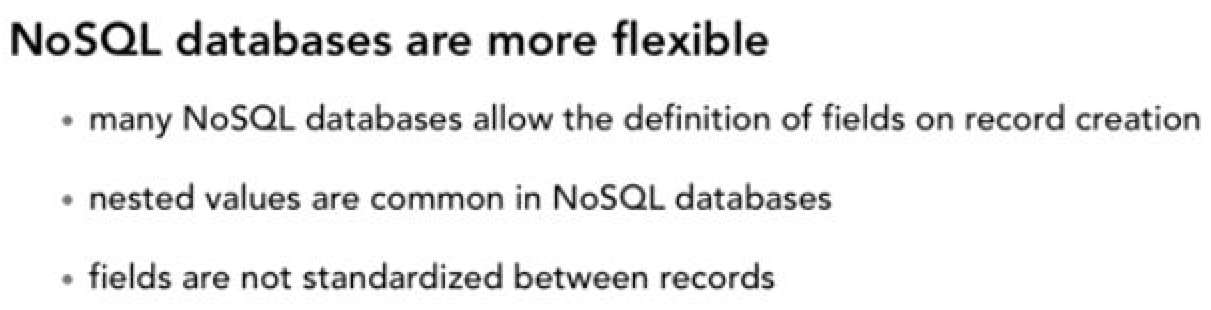
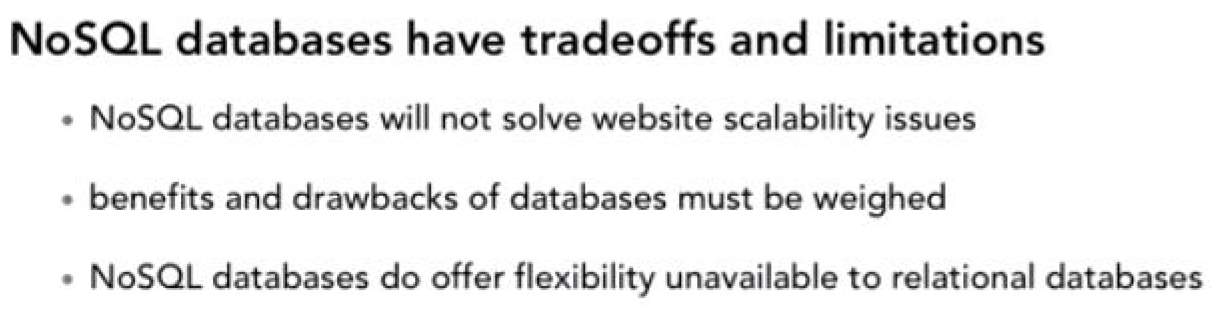
NO SQL DATABASES: RIAL REDIS MONGODB CASSANDRA MONGODB AND COUCHDB



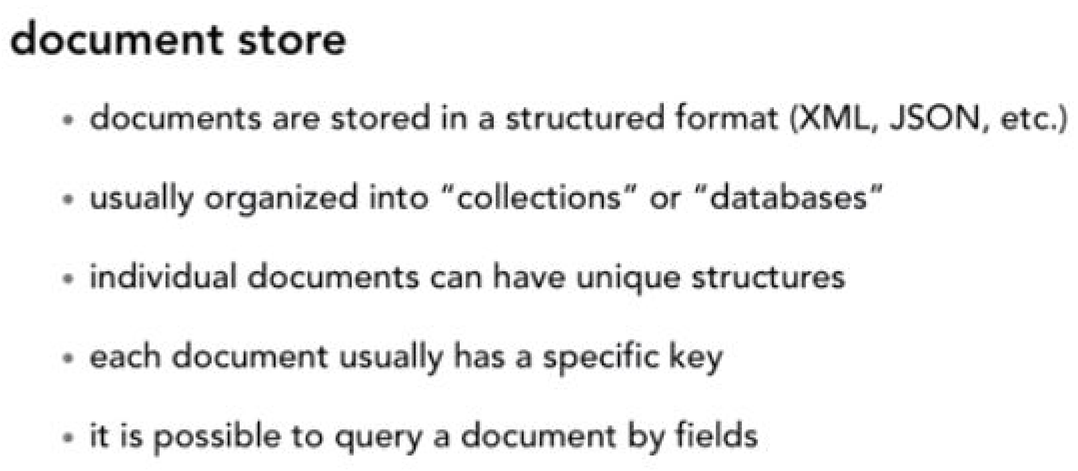
NoSQL databases are more flexible

- many NoSQL databases allow the definition of fields on record creation
- nested values are common in NoSQL databases
- fields are not standardized between records



NoSQL databases have tradeoffs and limitations

- NoSQL databases will not solve website scalability issues
- benefits and drawbacks of databases must be weighed
- NoSQL databases do offer flexibility unavailable to relational databases



document store

- documents are stored in a structured format (XML, JSON, etc.)
- usually organized into “collections” or “databases”
- individual documents can have unique structures
- each document usually has a specific key
- it is possible to query a document by fields

key-value store

- you have a key you can query by, and the value at that key
- drawback: you usually can't query by anything other than the key
- some key-value stores let you define more than one key
- sometimes used alongside relational databases for caching

BigTable/tabular

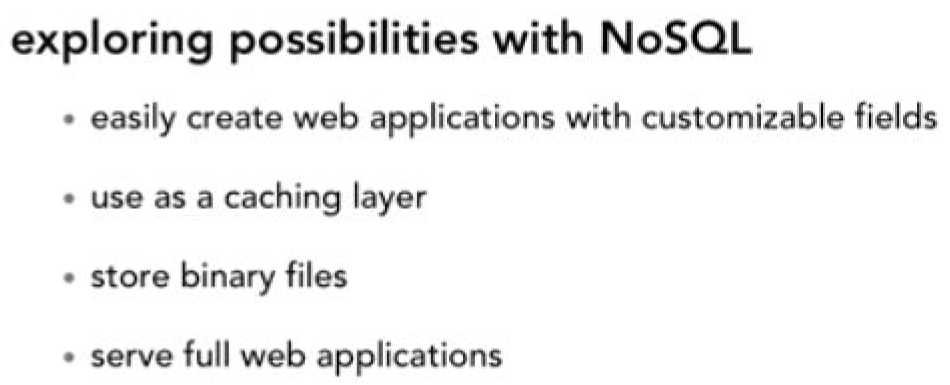
- named after Google's proprietary "BigTable" implementation
- each row can have a different set of columns
- designed for large numbers of columns
- rows are typically versioned

graph

- designed for data best represented as interconnected nodes
- example: a series of road intersections

object database

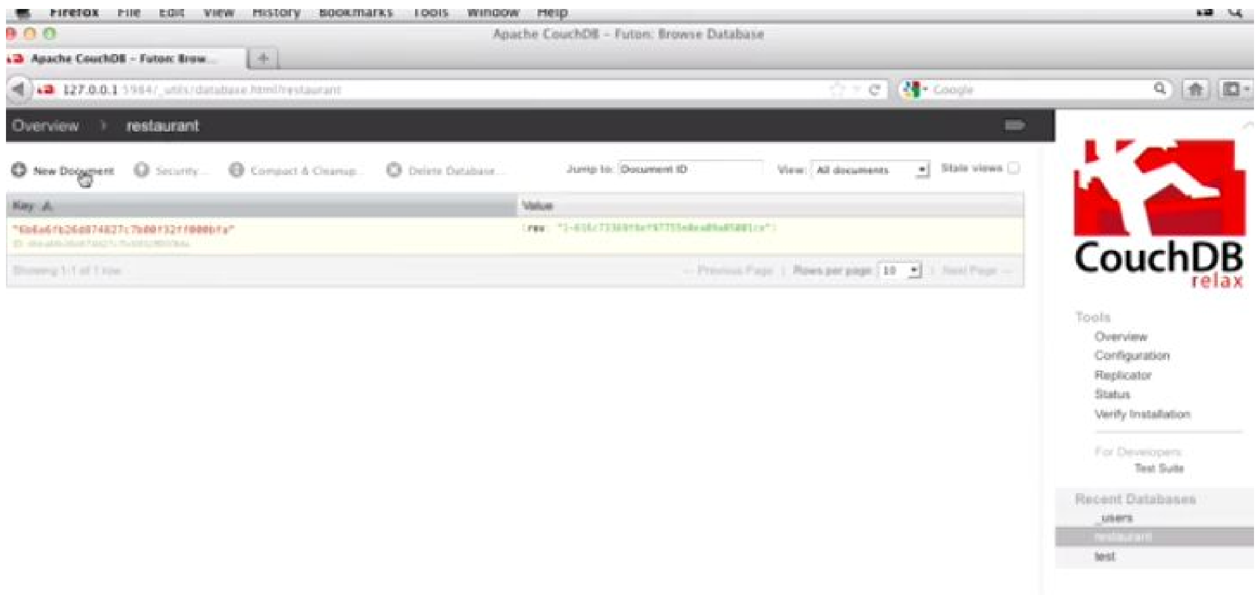
- tightly integrated with the object-oriented programming language used
- act as a persistence layer: store objects directly
- you can link objects together through pointers



exploring possibilities with NoSQL

- easily create web applications with customizable fields
- use as a caching layer
- store binary files
- serve full web applications

For handling metadata as well.
For handling data like script or stylesheet files.



Apache CouchDB – Futon: Browse Database

127.0.0.1:5984/_utils/database.html?restaurant

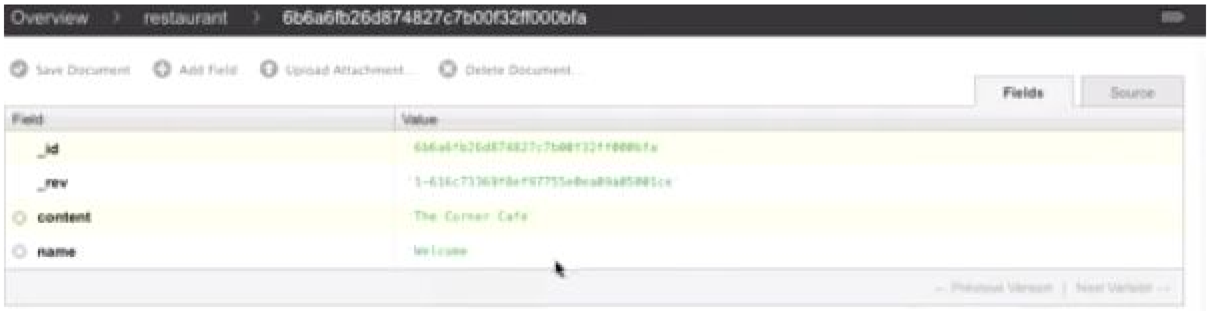
Overview › restaurant

Jump to: Document ID　View: All documents　Stale views

Key	Value
"6b6a6fb26d874827c7b00f32f000bfa"	{rev: "[illegible]"}

Showing 1-1 of 1 row　Rows per page: 10

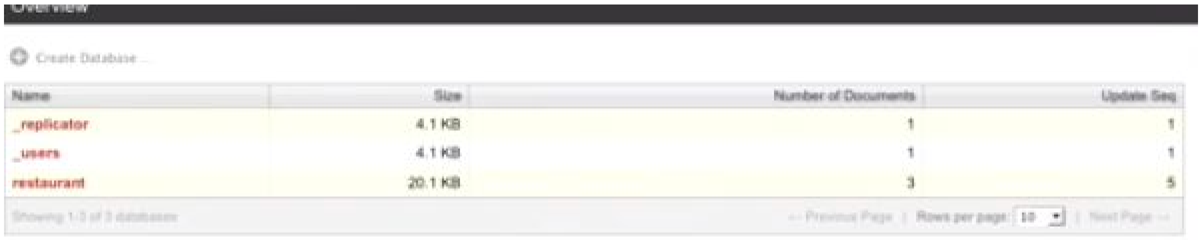
1key value pair generated automatically



Overview › restaurant › 6b6a6fb26d874827c7b00f32ff000bfa

Field	Value
_id	6b6a6fb26d874827c7b00f32ff000bfa
_rev	1-616c73369f8ef67755e0ea09a05001ce
content	The Corner Cafe
name	Welcome

Fieldsets in the document in couchdb (rev increments after each revision of document)



Overview

Name	Size	Number of Documents	Update Seq
_replicator	4.1 KB	1	1
_users	4.1 KB	1	1
restaurant	20.1 KB	3	5

Showing 1-3 of 3 databases

Each database can have many documents access using key value pair and each document has a number of fields whose data is structured using JSON.

The map function first retrieves a keyed set of data. Then, the reduce function takes the values mapped to each key, and transforms them into a single value. Using this combination allows you to retrieve summaries directly from the database, saving time and bandwidth.

Deployment

With photos and names already in the database, the only thing we need now is some way of displaying the data in a browser. This is made possible in CouchDB through the concept of applications. With an application, you can upload HTML and JavaScript that can be served directly to a browser. This is done by uploading these files as attachments on the design menu.

what is database partitioning?

- splitting data across multiple servers
- done by a consistent method
- ranges: A-L, M-Q, R-Z
- lists: textbooks, cookbooks, sci-fi
- hashes: function returning a value to determine membership

why partition?

- storage limitations: the entire dataset may not fit on one server
- performance: splitting the load between partitions can make writes faster
- availability: ensures you can get the data even if one server gets busy

why not partition?

- when your dataset is small

partitioning in relational databases vs. NoSQL

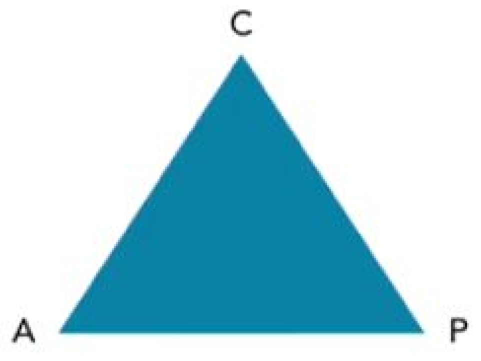
- relational databases can be partitioned horizontally or vertically
- horizontal partitioning puts different rows on different partitions
- vertical partitioning puts different columns on different partitions
- partitioning non-relational databases depends on type

what is the CAP theorem?

- proposed by Eric Brewer in 2000
- originally conceptualized around network shared data
- often used to generalize the tradeoffs between different databases

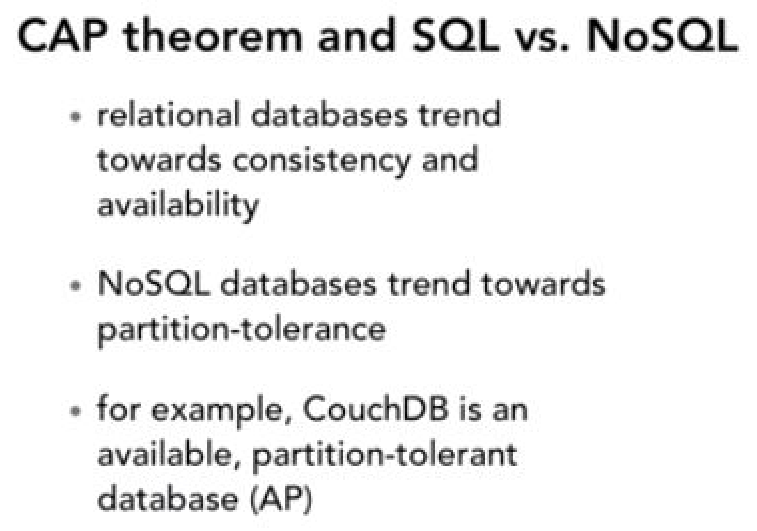
three desirable properties

- consistency - all users get the same data, no matter where they read it from
- availability - users can always read and write
- partition tolerance - works when divided across a network



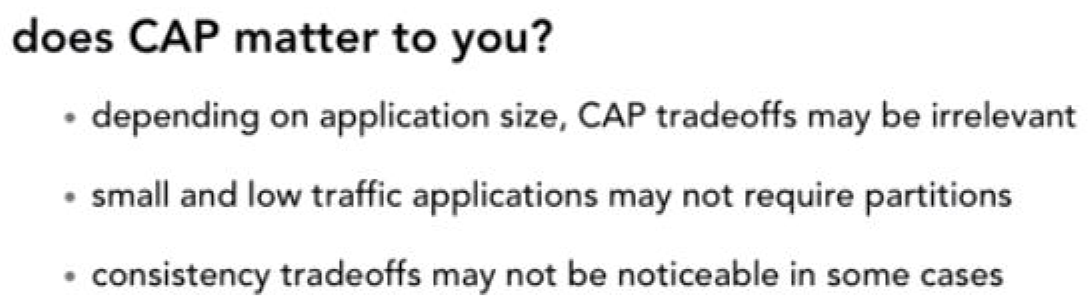
the theorem

- at most, you can only guarantee two of these three properties simultaneously
- available, partition-tolerant (AP)
- consistent, partition-tolerant (CP)
- consistent, available (CA)



CAP theorem and SQL vs. NoSQL

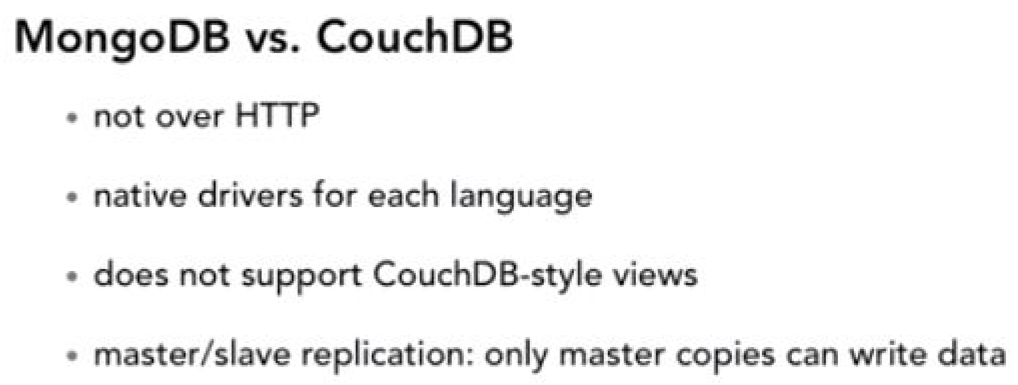
- relational databases trend towards consistency and availability
- NoSQL databases trend towards partition-tolerance
- for example, CouchDB is an available, partition-tolerant database (AP)



does CAP matter to you?

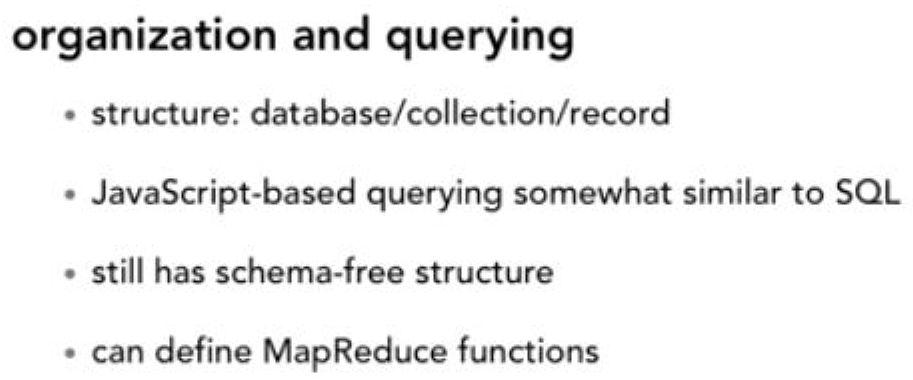
- depending on application size, CAP tradeoffs may be irrelevant
- small and low traffic applications may not require partitions
- consistency tradeoffs may not be noticeable in some cases

FEATURES OF MONGO DB



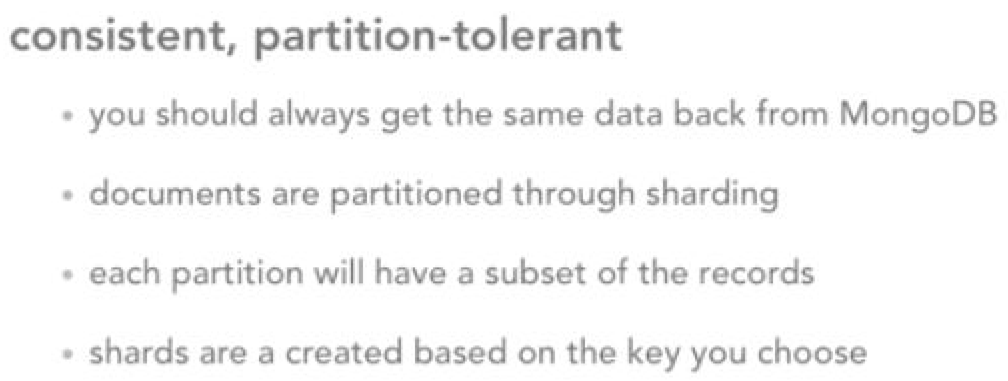
MongoDB vs. CouchDB

- not over HTTP
- native drivers for each language
- does not support CouchDB-style views
- master/slave replication: only master copies can write data



organization and querying

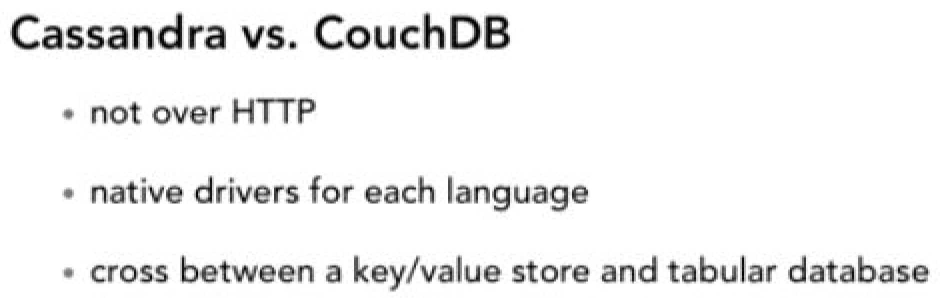
- structure: database/collection/record
- JavaScript-based querying somewhat similar to SQL
- still has schema-free structure
- can define MapReduce functions



consistent, partition-tolerant

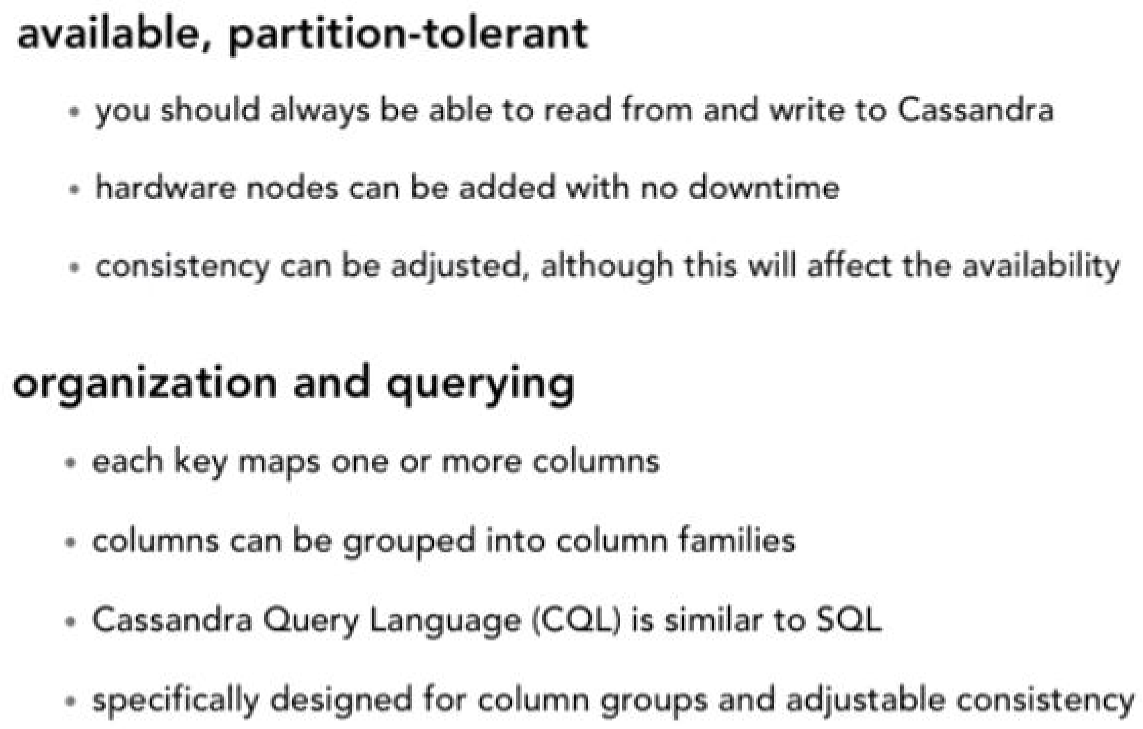
- you should always get the same data back from MongoDB
- documents are partitioned through sharding
- each partition will have a subset of the records
- shards are a created based on the key you choose

FEATURES OF CASSANDRA



Cassandra vs. CouchDB

- not over HTTP
- native drivers for each language
- cross between a key/value store and tabular database



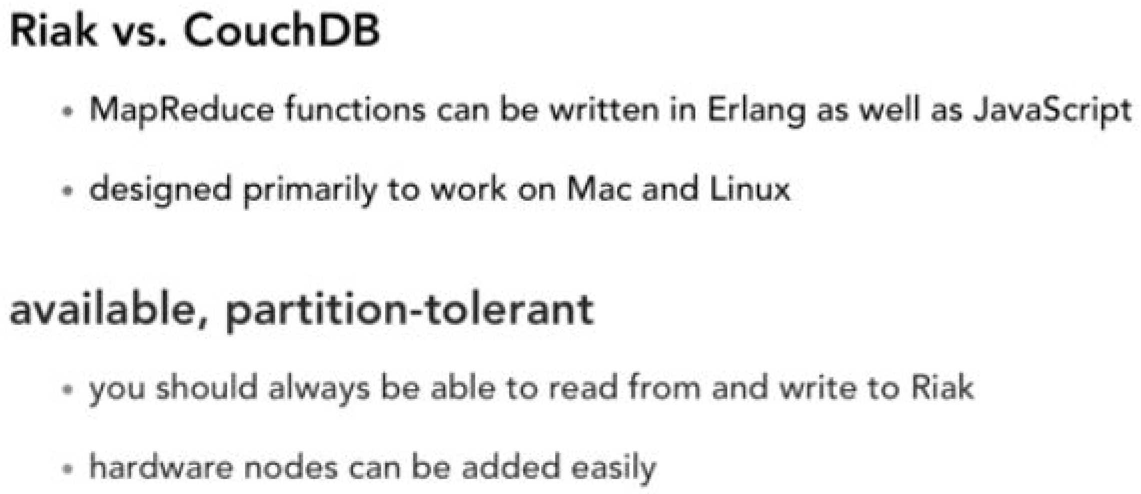
available, partition-tolerant

- you should always be able to read from and write to Cassandra
- hardware nodes can be added with no downtime
- consistency can be adjusted, although this will affect the availability

organization and querying

- each key maps one or more columns
- columns can be grouped into column families
- Cassandra Query Language (CQL) is similar to SQL
- specifically designed for column groups and adjustable consistency

FEATURES OF RIAK

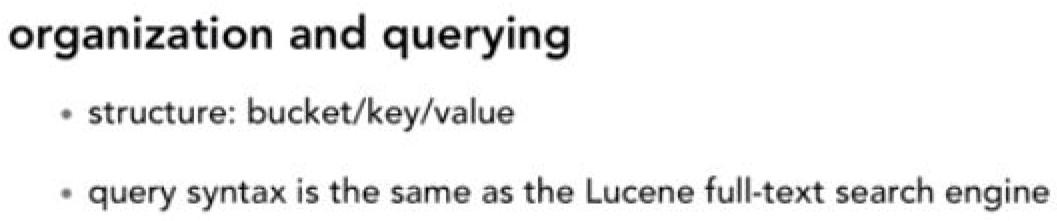


Riak vs. CouchDB

- MapReduce functions can be written in Erlang as well as JavaScript
- designed primarily to work on Mac and Linux

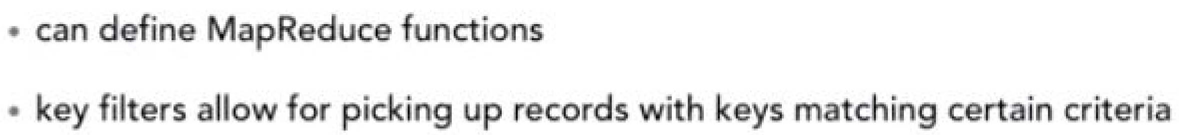
available, partition-tolerant

- you should always be able to read from and write to Riak
- hardware nodes can be added easily



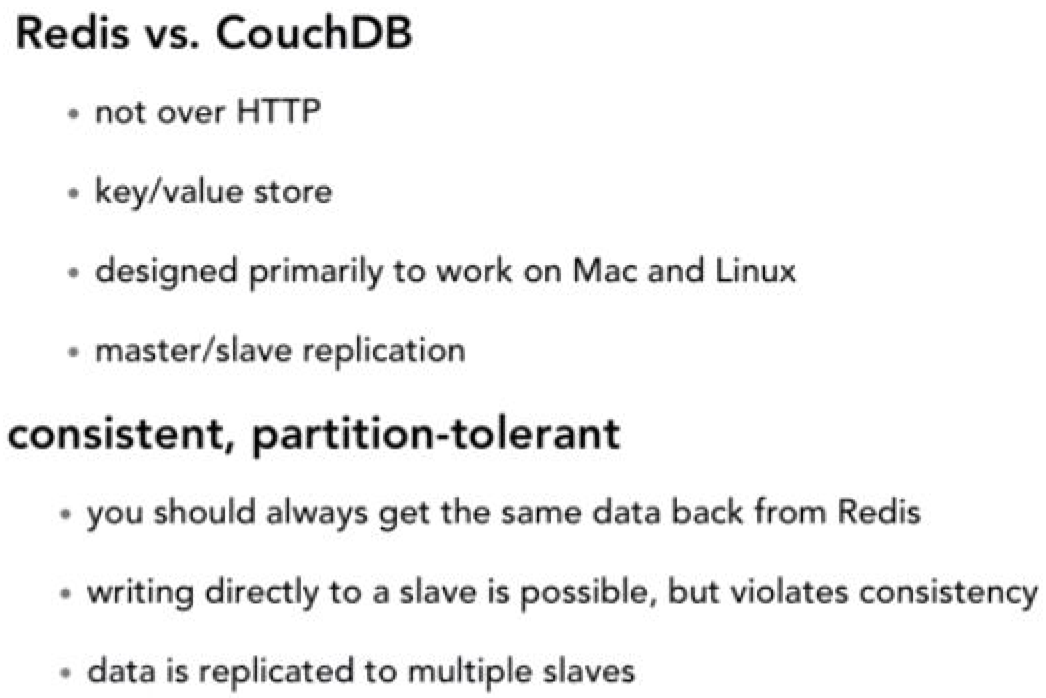
organization and querying

- structure: bucket/key/value
- query syntax is the same as the Lucene full-text search engine



- can define MapReduce functions
- key filters allow for picking up records with keys matching certain criteria

FEATURES OF REDIS



Redis vs. CouchDB

- not over HTTP
- key/value store
- designed primarily to work on Mac and Linux
- master/slave replication

consistent, partition-tolerant

- you should always get the same data back from Redis
- writing directly to a slave is possible, but violates consistency
- data is replicated to multiple slaves

organization and querying

- queries primarily by key
- specific values from hashes within records can be retrieved
- value does not have to be a string, unlike many key/value stores
- lists, sets, and hashes of strings
  - lists are lists of strings
  - hashes are further key/value pairs
  - sets are non-repeating values