

# Where do I put this Data?

Ezra Zygmuntowicz  
@ezmobius

#lessql



# SQL Databases Are Not A Panacea

**SQL Databases Are  
Not A Panacea**

**NO-SQL Databases  
Are Not A Panacea**

SQL Databases Are

Not A Panacea

**Duuh!**

NO-SQL Database

Are Not A Panacea

Who really needs a  
#lessql database?

Who really needs a  
#nosql database?

Probably not you!

Who really needs a  
#nosql database?

Probably not you!

Unless you do your homework first...

# Limitations of SQL

Don't think #NOSQL

Think #LESSQL

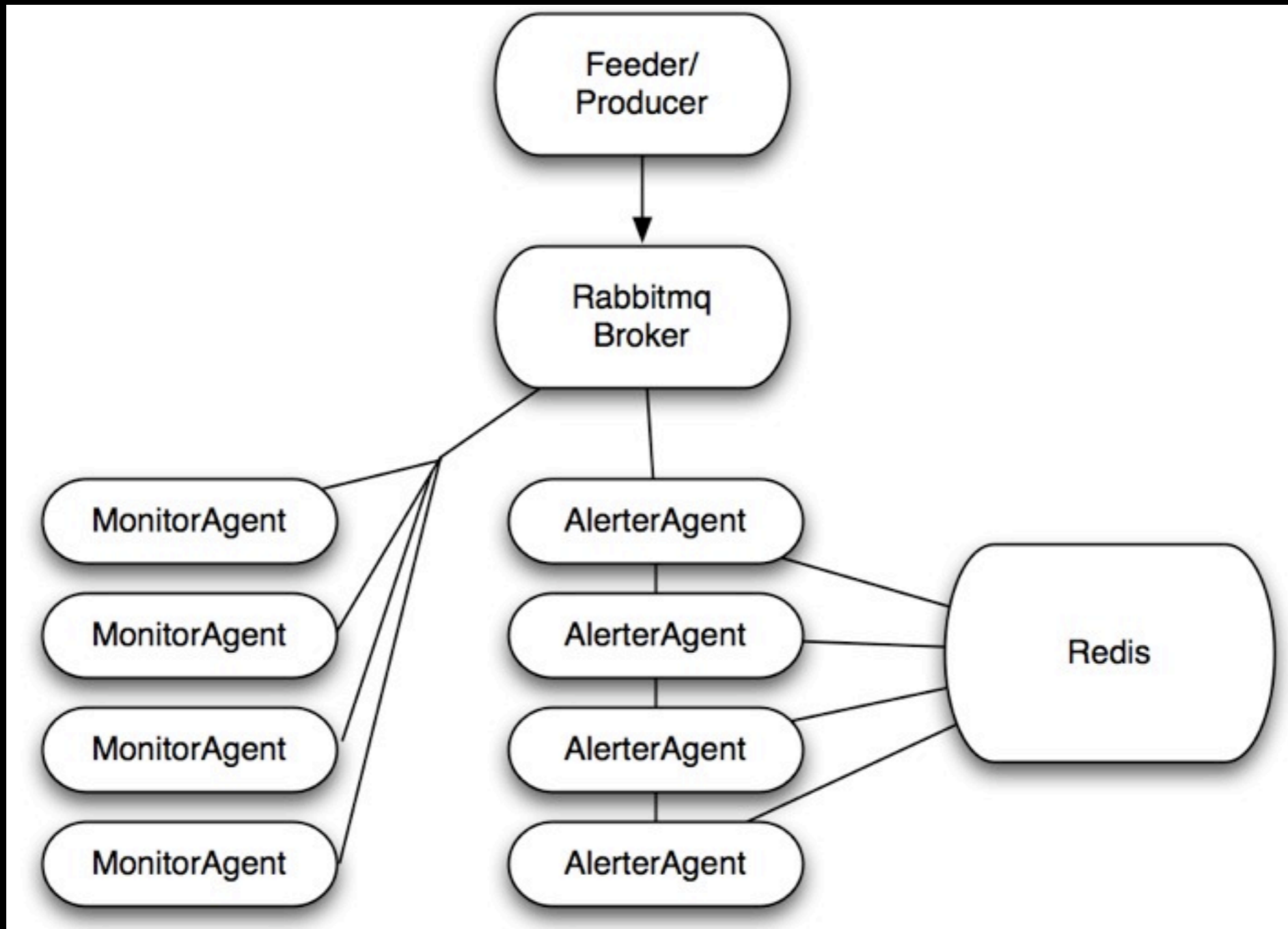
Hybrid systems are  
where it's at

Find a small part of your application that has pain because of the sql database and port just that part to one of these systems

Find a small part of your  
application that has pain  
because of the sql database  
and port just that part to  
one of these systems

rinse ... repeat...

# Example



# The Players

- Redis
- Tokyo\*
- MongoDB
- Riak
- Cassandra
- Dynamite



- Fast, in memory key/value store
- Async disk persistence
- STRING, LIST and SET data types
- Perfect Data Structure/State/Cache Server

<http://code.google.com/p/redis/>

<http://github.com/ezmobius/redis-rb>



## Pros:

- Very Fast(110k/ops/sec)
- High Level Data Types
- Sequential IO Only
- Very Clean C Code

## Cons:

- Data Set must fit in Memory
- Possible to lose some data between syncs(configurable)

Scales horizontally via consistent hashing in the client



Use Redis when you want:

As fast as you can get

Data structure sharing between ruby processes

A better, persistent memcached

Hit counters, rate limiters, circular log buffers and sessions

# Tokyo Cabinet 8192PiB

- Large data workhorse
- Native, memcached and http interfaces
- Fast with full disk persistence
- Key/value with Lua extensions available

<http://1978th.net/tokyocabinet/>

<http://1978th.net/tokyotyrant/>

<http://github.com/jmettraux/rufus-tokyo>

<http://copiousfreetime.rubyforge.org/tyrantmanager/>

**Tokyo  
Cabinet**  8192PiB

### Pros:

- Fast and Stable
- Ability to store Large amounts of data
- Embedded Lua in the tyrant server
- Pluggable storage engines

### Cons:

- No data types for values (table db type excluded)
- Some issues with data sets larger then 70Gigs

Scales horizontally via consistent hashing in the client

Tyrant also supports master/master and master /slave replication



Use Tokyo when you want:

As fast as you can get with synchronous writes

Store large amounts of persistent data

Use the smallest amount of disk space for your data set

Tunable RAM usage



{name: "mongo", type: "db"}

- JSON document database
- the 'mysql' of key/value stores
- Fast but flexible query engine
- Support for sharding baked in(sorta)
- Replication

<http://www.mongodb.org>

<http://github.com/mongodb/mongo-ruby-driver>

<http://github.com/jnunemaker/mongomapper>



# mongoDB

{name: "mongo", type: "db"}

**Pros:**

Schemaless

Advanced query features

Relatively Fast

GridFS

Easy transition from SQL databases

Active development

**Cons:**

AutoSharding not ready yet

No Transactions

Scales horizontally via auto sharding  
Supports master/slave replication for failover



`{name: "mongo", type: "db"}`

Use MongoDB when you want:

Very Fast with synchronous writes

Store large amounts of schemaless data

Great for logging, stats collection

Very powerful query API and indexing capabilities

# riak

- Document oriented database
- HTTP/JSON query interface
- Erlang map/reduce query interface
- Decentralized, just add or remove nodes to scale

<http://riak.basho.com/>

# riak

## Pros:

Schemaless

No master node/share  
nothing

Add/remove nodes  
easily to scale up/down

## Cons:

Still young project  
Not much documentation

Scales horizontally via shared nothing, hinted handoff and consistent hashing. Definitely one to watch

# riak

Use Riak when you want:

Ease of operations when adding/removing nodes

# Cassandra

Got logo?

- Eventually consistent, distributed, structured key/value store
- Cross between Big Table and Dynamo
- Column Families provide higher level data models than most key/value stores
- Truly add or remove nodes to scale capacity

<http://incubator.apache.org/cassandra/>

<http://blog.evanweaver.com/articles/2009/07/06/up-and-running-with-cassandra/>

# Cassandra

Got logo?

## Pros:

Always writable

Horizontally scalable

Add nodes easily to scale  
write capacity

Easy to get common  
sorted queries (recent  
blog posts etc)

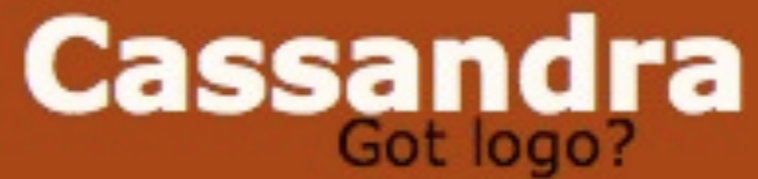
Scales horizontally via automatic replication, even tunable  
across racks/data centers

## Cons:

Restart whole system when  
making schema changes

Not much documentation

Rough edges abound



Use Cassandra when you want:

Truly just add nodes to scale out

Fairly rich data model, sorted supercolumn families

Store truly large amounts of data



- Eventually consistent, distributed, key/value store
- Based on Amazon's Dynamo Papers
- Data Partitioning, versioning and read repair
- Written in Erlang

<http://github.com/cliffmoon/dynomite>



## Pros:

Always writable

Horizontally scalable

Good for storing large binaries

Add nodes to repartition

Gossip protocol

Pluggable storage engines

Scales horizontally via automatic replication, talk to any node in the cluster from clients

## Cons:

New partitions come online before they are *\*ready\**

Migration is very painful

Still beta quality but used in production at powerset



Use Dynamite when you want:

Scale writes by adding nodes

Store large binaries (like image assets)

Nice web admin interface

Thats great but which  
one should I use?

Thats great but which  
one should I use?

Unless you have good reasons  
stick with Redis, Tokyo and  
MongoDB for now






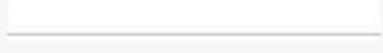







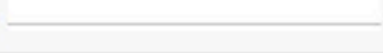
Thats great but which  
one should I use?

Unless you have good reasons  
stick with Redis, Tokyo and  
MongoDB for now

But keep an eye on the others  
for truly add node to scale out  
capacity

# Chef recipes to configure all of these on the Engine Yard Cloud

<http://github.com/ezmobius/ey-lessql>

lessql (Nginx + Passenger)		<a href="#">Deploy</a>	<a href="#">Terminate</a>	<a href="#">Snapshot</a>	<a href="#">Crontab</a>	<a href="#">Deploy Logs</a>	<a href="#">Clone</a>		
Instances	Applications	SSL	Monitored URL	Alerts					
 Utility[redis](i-17bf1f7f)		1.7G RAM, 1ECU, 32bit	<a href="#">ec2-75-101-222-11.c...</a>		CPU Usage: 	<a href="#">Terminate</a>			
server fully configured and ready									
 Utility[tokyo](i-e5bf1f8d)		1.7G RAM, 1ECU, 32bit	<a href="#">ec2-75-101-169-118....</a>		CPU Usage: 	<a href="#">Terminate</a>			
server fully configured and ready									
 Utility[mongodb](i-a1bf1fc9)		1.7G RAM, 1ECU, 32bit	<a href="#">ec2-67-202-56-145.c...</a>		CPU Usage: 	<a href="#">Terminate</a>			
server fully configured and ready									
 Utility[riak](i-bfbf1fd7)		1.7G RAM, 1ECU, 32bit	<a href="#">ec2-72-44-58-90.com...</a>		CPU Usage: 	<a href="#">Terminate</a>			
server fully configured and ready									
 Utility[cassandra](i-91bf1ff9)		1.7G RAM, 1ECU, 32bit	<a href="#">ec2-174-129-162-211...</a>		CPU Usage: 	<a href="#">Terminate</a>			
server fully configured and ready									
 Utility[dynomite](i-7380201b)		1.7G RAM, 1ECU, 32bit	<a href="#">ec2-67-202-44-48.co...</a>		CPU Usage: 	<a href="#">Terminate</a>			
server fully configured and ready									

# Pitfalls of #LESSQL

- No Referential Integrity
- Not as much tooling
- Almost non existent disaster recovery tools
- Not as much production, used in anger experience

Do not buy into the hype!  
Most of you do not need this  
stuff. Relational databases scale  
well for 99% of use cases.

Don't do it because it's  
“Cool”

But if you do your homework, #LESSQL can be very compelling and very useful.

Please make informed decisions so you don't have to hire me to clean up your mess!

Questions?