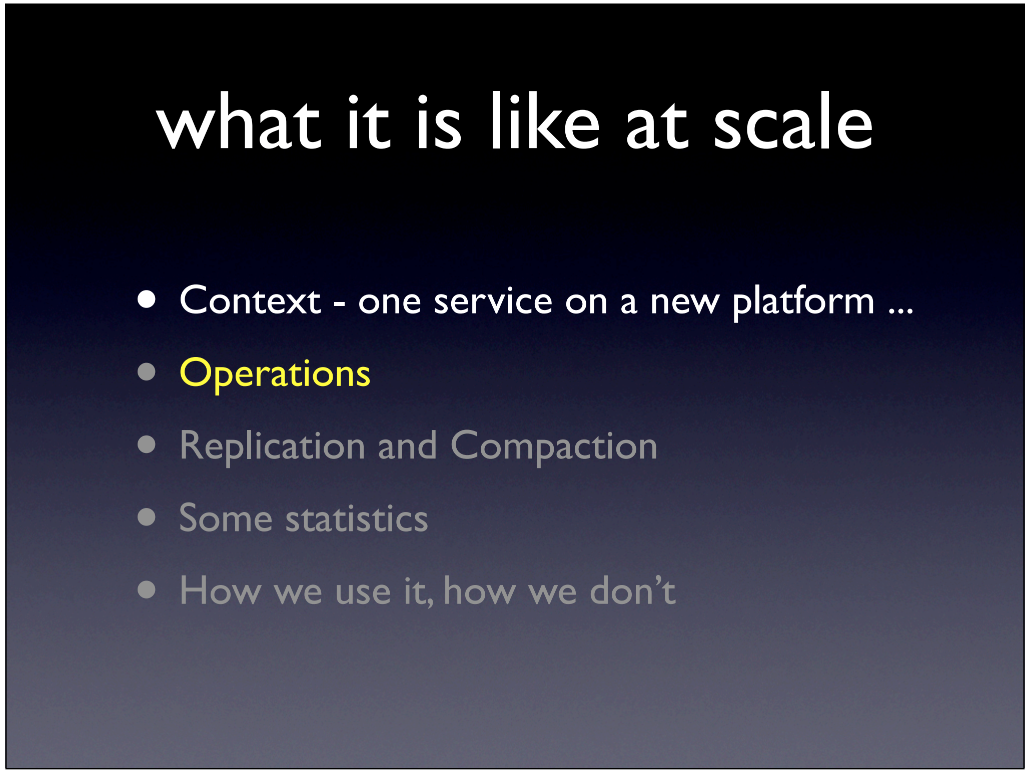


Key Value Store

- authorisation
- sharding
- SNMP / JMX
- storage
- replication
- compaction

KV
CouchDB
replicatr

To make CouchDB “fit” into our platform, we put a wrapper API above it, and to make operations simple, we put a “replication daemon” underneith.

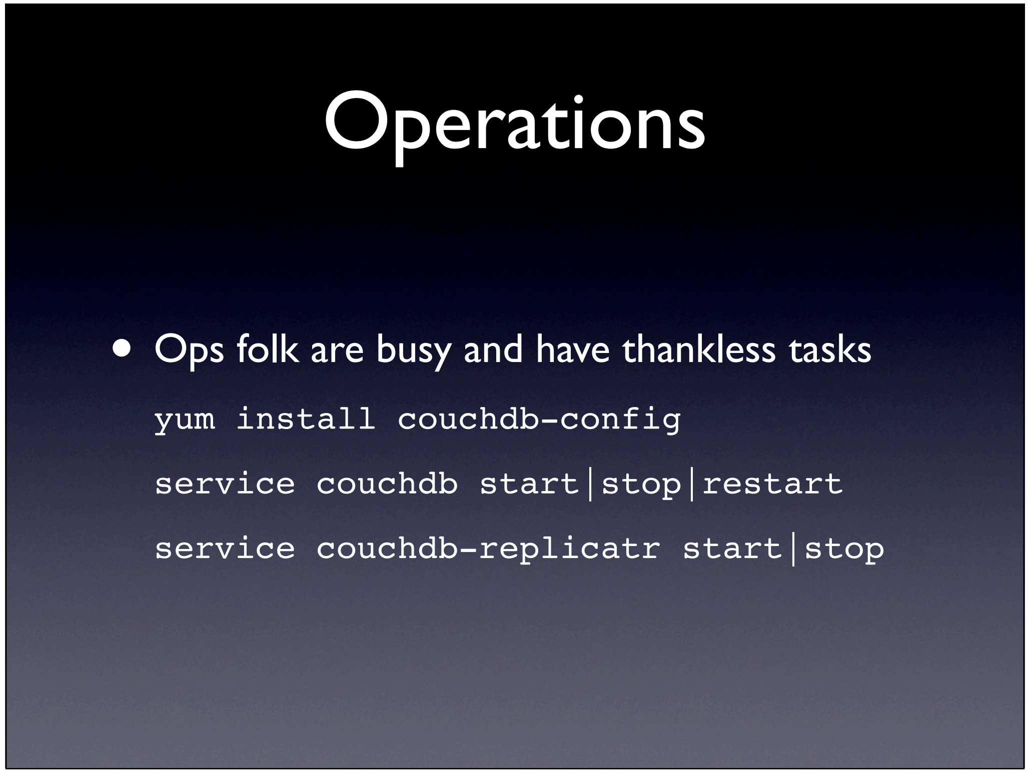


what it is like at scale

- Context - one service on a new platform ...
- Operations
- Replication and Compaction
- Some statistics
- How we use it, how we don’t

Operations

- Installation and running
- Instances and system utilisation
- Scalability



Operations

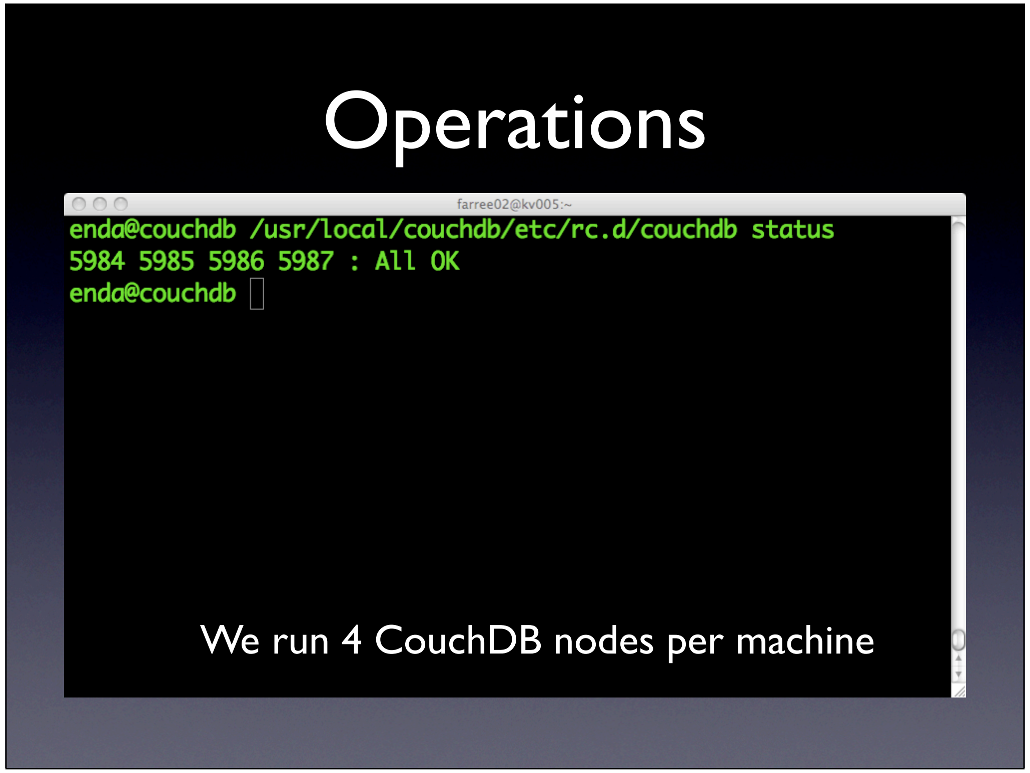
- Ops folk are busy and have thankless tasks

```
yum install couchdb-config

service couchdb start|stop|restart

service couchdb-replicatr start|stop
```

We did a little work in packing RPMs and made CouchDB look act and "smell" like any other service on the platfrom

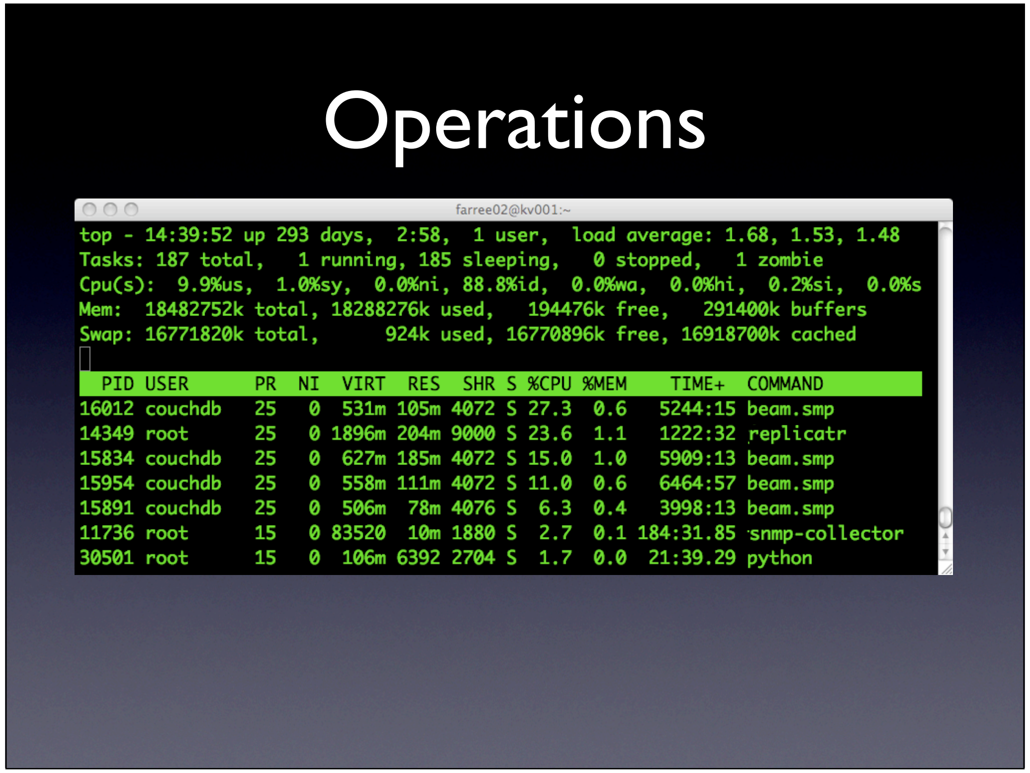


Operations

```
enda@couchdb /usr/local/couchdb/etc/rc.d/couchdb status
5984 5985 5986 5987 : All OK
enda@couchdb
```

We run 4 CouchDB nodes per machine

Apart from specifying IP bindings, database directories etc, the only “customisation” we have is to spin up (and down) 4 nodes per physical machine



Operations

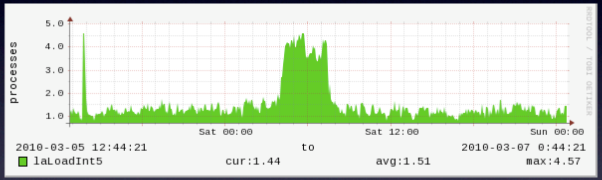
farree02@kv001:~

```
top - 14:39:52 up 293 days,  2:58,  1 user,  load average: 1.68, 1.53, 1.48
Tasks: 187 total,   1 running, 185 sleeping,   0 stopped,   1 zombie
Cpu(s):  9.9%us,  1.0%sy,  0.0%ni, 88.8%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%s
Mem:  18482752k total, 18288276k used,   194476k free,   291400k buffers
Swap: 16771820k total,      924k used, 16770896k free, 16918700k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
16012 couchdb   25   0  531m 105m 4072 S 27.3  0.6   5244:15 beam.smp
14349 root      25   0 1896m 204m 9000 S 23.6  1.1   1222:32 replicatr
15834 couchdb   25   0  627m 185m 4072 S 15.0  1.0   5909:13 beam.smp
15954 couchdb   25   0  558m 111m 4072 S 11.0  0.6   6464:57 beam.smp
15891 couchdb   25   0  506m  78m 4076 S  6.3  0.4   3998:13 beam.smp
11736 root      15   0 83520  10m 1880 S  2.7  0.1 184:31.85 snmp-collector
30501 root      15   0  106m 6392 2704 S  1.7  0.0  21:39.29 python
```

8 cores, 16 GB RAM. CouchDB is mostly kind on CPU, and if you do not run views, has a v consistent memory footprint.

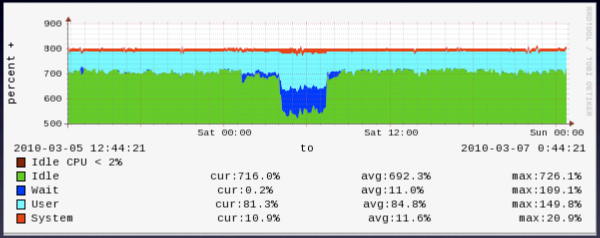
Operations



Low load average

Look - doing backups - which by the way are as simple as “copy the files in these directories” - has a big load effect. Sat/Sun are not “quiet” on the platform - this is essentially the same 7 days a week

Operations



```
2010-03-05 12:44:21                 to                  2010-03-07 0:44:21
■ Idle CPU < 2%
■ Idle                  cur:716.0%        avg:692.3%        max:726.1%
■ Wait                   cur:0.2%         avg:11.0%         max:109.1%
■ User                  cur:81.3%         avg:84.8%         max:149.8%
■ System                 cur:10.9%         avg:11.6%         max:20.9%
```

Kind to CPU

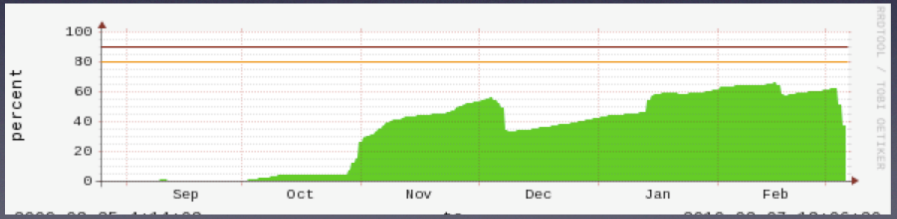
The green is idle time on these graphs.

Operations

- Robust - very robust
- restarts < 1 sec
- no “fix-up” if it crashes - append only B-tree
- No “scheduled downtime” needed to restart

Op ☞ scalability

- Still in our early stages
- We can double and double again our infra with only small rc.d script & DNS changes

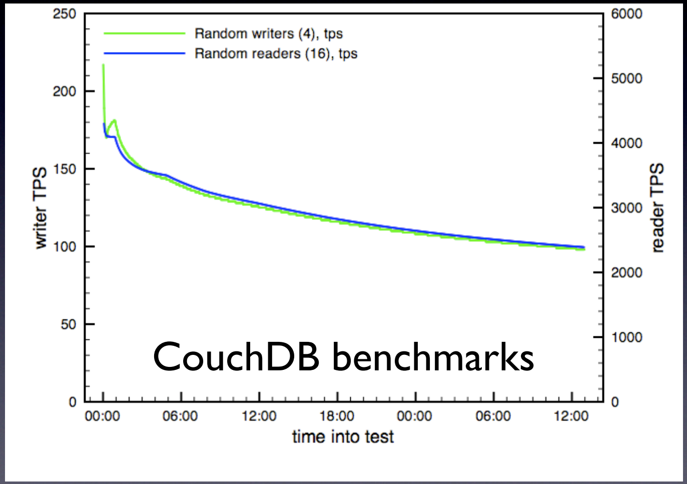


This somewhat shows how we are still “beginning” our scalability journey.

Scalability

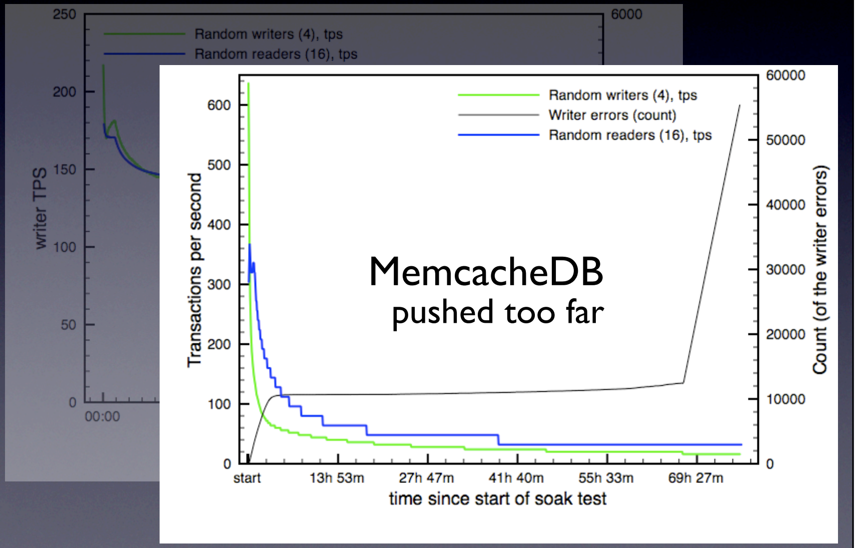
- What do “you” need in the next 12 months?
- If you don’t know, what attributes do you rely on to deal with this?
  - consistency - linear or O(log n) graphs
  - reliable empirical stats
  - known break points - stress tests

Scalability - consistency



Order log n decay of performance with data sizes - watch the blip as we break through the machine’s working set.
We ran out of disk before we hit a break-point of these tests.
Writers finished at 100 tps, readers at 2400 in this test

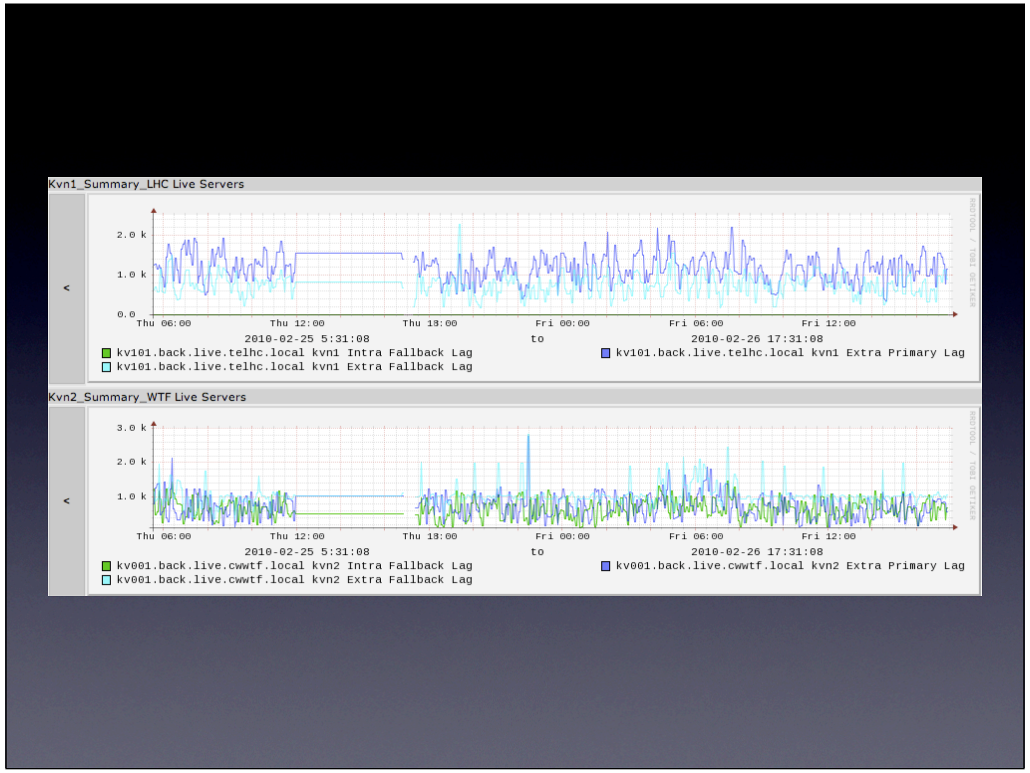
Scalability - consistency



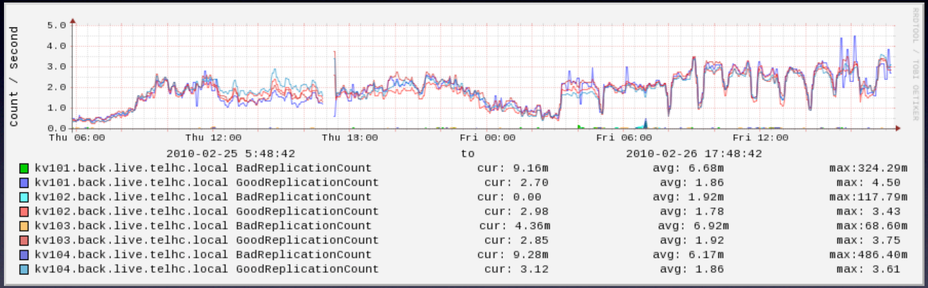
When you push a system too far - like an in-memory DB beyond the working set - you see this sort of graph. Exponential decay, order of magnitude drops beyond the working set, a findable break point beyond which you cannot scale. Writers finished at 40 tps, readers at 60 in this test - though started much better.

Scalability - reliable stats

- Throughout the platform we use SNMP to collect, organise, store and present the data
- We can scale by looking at where we need to - proactively

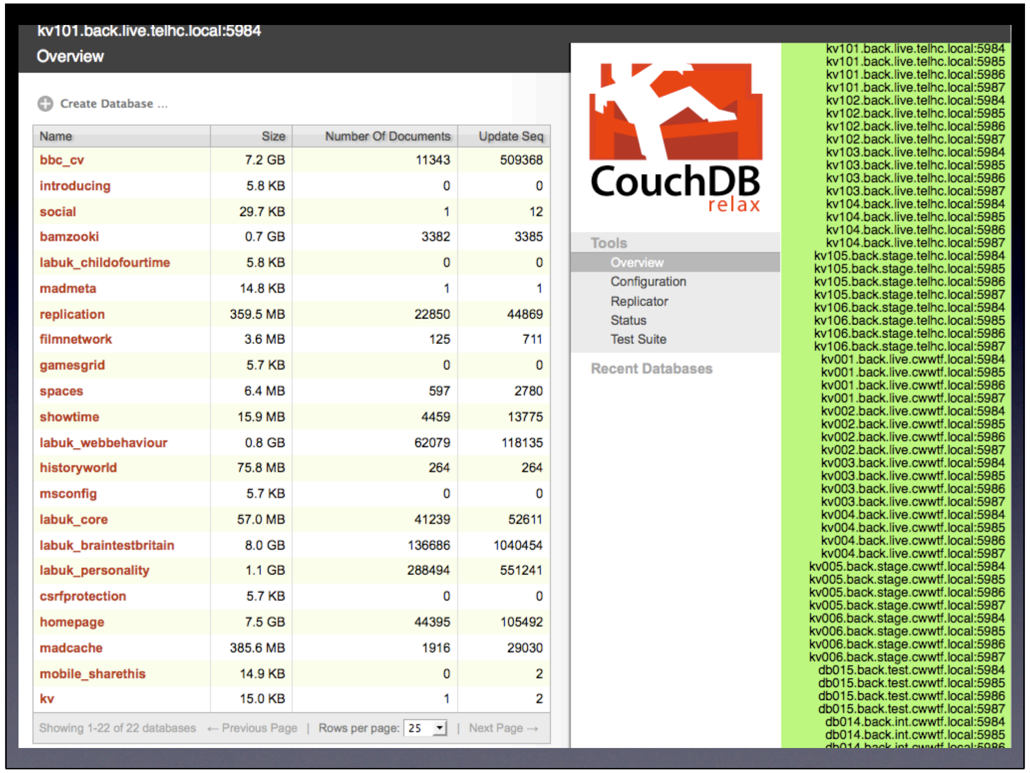


CouchDB access speeds, num accesses, replication lag, counts of http actions, KV access speeds, KV namespace stats, replication stats



```
             2010-02-25 5:48:42                       to                      2010-02-26 17:48:42
■ kv101.back.live.telhc.local BadReplicationCount        cur: 9.16m             avg: 6.68m             max:324.29m
■ kv101.back.live.telhc.local GoodReplicationCount        cur: 2.70              avg: 1.86               max: 4.50
■ kv102.back.live.telhc.local BadReplicationCount        cur: 0.00              avg: 1.92m             max:117.79m
■ kv102.back.live.telhc.local GoodReplicationCount        cur: 2.98              avg: 1.78               max: 3.43
■ kv103.back.live.telhc.local BadReplicationCount         cur: 4.36m             avg: 6.92m             max:68.60m
■ kv103.back.live.telhc.local GoodReplicationCount        cur: 2.85              avg: 1.92               max: 3.75
■ kv104.back.live.telhc.local BadReplicationCount        cur: 9.28m             avg: 6.17m             max:486.40m
■ kv104.back.live.telhc.local GoodReplicationCount        cur: 3.12              avg: 1.86               max: 3.61
```

Summary charts for replication statistics


kv101.back.live.telhc.local:5984

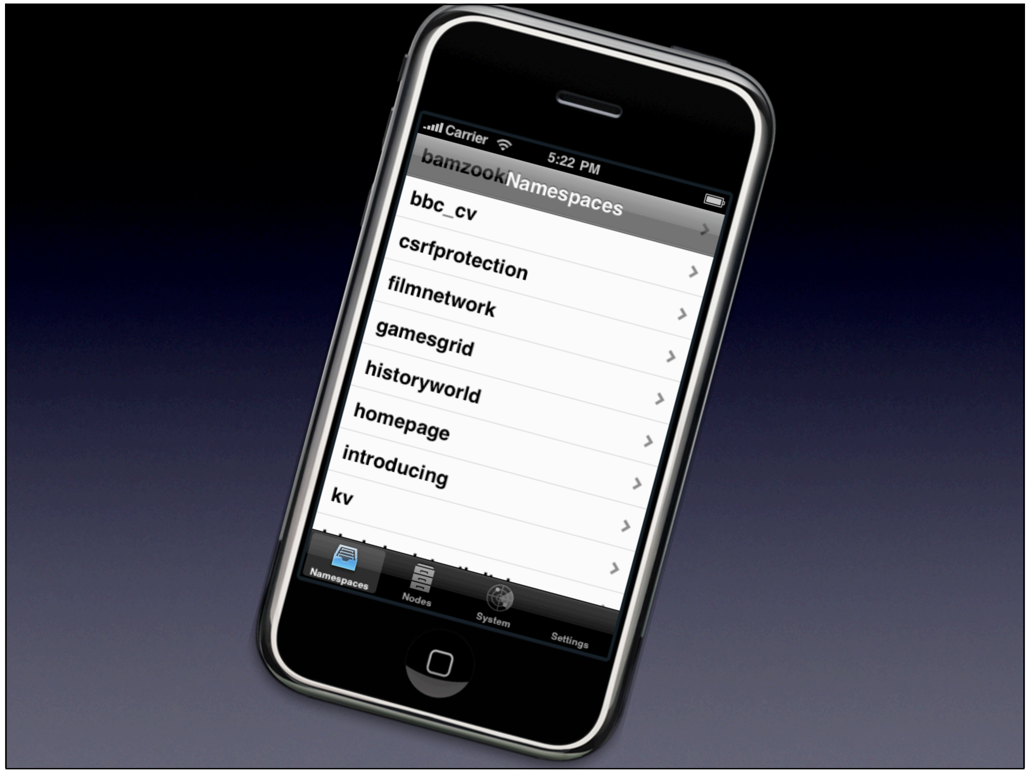
Overview

Name	Size	Number Of Documents	Update Seq
bbc_cv	7.2 GB	11343	509368
introducing	5.8 KB	0	0
social	29.7 KB	1	12
bamzooki	0.7 GB	3382	3385
labuk_childofourtime	5.8 KB	0	0
madmeta	14.8 KB	1	1
replication	359.5 MB	22850	44869
filmnetwork	3.6 MB	125	711
gamesgrid	5.7 KB	0	0
spaces	6.4 MB	597	2780
showtime	15.9 MB	4459	13775
labuk_webbehaviour	0.8 GB	62079	118135
historyworld	75.8 MB	264	264
msconfig	5.7 KB	0	0
labuk_core	57.0 MB	41239	52611
labuk_braintestbritain	8.0 GB	136686	1040454
labuk_personality	1.1 GB	288494	551241
csrfprotection	5.7 KB	0	0
homepage	7.5 GB	44395	105492
madcache	385.6 MB	1916	29030
mobile_sharethis	14.9 KB	0	2
kv	15.0 KB	1	2

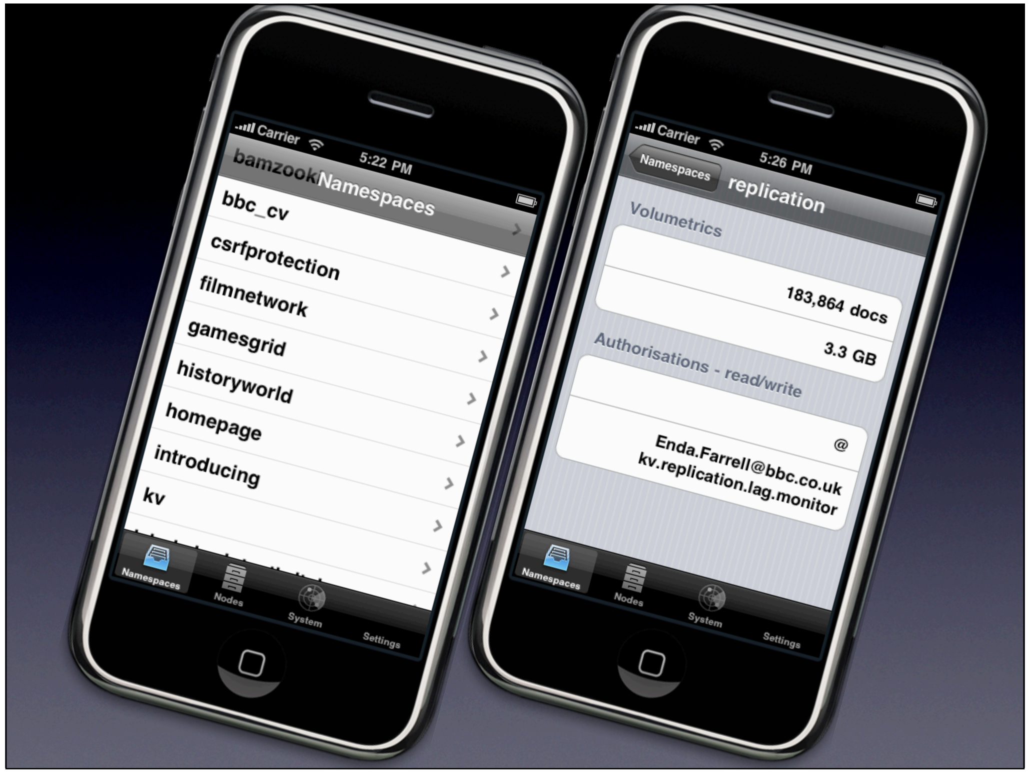
Showing 1-22 of 22 databases ← Previous Page | Rows per page: 25 | Next Page →

kv101.back.live.telhc.local:5984
kv101.back.live.telhc.local:5985
kv101.back.live.telhc.local:5986
kv101.back.live.telhc.local:5987
kv102.back.live.telhc.local:5984
kv102.back.live.telhc.local:5985
kv102.back.live.telhc.local:5986
kv102.back.live.telhc.local:5987
kv103.back.live.telhc.local:5984
kv103.back.live.telhc.local:5985
kv103.back.live.telhc.local:5986
kv103.back.live.telhc.local:5987
kv104.back.live.telhc.local:5984
kv104.back.live.telhc.local:5985
kv104.back.live.telhc.local:5986
kv104.back.live.telhc.local:5987
kv105.back.stage.telhc.local:5984
kv105.back.stage.telhc.local:5985
kv105.back.stage.telhc.local:5986
kv105.back.stage.telhc.local:5987
kv106.back.stage.telhc.local:5984
kv106.back.stage.telhc.local:5985
kv106.back.stage.telhc.local:5986
kv106.back.stage.telhc.local:5987
kv001.back.live.cwwtf.local:5984
kv001.back.live.cwwtf.local:5985
kv001.back.live.cwwtf.local:5986
kv001.back.live.cwwtf.local:5987
kv002.back.live.cwwtf.local:5984
kv002.back.live.cwwtf.local:5985
kv002.back.live.cwwtf.local:5986
kv002.back.live.cwwtf.local:5987
kv003.back.live.cwwtf.local:5984
kv003.back.live.cwwtf.local:5985
kv003.back.live.cwwtf.local:5986
kv003.back.live.cwwtf.local:5987
kv004.back.live.cwwtf.local:5984
kv004.back.live.cwwtf.local:5985
kv004.back.live.cwwtf.local:5986
kv004.back.live.cwwtf.local:5987
kv005.back.stage.cwwtf.local:5984
kv005.back.stage.cwwtf.local:5985
kv005.back.stage.cwwtf.local:5986
kv005.back.stage.cwwtf.local:5987
kv006.back.stage.cwwtf.local:5984
kv006.back.stage.cwwtf.local:5985
kv006.back.stage.cwwtf.local:5986
kv006.back.stage.cwwtf.local:5987
db015.back.test.cwwtf.local:5984
db015.back.test.cwwtf.local:5985
db015.back.test.cwwtf.local:5986
db015.back.test.cwwtf.local:5987
db014.back.int.cwwtf.local:5984
db014.back.int.cwwtf.local:5985
db014.back.int.cwwtf.local:5986

CouchDB users will be familiar with the white background - “Futon” a relaxing admin UI (which does NOT have any “special” hooks - it just uses the same API calls). The panel on the left is an addition of ours - showing the shards across different DCs for different environments (live, stage, test, int). Every few seconds, some funky AJAX goes and checks each - giving it a set of colours if not.



bbc_cv
csrfprotection
filmnetwork
gamesgrid
historyworld
homepage
introducing
kv



Carrier 5:22 PM

bamzook Namespaces

- bbc_cv
- csrfprotection
- filmnetwork
- gamesgrid
- historyworld
- homepage
- introducing
- kv

Carrier 5:26 PM

Namespaces replication

Volumetrics

183,864 docs

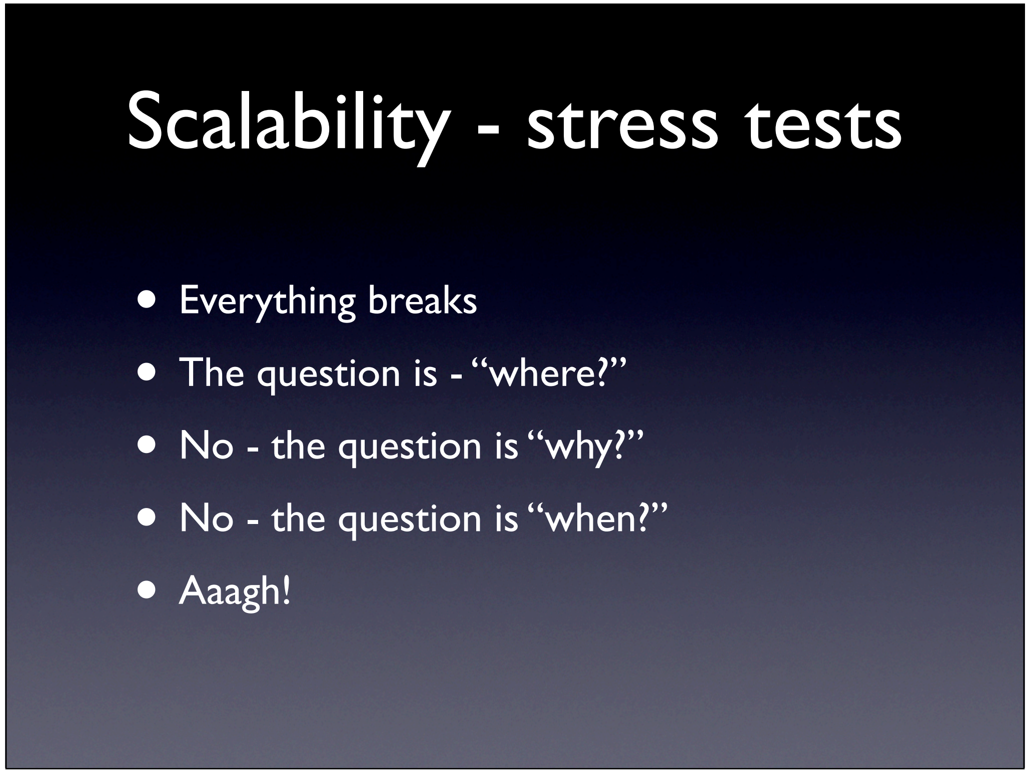
3.3 GB

Authorisations - read/write

@

Enda.Farrell@bbc.co.uk

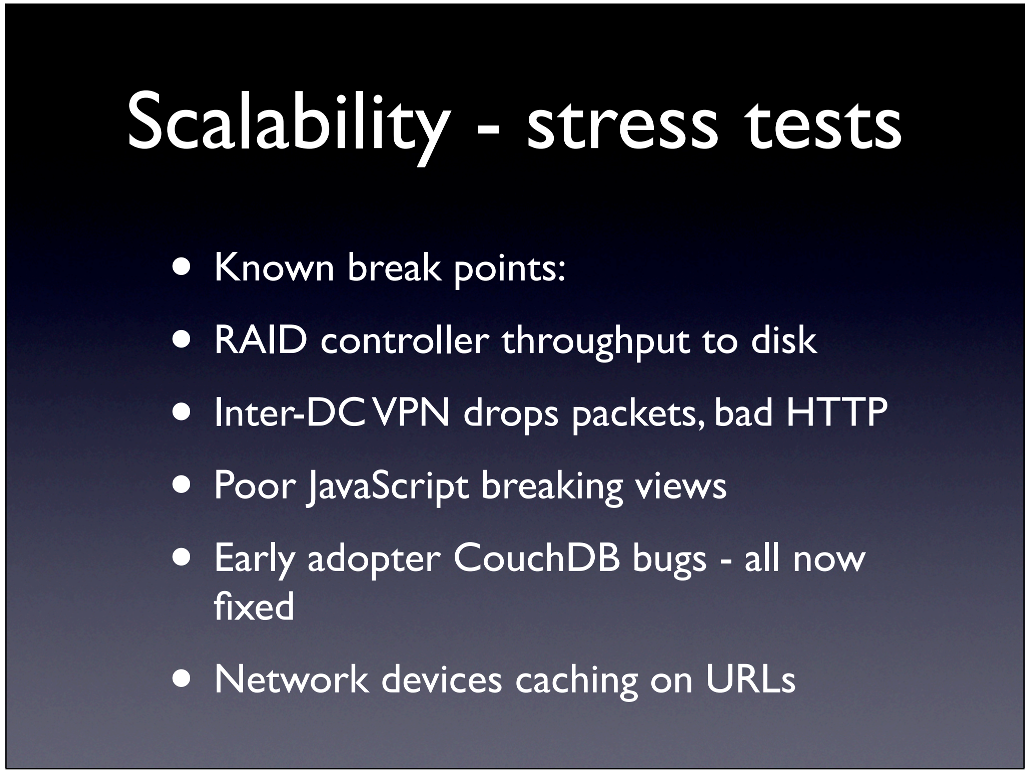
kv.replication.lag.monitor



Scalability - stress tests

- Everything breaks
- The question is - “where?”
- No - the question is “why?”
- No - the question is “when?”
- Aaagh!

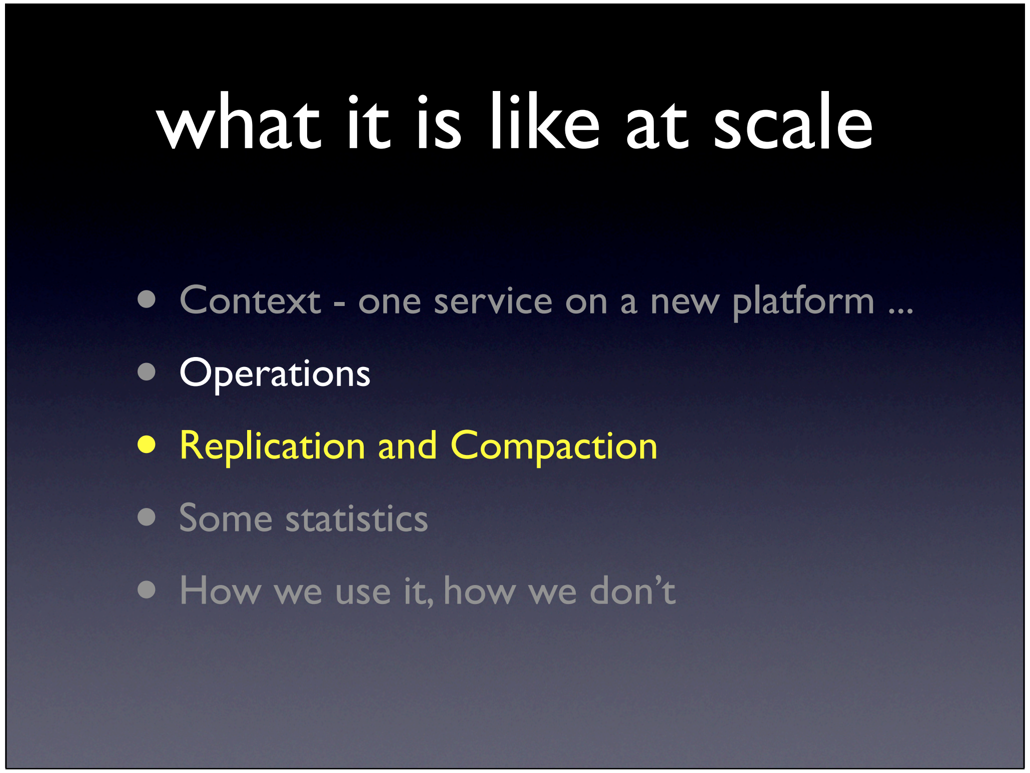
CPU on firewalls, network interrupts on NICs, high churn data evicts memcache and > 10% f/e calls go back to service, bandwidth of traffic managers - all platforms break. Code sometimes breaks too ;-)



Scalability - stress tests

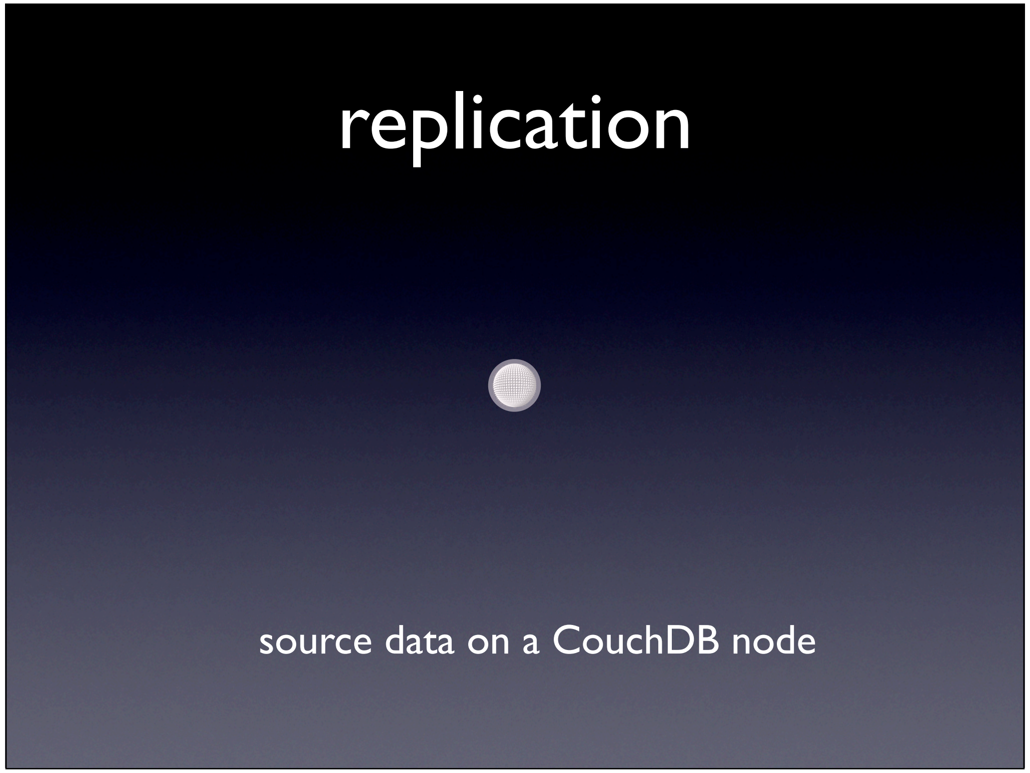
- Known break points:
- RAID controller throughput to disk
- Inter-DC VPN drops packets, bad HTTP
- Poor JavaScript breaking views
- Early adopter CouchDB bugs - all now fixed
- Network devices caching on URLs

1 - Our RAID controllers are a bottle neck - if we try to push MORE than they can handle, the OS on the box starts to back up and that causes problems. Not a CouchDB issue.
2 - Can cause sessions to hang as ACKs are not reliably delivered. If the session is a replication, it makes it look like its hung. Can't really blame CouchDB for that!
3 - Traffic manager CPU - (platform wide, but as one of the most shared network resources, seen on the KV service) - hit that and requests back up
4 - Poor Javascript in views - can completely kill the use of that database on that node - slow response times leading to repeated requests when timeouts occur, leading to a snowball of higher and higher load
5 - Compaction, replication 404 === 6 - Too clever for its own good - poor corporate networks



what it is like at scale

- Context - one service on a new platform ...
- Operations
- Replication and Compaction
- Some statistics
- How we use it, how we don’t

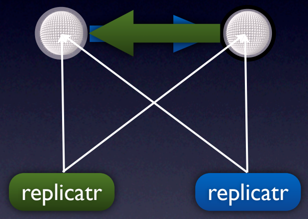


replication

source data on a CouchDB node

This is “trigger” replication, to be replaced with 0.10’s “continuous” replication

replication

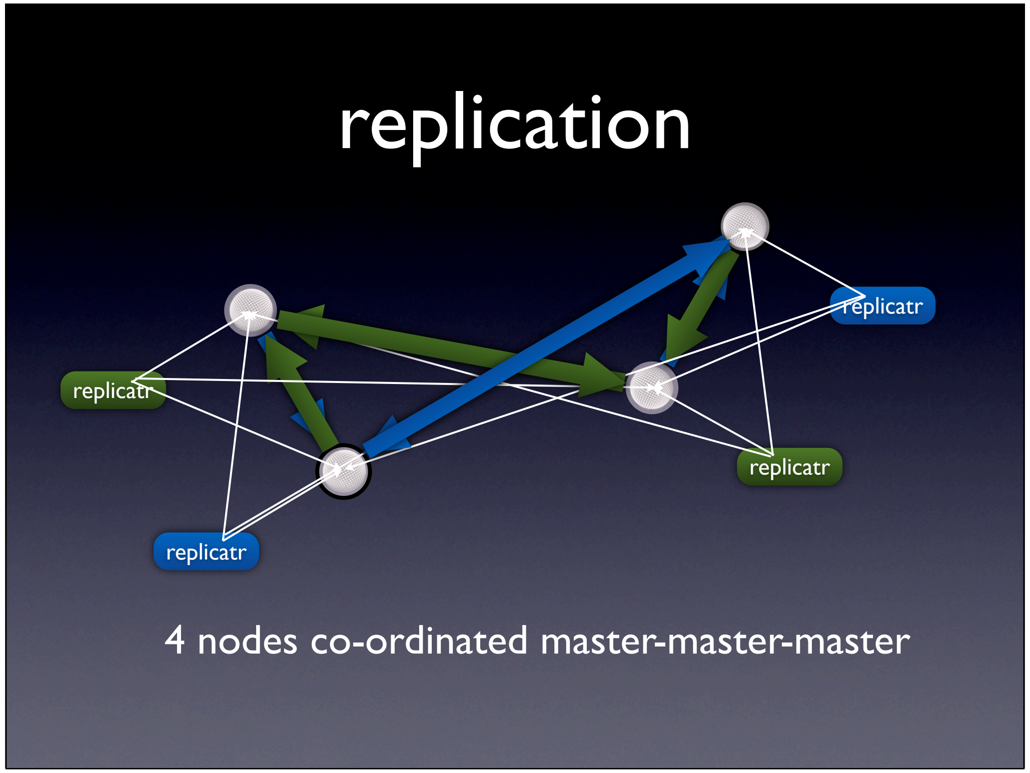


has source changed?

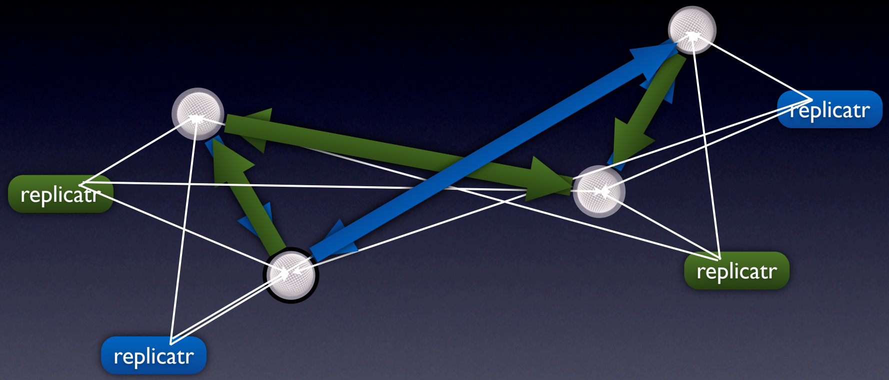
```
POST /db/_replicate
```

CouchDB replicates

replicated pair
nested star

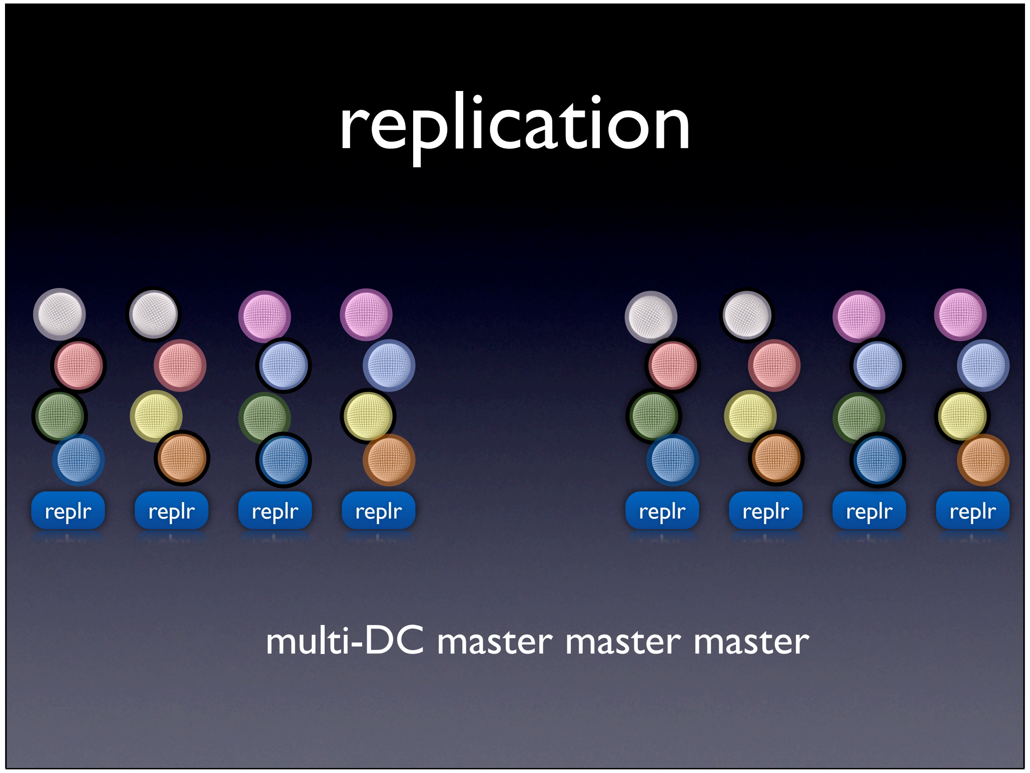


replication



4 nodes co-ordinated master-master-master

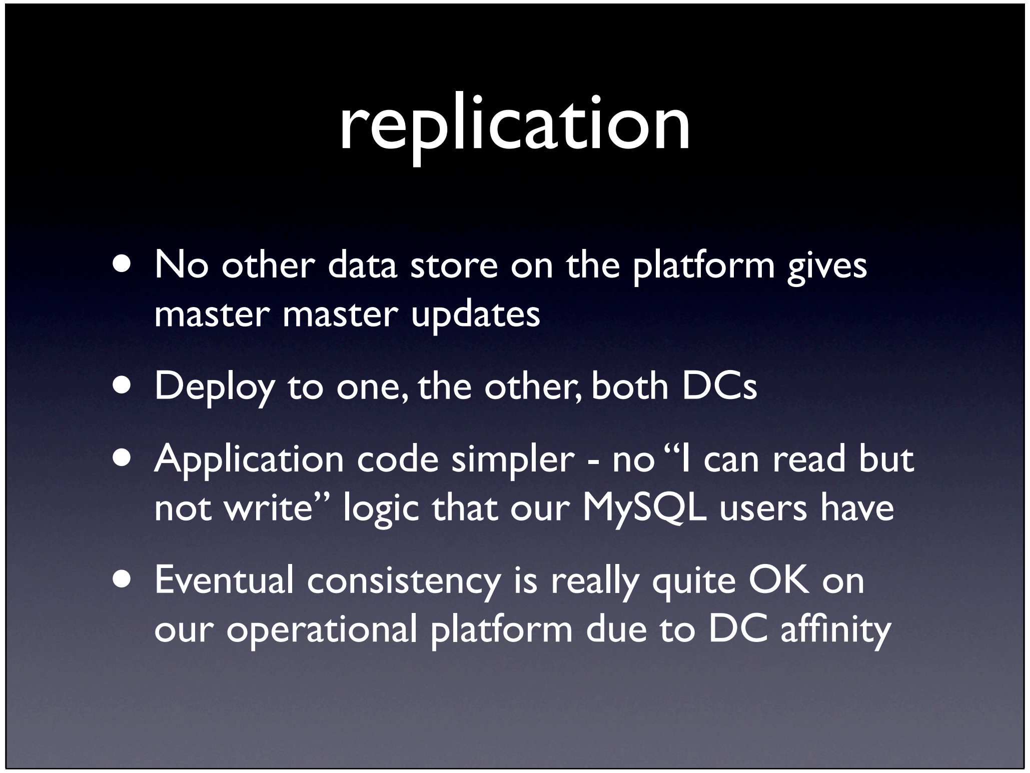
OK - this “looks” scary - but it’s quite normal on our platform, and across the web. It looks good though - helps the business understand some of the hidden complexities



replication

multi-DC master master master

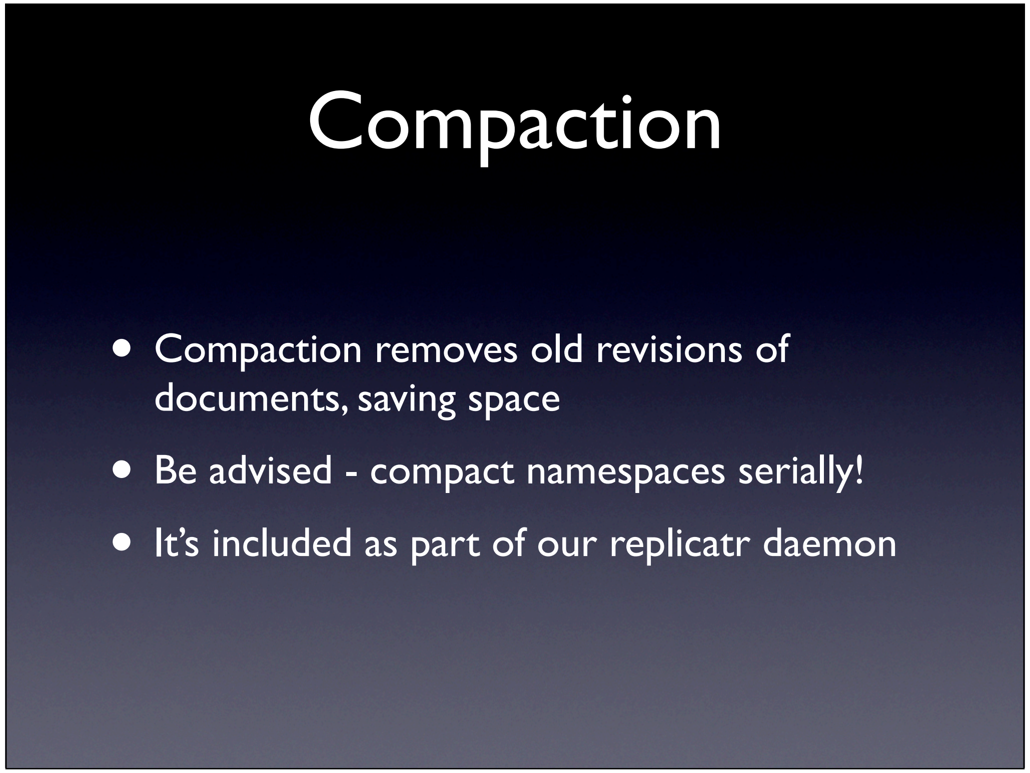
It’s a step up from master-slave to master master. Another one to go to 4 node co-ordinated master-master-master. Another one when you see all such shards together. There’s another step up when you remember that replication is per database - we will have 100s.



replication

- No other data store on the platform gives master master updates
- Deploy to one, the other, both DCs
- Application code simpler - no “I can read but not write” logic that our MySQL users have
- Eventual consistency is really quite OK on our operational platform due to DC affinity

What business advantages come from this? Cool graphs - perhaps! Most importantly, other code using the KV store can be simpler, easier to understand, easier to deploy, and perhaps significantly does NOT need to know whether they are running in a DC which allows writes.

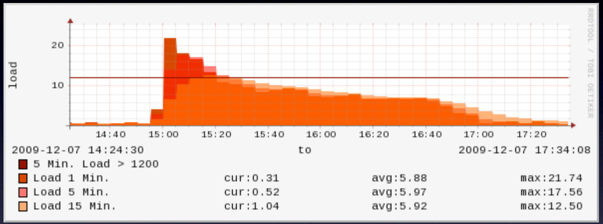


Compaction

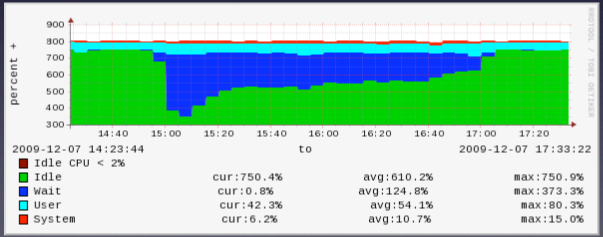
- Compaction removes old revisions of documents, saving space
- Be advised - compact namespaces serially!
- It’s included as part of our replicatr daemon

If you do not update existing docs, there is little benefit in compacting, with the cost of slower access during the compaction process. Some logic is therefore advisable in deciding on when.
On our platform, previous revisions are not important, so we save space.

Compaction

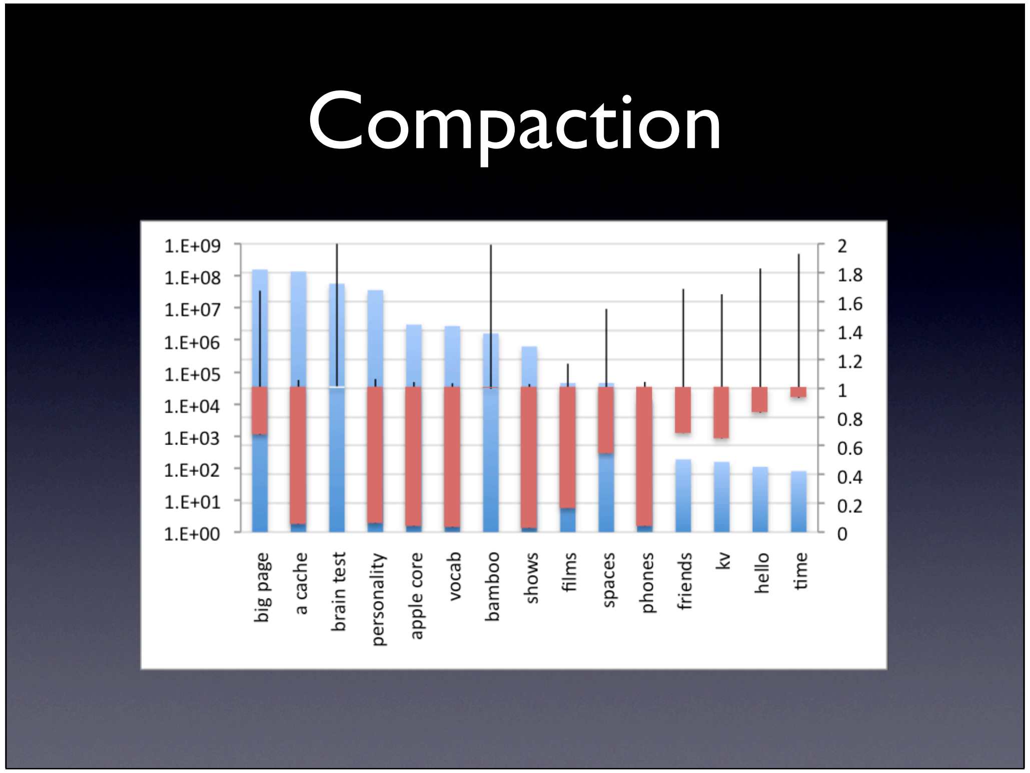


```
2009-12-07 14:24:30                    to                   2009-12-07 17:34:08
■ 5 Min. Load > 1200
■ Load 1 Min.              cur:0.31            avg:5.88            max:21.74
■ Load 5 Min.              cur:0.52            avg:5.97            max:17.56
■ Load 15 Min.             cur:1.04            avg:5.92            max:12.50
```

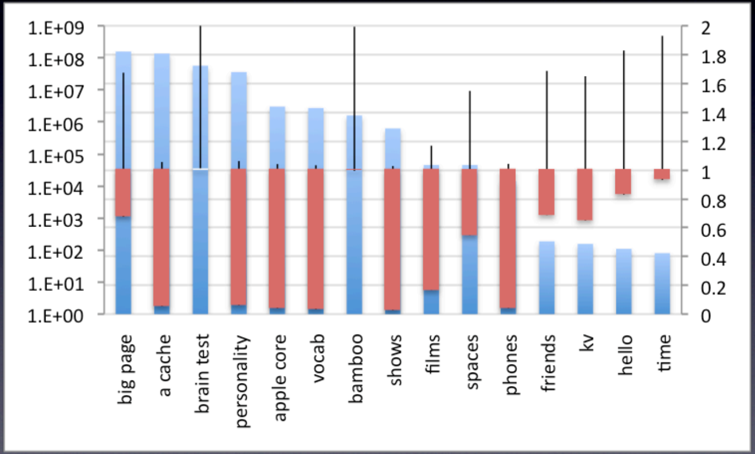


```
2009-12-07 14:23:44                    to                   2009-12-07 17:33:22
■ Idle CPU < 2%
■ Idle                  cur:750.4%          avg:610.2%          max:750.9%
■ Wait                   cur:0.8%           avg:124.8%          max:373.3%
■ User                   cur:42.3%           avg:54.1%           max:80.3%
■ System                  cur:6.2%           avg:10.7%           max:15.0%
```

In Dec we were starting to hit 60% disk, I was going to be away for almost a month and we hadn’t compacted for a while. Instead of doing it serially I compacted almost everything together at once. It was more “ouch” than we had expected!



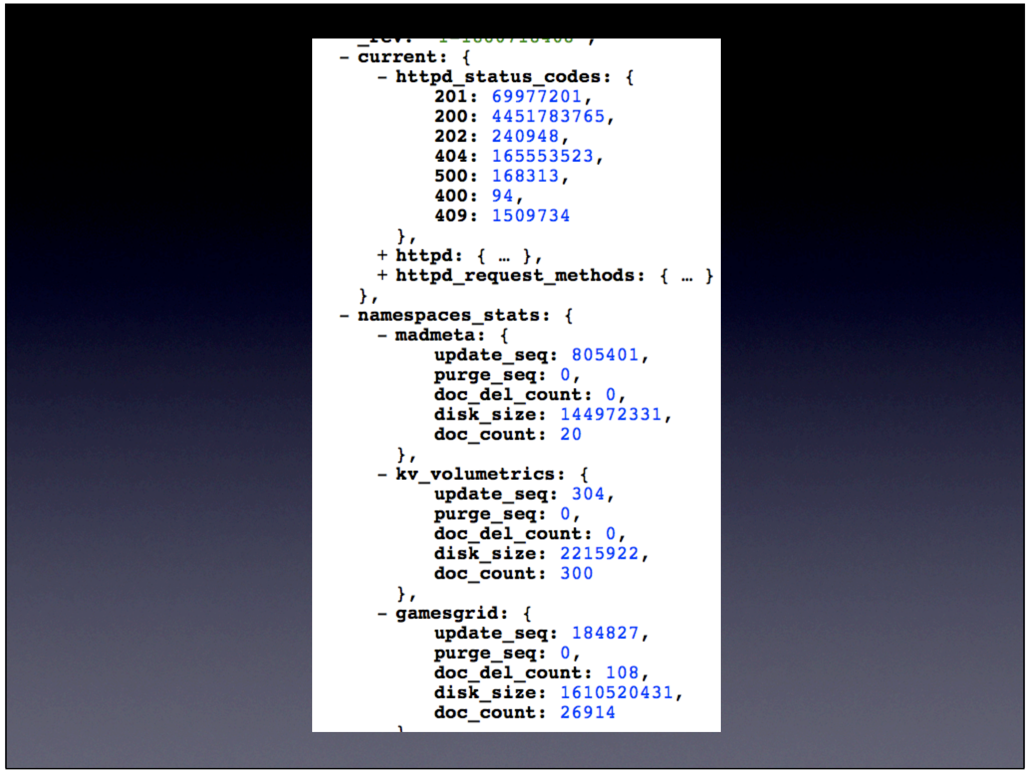
Compaction



Log 10 scale on the left - the size of the databases. Red is compaction RATIO saved. “a cache” compacted fantastically, bamboo not at all and ran at a high possible cost.

what it is like at scale

- Context - one service on a new platform ...
- Operations
- Replication and Compaction
- Some statistics
- How we use it, how we don't



```
- current: {
    - httpd_status_codes: {
          201: 69977201,
          200: 4451783765,
          202: 240948,
          404: 165553523,
          500: 168313,
          400: 94,
          409: 1509734
      },
    + httpd: { … },
    + httpd_request_methods: { … }
  },
- namespaces_stats: {
    - madmeta: {
          update_seq: 805401,
          purge_seq: 0,
          doc_del_count: 0,
          disk_size: 144972331,
          doc_count: 20
      },
    - kv_volumetrics: {
          update_seq: 304,
          purge_seq: 0,
          doc_del_count: 0,
          disk_size: 2215922,
          doc_count: 300
      },
    - gamesgrid: {
          update_seq: 184827,
          purge_seq: 0,
          doc_del_count: 108,
          disk_size: 1610520431,
          doc_count: 26914
```

hourly, we gather up lots of stats and as we eat our own food, being fans of our own service, we keep them for posterity in CouchDB ;-)
So - they were some stats.

1,030

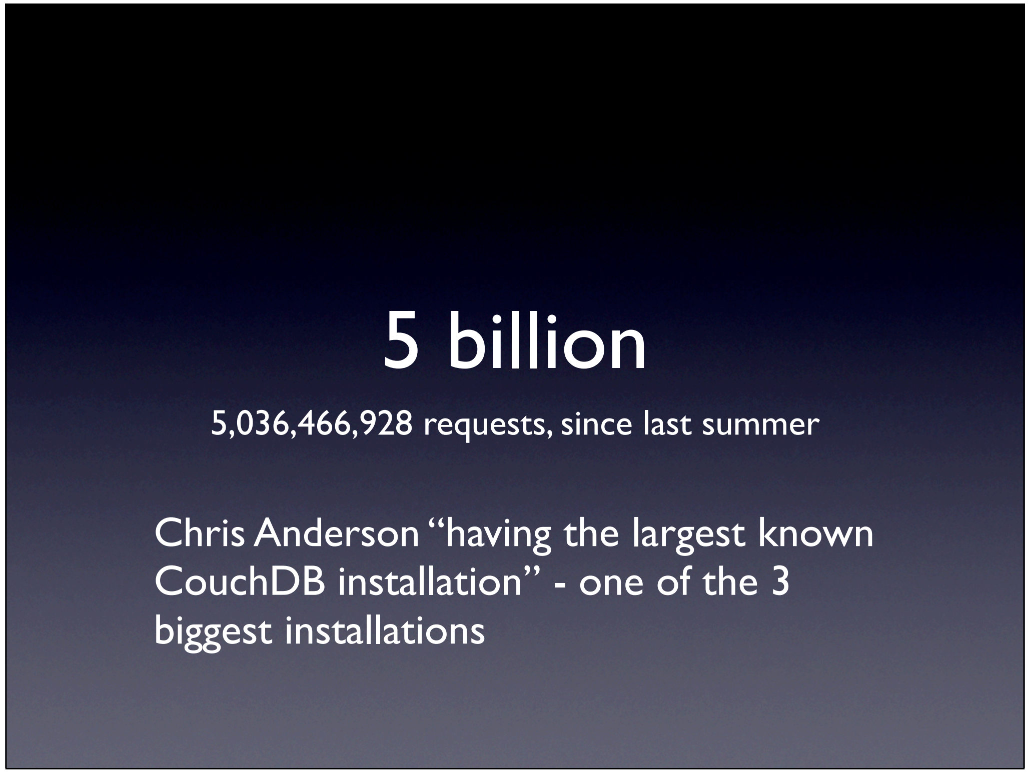
GB

155 million

requests on an average day

20 million

documents



5 billion

5,036,466,928 requests, since last summer

Chris Anderson “having the largest known CouchDB installation” - one of the 3 biggest installations

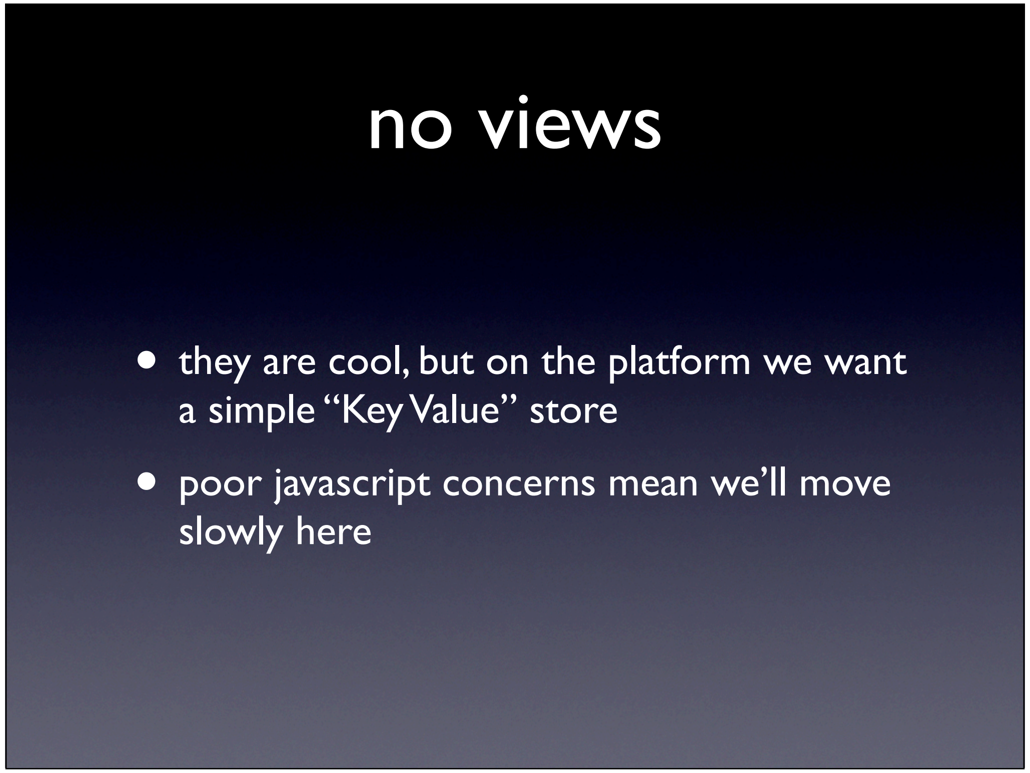
96% are GET, 3% are POST (for replication, compaction and new database creation), 1% are the PUTs which create and update document, we discourage DELETEs due to the highly parallel nature of our platform. One of the 3 biggest known CouchDB installations in the world.

what it is like at scale

- Context - one service on a new platform ...
- Operations
- Replication and Compaction
- Some statistics
- How we use it, how we don’t

How we don’t use it

- Views
- Attachments

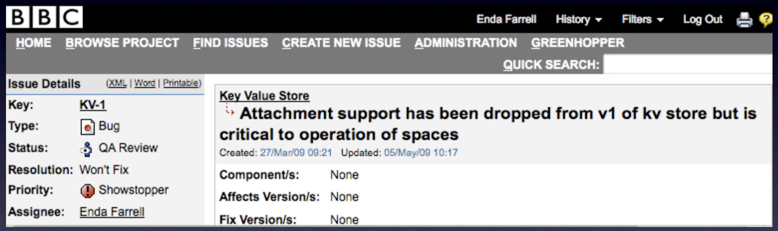


no views

- they are cool, but on the platform we want a simple "Key Value" store
- poor javascript concerns mean we'll move slowly here

Simple - we want to use things in a way that is simpler than CouchDB CAN be used.
My engineering team does not "use" the service much, other developers at the BBC do. Given that CouchDB is schema-less, the structure of documents can change. I can't trust that every developer will take each of their own edge-cases into account here.

no attachments



Enda Farrell History Filters Log Out

HOME BROWSE PROJECT FIND ISSUES CREATE NEW ISSUE ADMINISTRATION GREENHOPPER

QUICK SEARCH:

Issue Details (XML | Word | Printable)

Key:	KV-1
Type:	Bug
Status:	QA Review
Resolution:	Won't Fix
Priority:	Showstopper
Assignee:	Enda Farrell

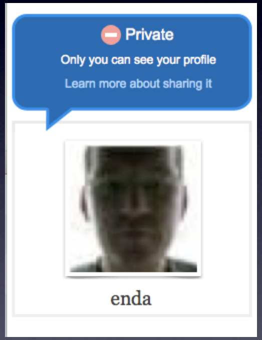
Key Value Store

Attachment support has been dropped from v1 of kv store but is critical to operation of spaces

Created: 27/Mar/09 09:21 Updated: 05/May/09 10:17

Component/s:	None
Affects Version/s:	None
Fix Version/s:	None

no attachments



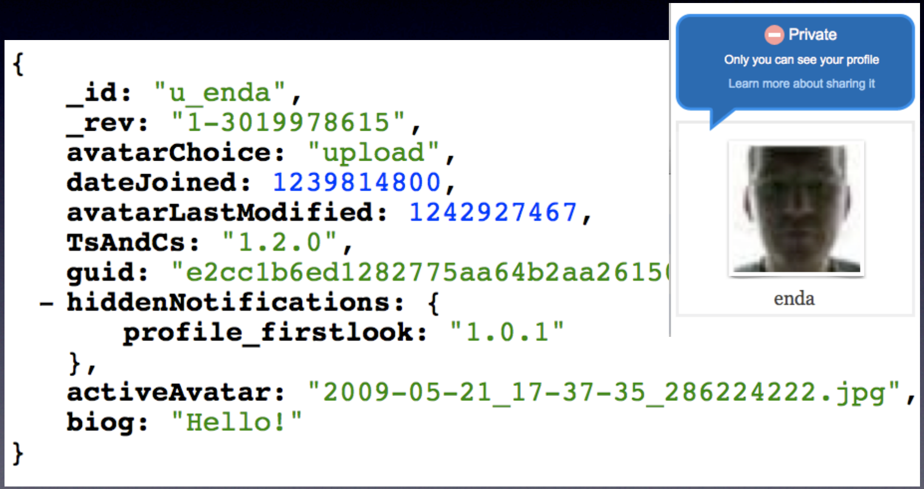
Private

Only you can see your profile

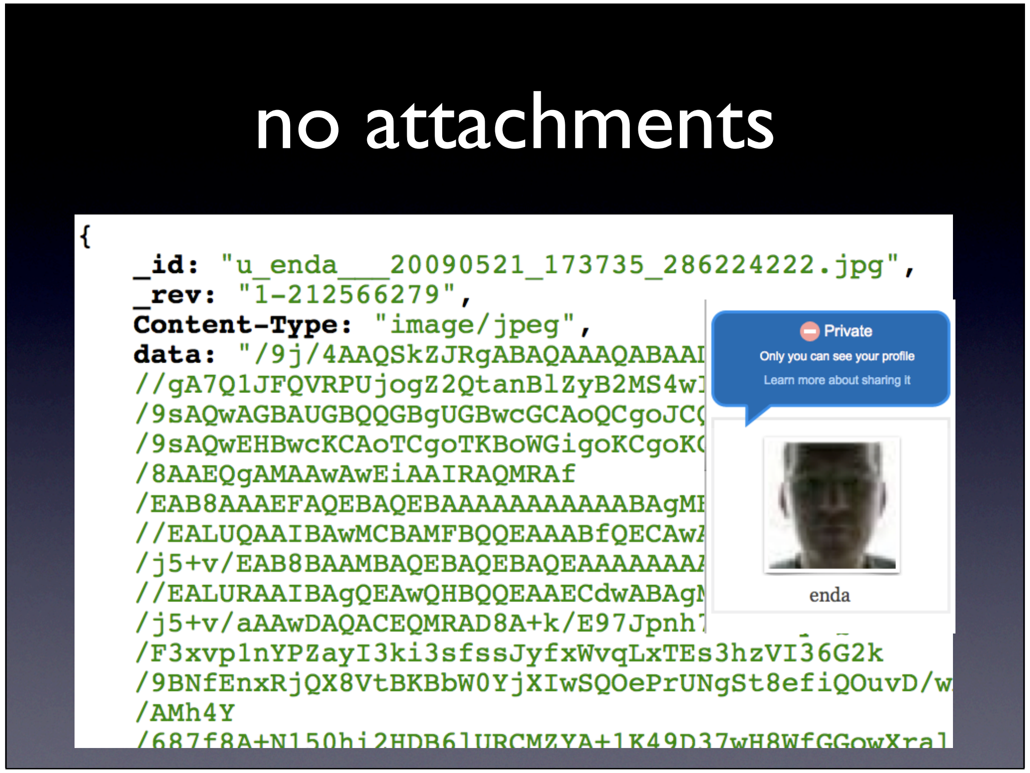
Learn more about sharing it

enda

no attachments



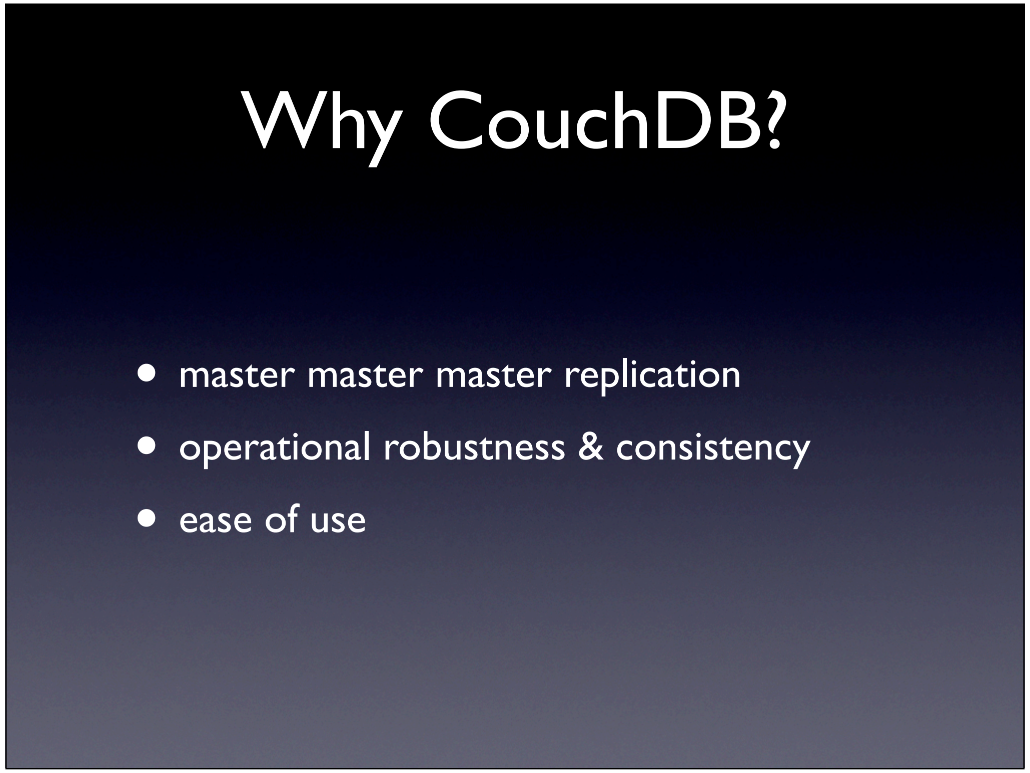
```
{
    _id: "u_enda",
    _rev: "1-3019978615",
    avatarChoice: "upload",
    dateJoined: 1239814800,
    avatarLastModified: 1242927467,
    TsAndCs: "1.2.0",
    guid: "e2cc1b6ed1282775aa64b2aa2615
  - hiddenNotifications: {
        profile_firstlook: "1.0.1"
    },
    activeAvatar: "2009-05-21_17-37-35_286224222.jpg",
    biog: "Hello!"
}
```



no attachments

```
{
    _id: "u_enda___20090521_173735_286224222.jpg",
    _rev: "1-212566279",
    Content-Type: "image/jpeg",
    data: "/9j/4AAQSkZJRgABAQAAAQABAAI
    //gA7Q1JFQVRPUjogZ2QtanBlZyB2MS4wI
    /9sAQwAGBAUGBQQGBgUGBwcGCAoQCgoJCQ
    /9sAQwEHBwcKCAoTCgoTKBoWGigoKCgoKC
    /8AAEQgAMAAwAwEiAAIRAQMRAf
    /EAB8AAAEFAQEBAQEBAAAAAAAAAAABAgMI
    //EALUQAAIBAwMCBAMFBQQEAAABfQECAwA
    /j5+v/EAB8BAAMBAQEBAQEBAQEAAAAAAAA
    //EALURAAIBAgQEAwQHBQQEAAECdwABAgM
    /j5+v/aAAwDAQACEQMRAD8A+k/E97Jpnh7
    /F3xvp1nYPZayI3ki3sfssJyfxWvqLxTEs3hzVI36G2k
    /9BNfEnxRjQX8VtBKBbW0YjXIwSQOePrUNgSt8efiQOuvD/w
    /AMh4Y
    /687f8A+N150hj2HDB61URCMZYA+1K49D37wH8WfGGowXral
```

Actually, in our environment, attachments are usually images or media assets and they really ought not be served from inside a database - so this too is a platform architecture restriction.



Why CouchDB?

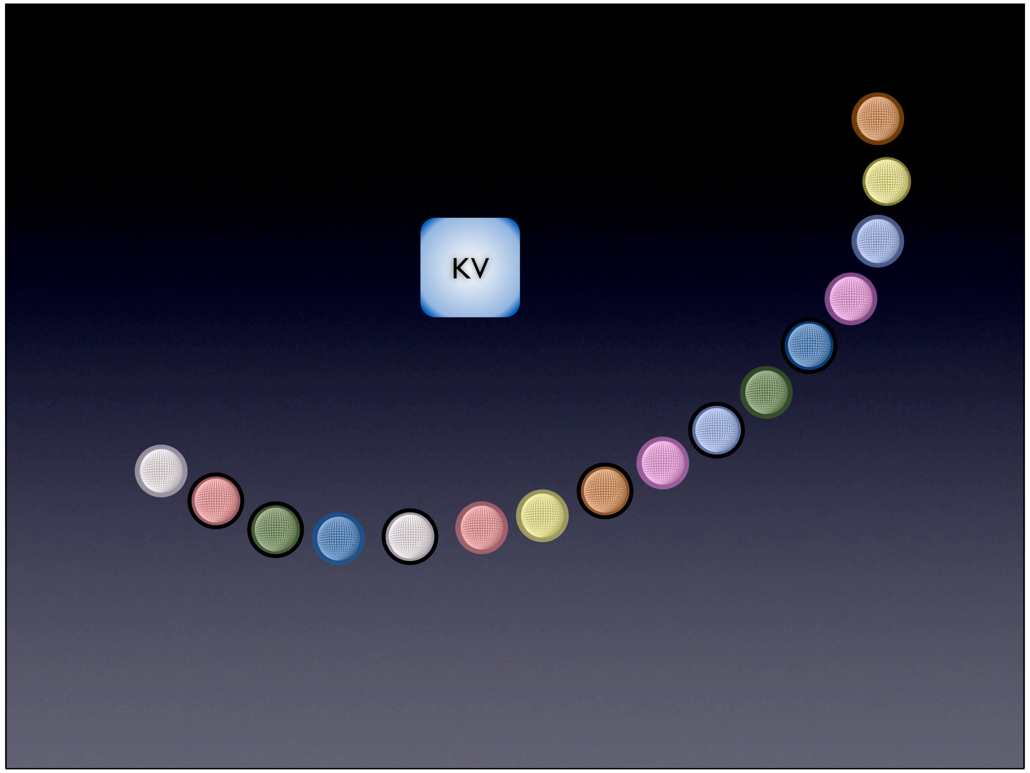
- master master master replication
- operational robustness & consistency
- ease of use

Even though we knew the code wasn’t even “beta”, and we knew that some high-profile sites would depend on it, it did exactly what it said on the tin, and we could wrap it to stop over-zealous developers (who don’t spend enough time thinking about operational impact) using features which may cause headaches

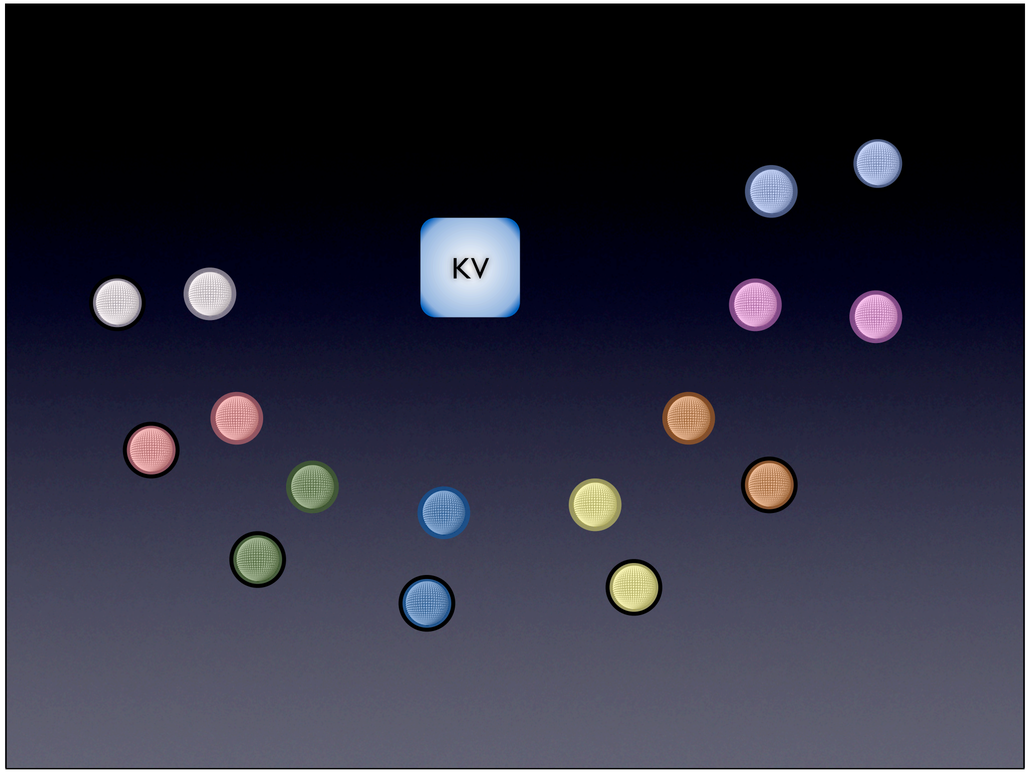
OuchDB*

just one horror story

Rachel's response to hearing about one of our mishaps



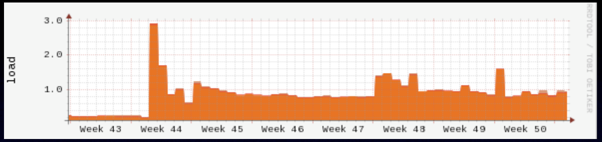
At first we did not have hardware redundancy, so had 16 shards

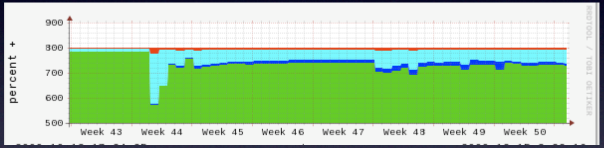


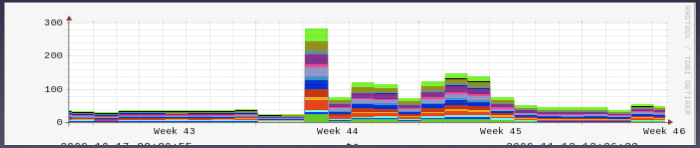
Our intention was to reduce to 8 shards, freeing up the other 8 to be hot-failovers in the event of hardware problems. We had a config error, resulting in us not being able to find some data in one DC and not being able to find other data until replication had finished.

2⁄8 unavailable 50%

1⁄2 unavailable until replication done

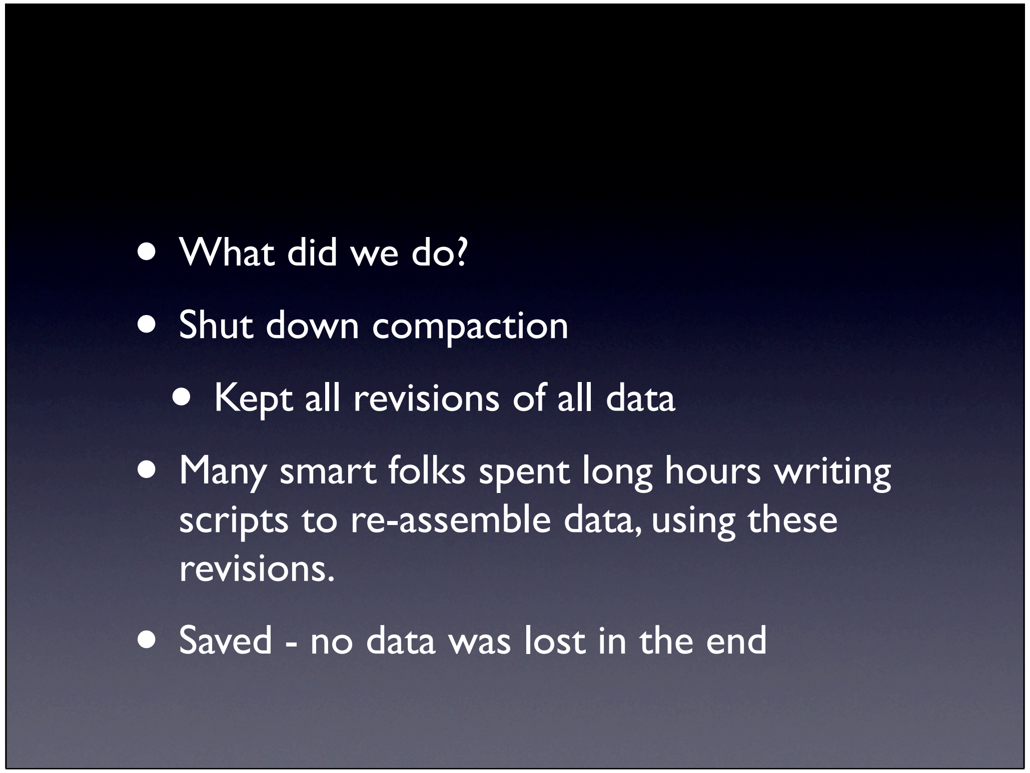






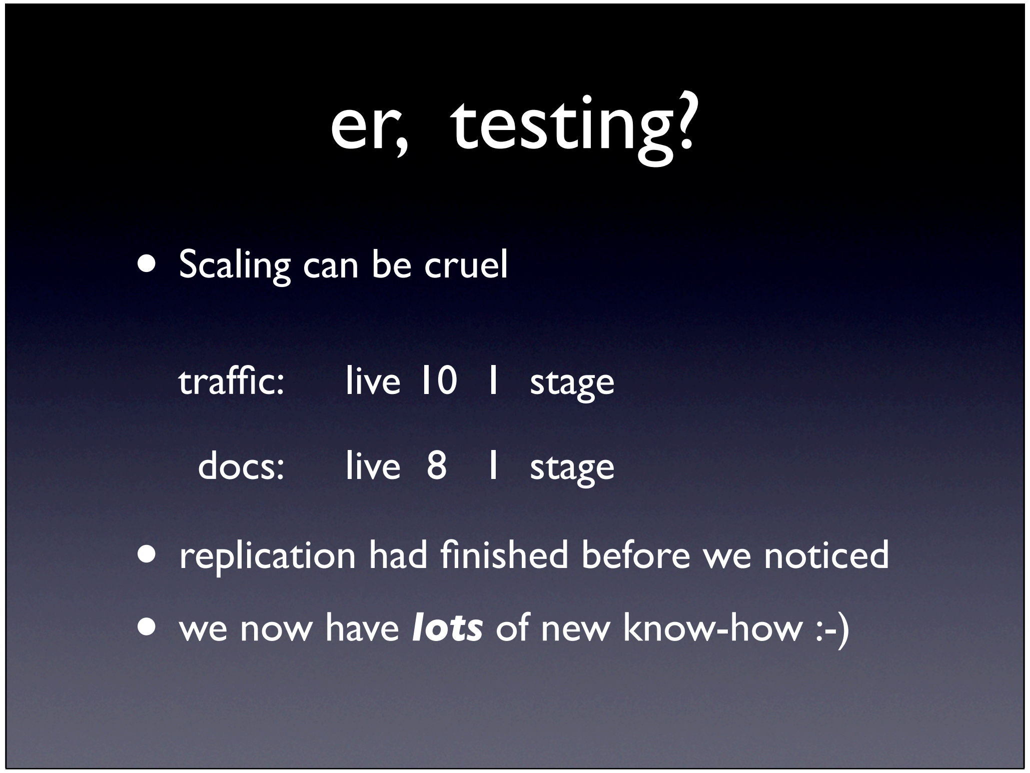
The load - partially driven by 404 snowballs - was much higher than expected.

- 3.5% of our requests result in 404s
  - this we consider normal
- Some applications created new docs
- The load was not expected
- Replication ought to have taken 30 min - but with bad config ~ 7+ hours



- What did we do?
- Shut down compaction
  - Kept all revisions of all data
- Many smart folks spent long hours writing scripts to re-assemble data, using these revisions.
- Saved - no data was lost in the end

CouchDB to the rescue! If you don’t compact, CouchDB’s MVCC can come to the rescue



er, testing?

- Scaling can be cruel

traffic:	live	10	1	stage
docs:	live	8	1	stage

- replication had finished before we noticed
- we now have ***lots*** of new know-how :-)

This is “glib” in comparison to what we went through, but in summary, it’s what mattered.

CouchDB: we like it.

thank you

twitter: @endafarrell
blog: endafarrell.net