

Cloud Native Applications Workshop

Cloud Data Services

WW Developer Advocacy

IBM Developer



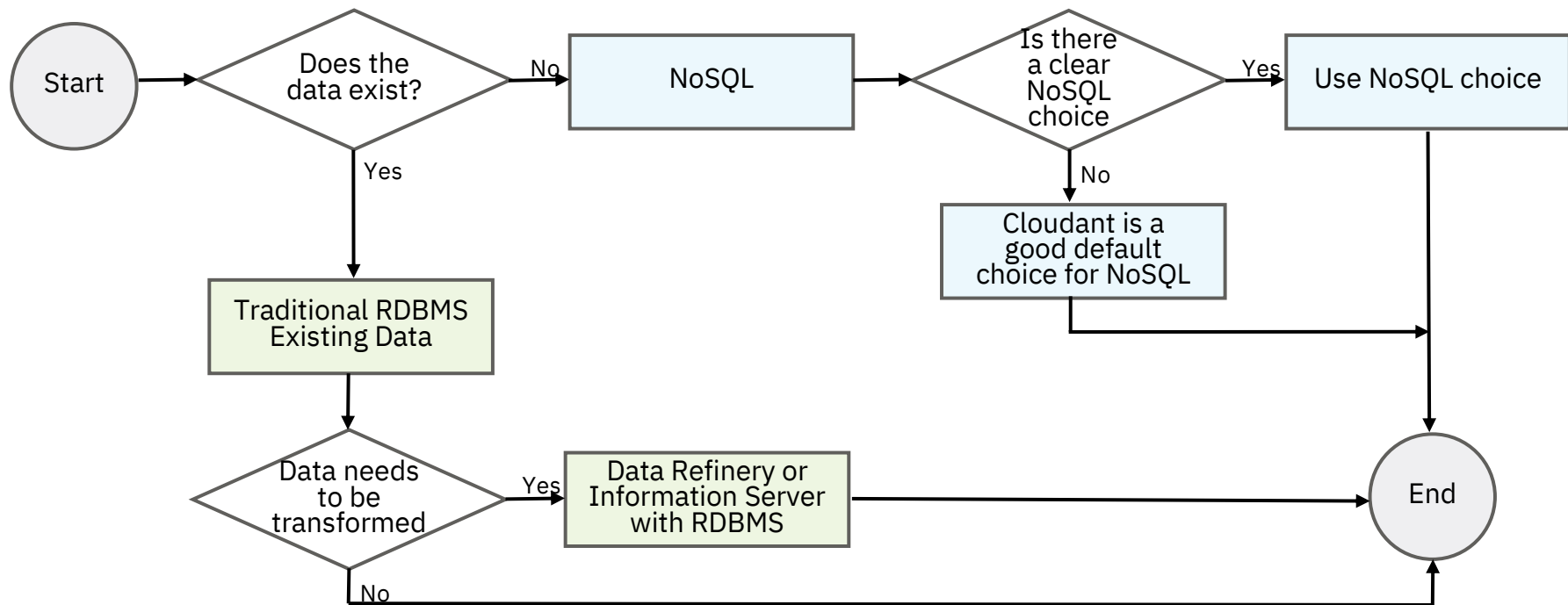
Where and how will
data be used?

Polyglot Persistence

- A basic assumption is that your application will probably have to store some data persistently
 - For longer than the length of a single user session
- At one time, it was a simple assumption that Web programs would run on a relational database
 - Multiple relational database options exist for IBM Cloud Public
- The assumption of using a SQL database is challenged by the data management options collectively called “NoSQL”
- These data stores are grouped into four basic types
 - Key-value
 - Document
 - Column-family
 - Graph

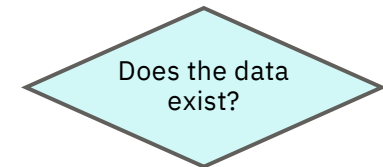


What data service should be used?

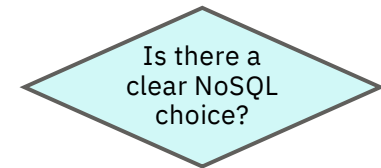


Ask the right questions about the data

- Is there existing data in a relational database management system (RDBMS)?
- For cloud native development with new data don't create a new RDBMS
 - Initially it might seem to be easier and quicker
 - Can hinder rapid development
 - Traditional RDBMS requires structure, a schema
- What does the data "look like"
- Existing data probably already lives in an RDBMS
 - That doesn't mean you need to use it "as is"
 - There are services for data transformation and caching
- NoSQL databases provide:
 - Faster prototyping
 - No need for strict data typing, schema-less
 - No need to throw away non-conforming data



Choosing NoSQL over RDBMS is not the final answer



- Look to see what type of problem you're trying to solve. Let the structure of your data and your queries define your technology choice.
- Simple key-value cache?
 - Use Redis
- Storing documents?
 - Use Cloudant
- Representing graph data?
 - Use Compose for JanusGraph
- Cloudant is a good default NoSQL database

Why use NoSQL?

Why use NoSQL databases

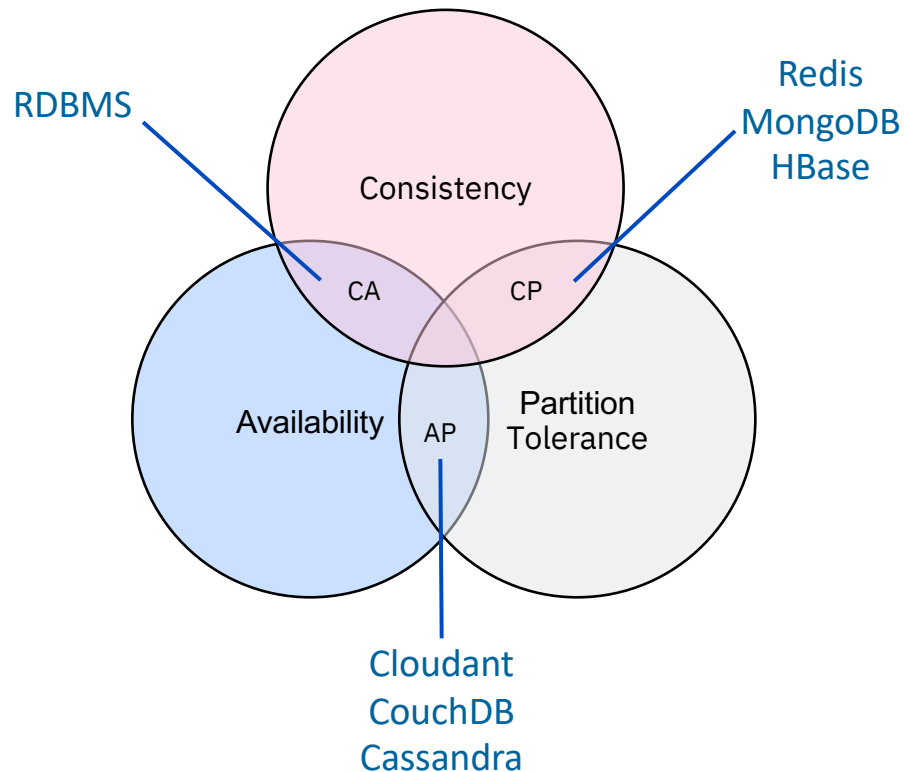
- Some data does not need to be normalized or categorized
 - CLOB, BLOB, reports, customer reviews, and so on
- If the data would be stored as a single unit, it is a waste of effort to write it to columns and tables of an RDBMS
 - A NoSQL database can handle this type of data much more rapidly
- Horizontal scaling
 - Query massive amount of data
 - Parallel processing is a common capability
- Designed to be reliable on unreliable hardware
 - NoSQL databases are distributed

NoSQL Landscape

Database Type	Examples	When best used
Key-Value	Compose for Redis, Memcached, Amazon DynamoDB	Storing interaction data, user preferences, simple shopping carts. Not great at set operations.
Document	Cloudant, Compose for MongoDB	Event logging, content management, analytics. Not great at complex queries.
Column-Family	Cassandra, Db2 Warehouse	Event logging, counters, blogs. Not compatible with ACID transactions.
Graph	GraphDB, Compose for JanusGraph, Neo4J, OrientDB	Social Networks, Location-based Services. Not optimal for certain types of bulk updates.

Attributes of NoSQL databases

CAP Theorem



CAP Theorem says you can only two:

- Consistency
- Availability
- Partition Tolerance

Horizontal replicas can become partitioned

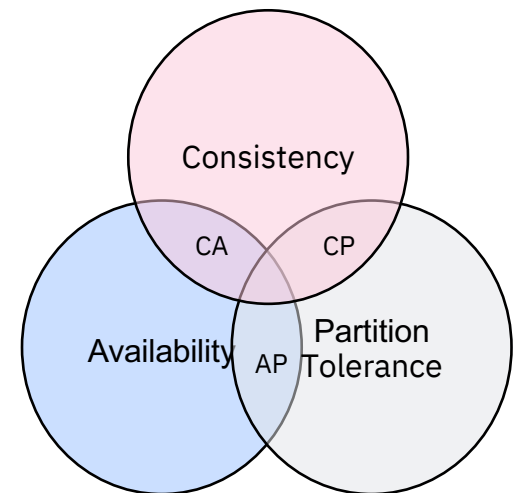
- Consequence: Choose always consistent or always available (can't have both!)
- Configurable in many NoSQL databases

Eventually consistent

- To maintain availability, partitioned replicas lose synchronization

NoSQL and eventual consistency

- Eventual consistency
 - Many NoSQL databases use replicas (horizontal scaling) to maintain high availability
 - The network between database replicas can become partitioned
 - CAP Theorem: When partitioned, either consistency or availability must be sacrificed
 - To maintain availability, most databases are configured to sacrifice consistency
 - When the partitioning is resolved, synchronization reestablishes consistency
- Consequences
 - Transactions are not ACID across replicas
 - Updates are inconsistent until they replicate (usually milliseconds)
 - Applications must tolerate the possibility that two reads may produce inconsistent results
- Be aware of the limitations of your particular database choice
 - Don't assume that your data is always consistent!



Eventual consistency works well

- Eventual consistency sounds problematic
 - Many business domains are eventually consistent (hotels, airlines, and so on)
- How is conflict resolution handled?
 - “Last writer wins” can overwrite data, resolution is non trivial
 - Reconciliation can be achieved by time stamps (epochs, vector clocks)
- CouchDB (Cloudant) uses Multi-Version Concurrency Control (MVCC)
 - No locking required
 - Each read request sees the most recent snapshot of the database
- Database detects update conflicts
 - Application or user can reconcile the conflicts

Understanding NoSQL data retrieval

- There are several approaches for accessing data
 - These are not necessarily specific to NoSQL
 - Understanding these options can help in deciding which NoSQL to use
- Four common approaches
 1. SQL-style query
 2. Look-up by ID
 3. Object navigation
 4. Map-reduce

SQL-style queries

- If you have always used SQL queries, your natural tendency is to use them for your new cloud app

```
SELECT * FROM customers WHERE postal_code = '51922'
```

- Database considers all elements, filters out those that don't match



• Don't Do It

- Sequential queries perform poorly with most NoSQL databases
- NoSQL databases provide APIs for other methods to retrieve your data much more efficiently

Lookup by ID

- Do you already know the key or ID of the data you need?
 - Primary key access
 - Typical for relational databases
 - Available for most NoSQL databases

Yes



Key Value Pair Retrieval

- Provide the key and retrieve the values
- A key-value database fits this scenario well

No



This requires another approach for accessing the data

- Object navigation
- Map-reduce

Many NoSQL databases provide built-in map-reduce functionality

Lookup by ID in a Key-value Store Databases

- Values are stored in relation to unique keys
- Values are retrieved by supplying their key

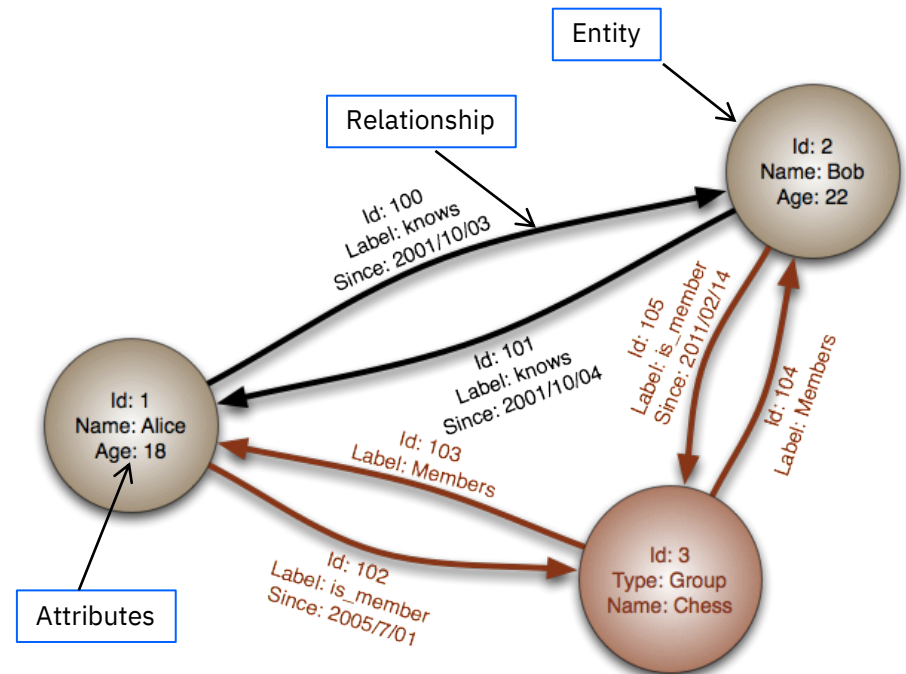
```
String key    = "location";  
Object value = find(key);
```

- Value data usually has no known structure
 - Any kind of data can be stored as a value
 - Very flexible
 - Can even contain key to a document database

Key	Value
firstName	Bugs
lastName	Bunny
location	{"number": 1234, "street": "Bunny Hole Ln", "planet": "Earth"}
picture	X89 PNG \r \n x1A \n
factfile	10930231543

Object Navigation

- Entity-Relationship-Attribute (ERA)
 - Each entity has attributes
 - Entities are connected by relationships
 - a.k.a. Nodes-Edges-Properties
- Implemented in graph databases
 - Examples: Compose for JanusGraph, Neo4J, OrientDB
 - Most NoSQL databases have this capability
- Object-oriented
 - Entities are objects
 - Relationships are links
 - Attributes are instance variables
- Retrieval done by following relationships
 - “Give me the members of the Chess Club”
 - Follows links rather than searching multiple tables – more efficient

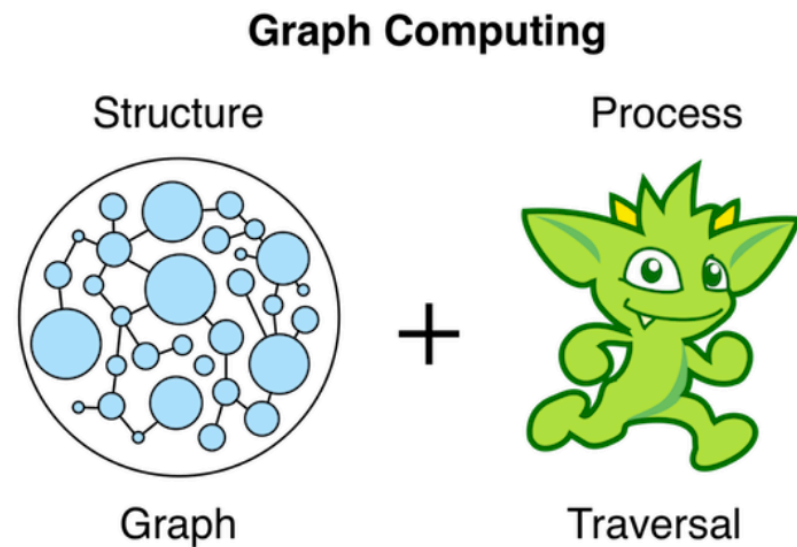


Why use a graph database?

- Relationships are important in graph databases
 - Allow more complex relationships than relational databases
 - Allow for dynamic relationships
 - Can describe relationships in greater detail
- Graph databases can be schema-less
 - Allows flexibility in data typing
- Can apply Graph Theory to databases
 - Discover disjoint sets within the data
 - Find minimal routes between nodes

Property Graphs

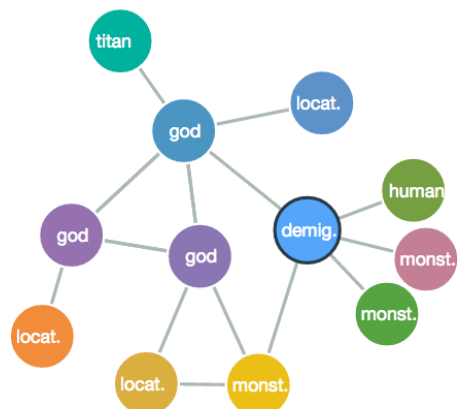
- Contains vertices and edges with properties
- Vertex represents an entity
 - A vertex has a label
 - Vertices have unique IDs
 - Vertices have properties
- Edge represents a directional relationship
 - Edges have labels
 - Each edge has a unique ID
 - Edges have properties
- Graph computing makes a distinction between the structure of the graph and the process of navigating the graph
 - JanusGraph based on TinkerPop3 and the traversal language is Gremlin



The JanusGraph data browser

```
def g=ConfiguredGraphFactory.open("example").traversal();def saturn=g.V().has("name", "saturn");saturn.V
```

```
1  ▾ [
2  ▾ {
3  ▾   "objects": [
4      {
5        "id": 4296,
6        "label": "titan"
7      },
8      {
9        "id": 4128,
10       "label": "god"
11      },
12      {
13        "id": "4qs-36o-6c5-3bc",
14        "label": "father"
15      },
16    ]
17  }
```



Vertices: 12

Filter: Label Type Properties

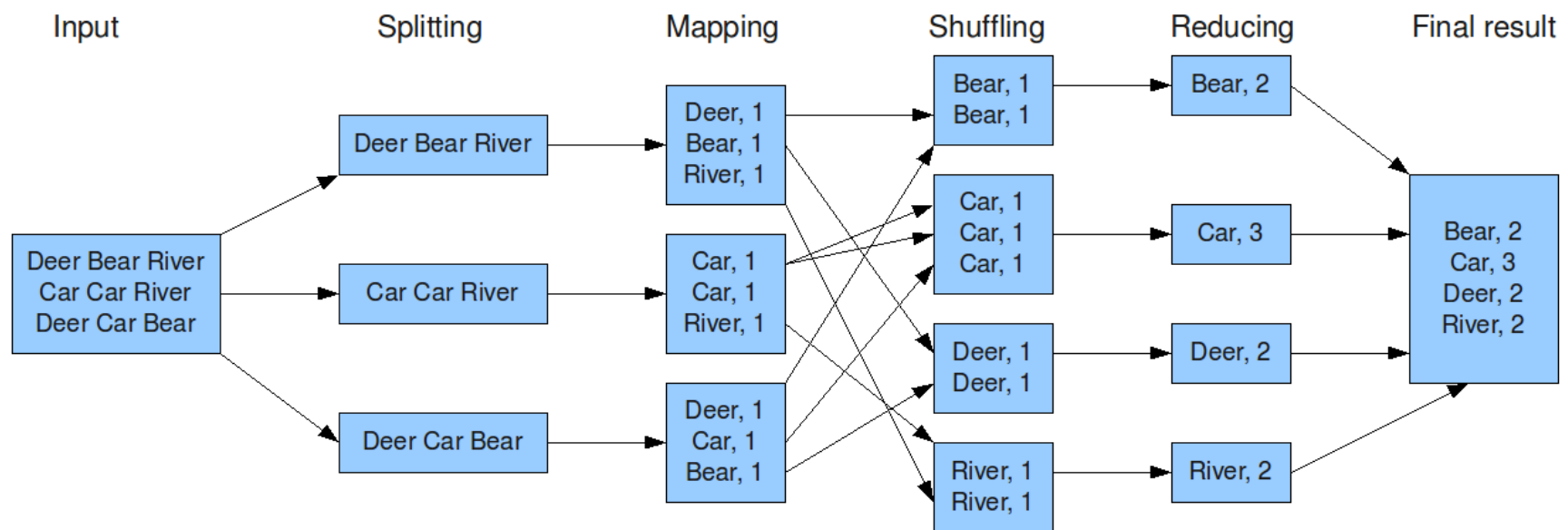
Map-Reduce

- Programming model designed to exploit parallel processing opportunities of clusters
 - Invented at Google Research
 - Horizontal scalability – job can be split across multiple servers
 - Hadoop is most well known implementation
 - Many NoSQL databases provide this capability



- Comprised of 2 major steps:
 1. Map
 - Converts a set of data into intermediate key / value pairs
 2. Reduce
 - Takes the output of Map step as input
 - Combines output of many map steps into a smaller set of key / value pairs

Consider this Example – How many times is each word used?



Note that this method could be used to answer other questions:
What word is used most, least?

Implementation: Cloudant

Introduction to Cloudant

A fully managed NoSQL Database as a Service

Transactional JSON *document* database with RESTful API

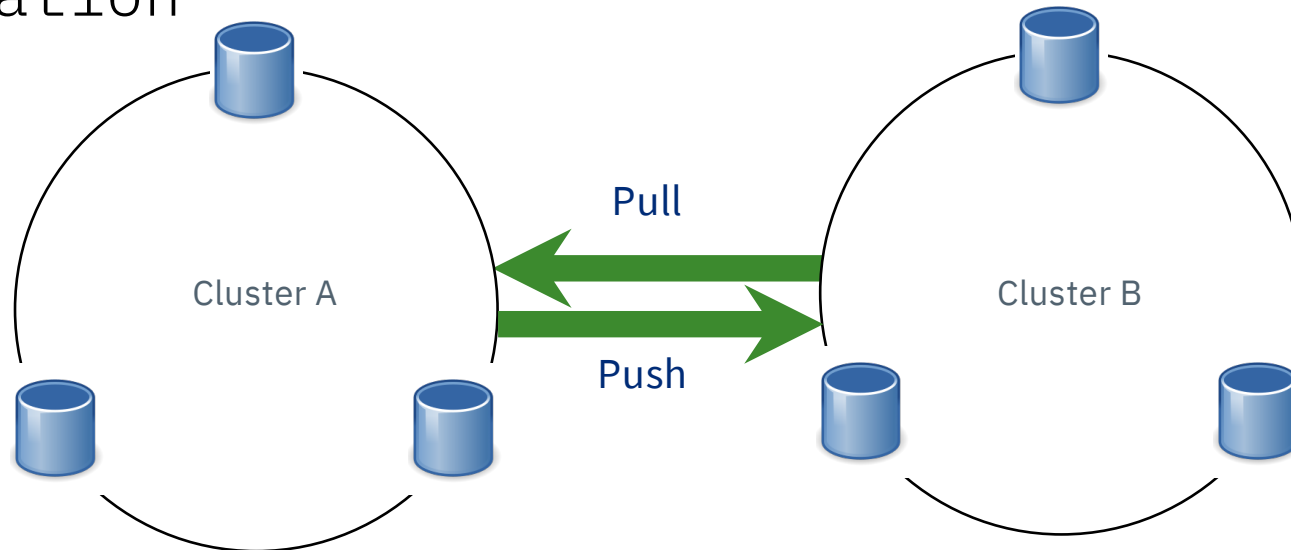
Can spread data across data centers and devices for scale plus high availability (HA)

Ideal for apps that require these features:

- Massive, elastic scalability
- High availability
- Geolocation services
- Full-text search
- Occasionally connected users

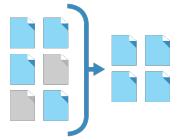
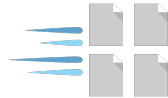


Cloudbant database replication



- Bi-directional (push/pull == sync)
- Continuous or point-in-time
- Filterable (replicate subset of docs in a database)
- Durable (replication will pick up where it left off)
- Allows for eventual consistency
- Separate clusters
 - In different regions, active-active
 - Can be used to cache data on a phone

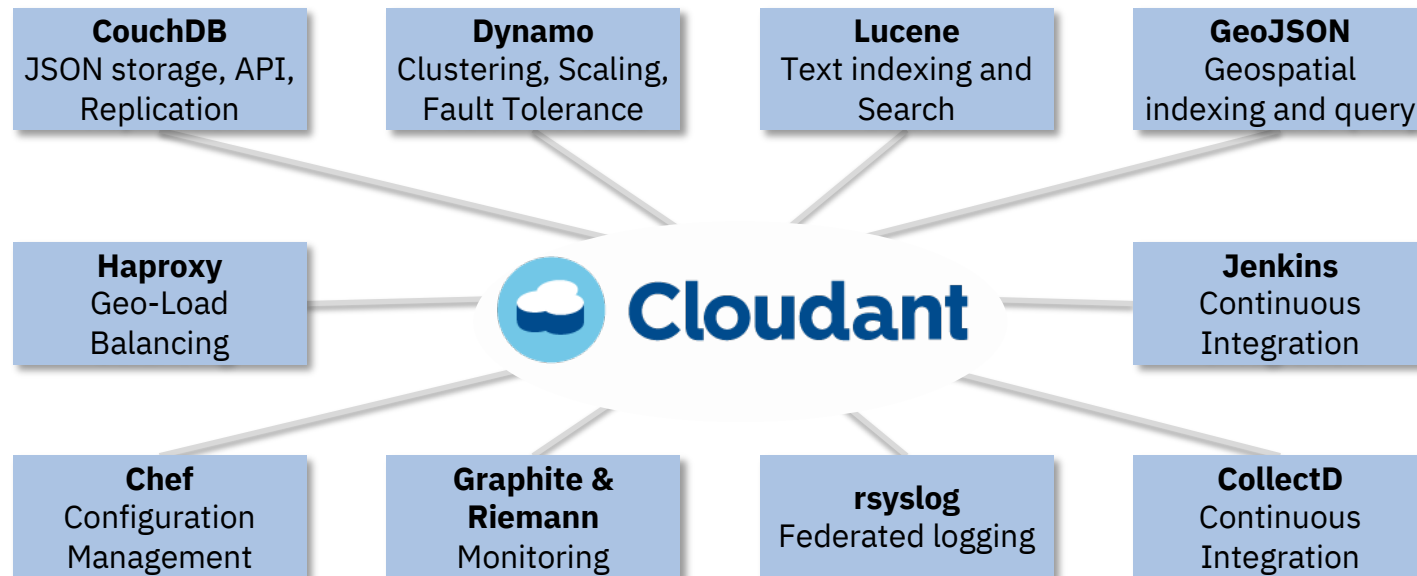
Cloudant API and Query Options



JSON Documents	Primary Index	Secondary Indices	Search	Cloudant Geospatial	Cloudant Query
<ul style="list-style-type: none"> Direct doc lookup by <code>_id</code> Use when you want a single document and can find by the doc <code>_id</code> 	<ul style="list-style-type: none"> Exists OOTB Stored in a b-tree Primary key \rightarrow <code>doc._id</code> Use when you can find documents based on their <code>_id</code> Pull back a range of keys 	<ul style="list-style-type: none"> Built using MapReduce Stored in a B-tree Key \rightarrow user-defined field(s) Use when you need to analyze data or get a range of keys Ex: count data fields, sum/average numeric results, advanced stats, group by date, etc. 	<ul style="list-style-type: none"> Built using Lucene FTI: Any or all fields can be indexed Ad-hoc queries Find docs based on their contents Can do groups, facets, and basic geo queries (bbox & sort by distance) 	<ul style="list-style-type: none"> Stored in R* tree Lat/Long coordinates in GeoJSON Complex geometries (polygon, circularstring, etc.) Advanced relations (intersect, overlaps, etc.) 	<ul style="list-style-type: none"> Mongo-style querying Not a 1:1 mapping Built natively in Erlang Ad-hoc queries Lots of operators (<code>></code>, <code><</code>, <code>IN</code>, <code>OR</code>, <code>AND</code>, etc.) Intuitive for people coming from Mongo or SQL backgrounds

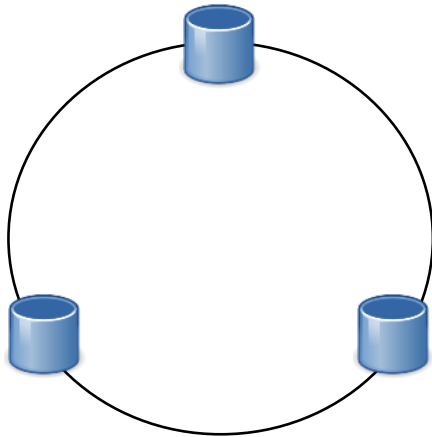
Cloudbant and other technologies

- Cloudbant combines the best open source technology and thinking to create the most scalable, flexible, always-on DBaaS for big mobile and the Internet of Things (IoT)

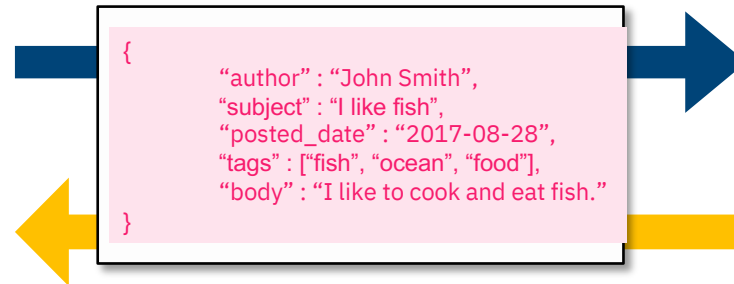


Cloudant HTTP RESTful API

- CRUD operations, document retrieval using indexes, search
- Agnostic to programming language
- Apache CouchDB compatibility



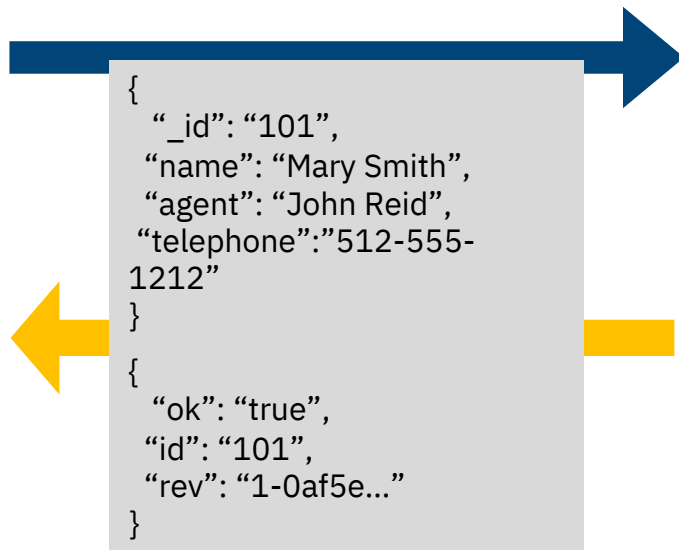
GET */<database>/<doc._id>*



Cloudant API: Inserting a document - HTTP PUT or POST

- using **POST** with `_id` in the document body:

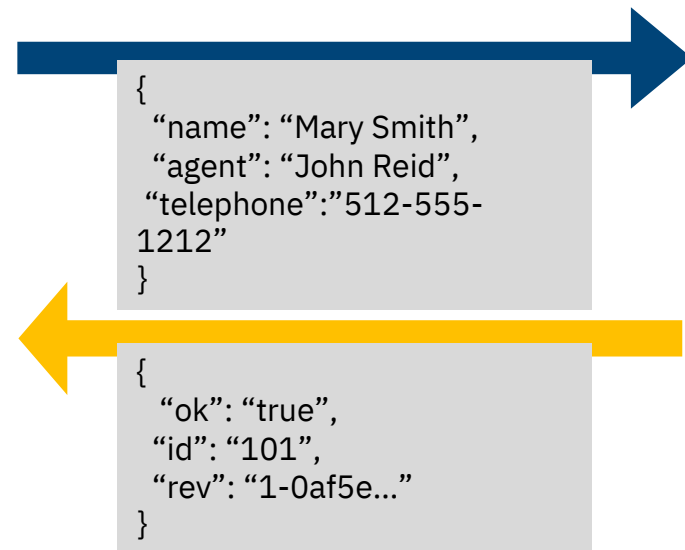
POST <https://<username>.cloudant.com/authors>



- using **PUT** with `_id` in the URL:

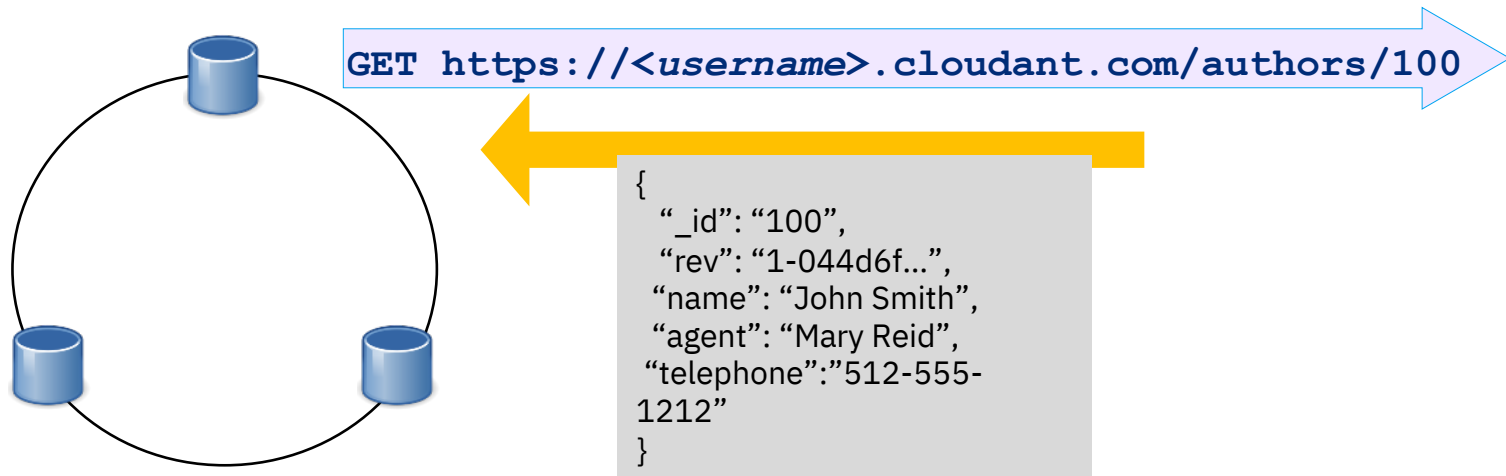
PUT

<https://<username>.cloudant.com/authors/101>



Cloudant API: Returning a single document

- HTTP GET with database name and _id of document
- For example, to get document with _id **100** from the **authors** database:



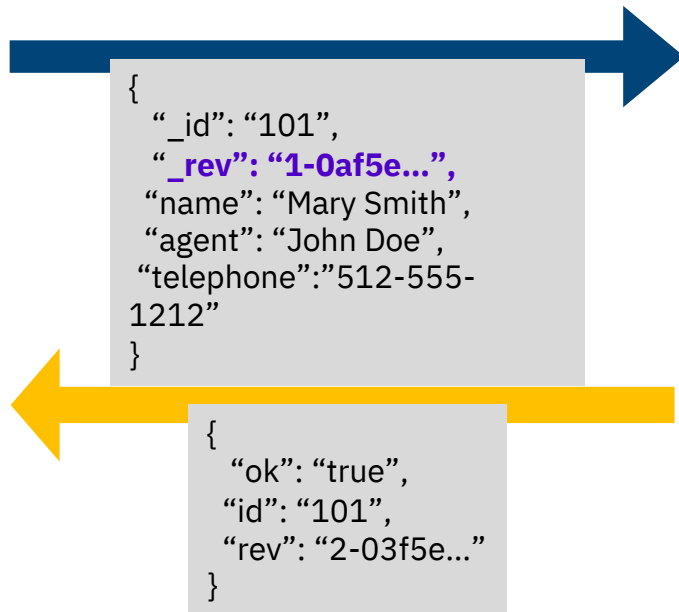
Cloudant API: Updating a document – HTTP PUT or POST

Same HTTP operations as insert; If `_id` exists, it is an update

Latest `_rev` is required or operation fails

- using **POST** with `_id` in the document body:

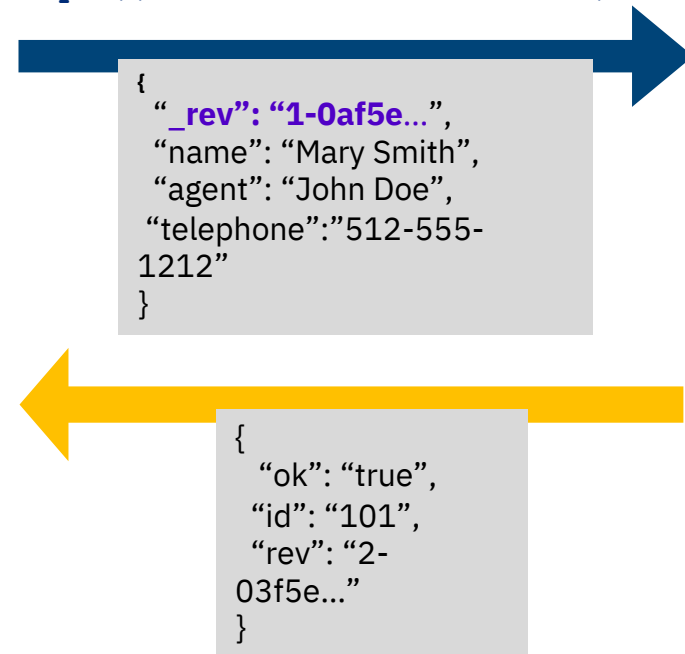
POST <https://<username>.cloudant.com/authors>



- using **PUT** with `_id` in the URL:

PUT

<https://<username>.cloudant.com/authors/101>

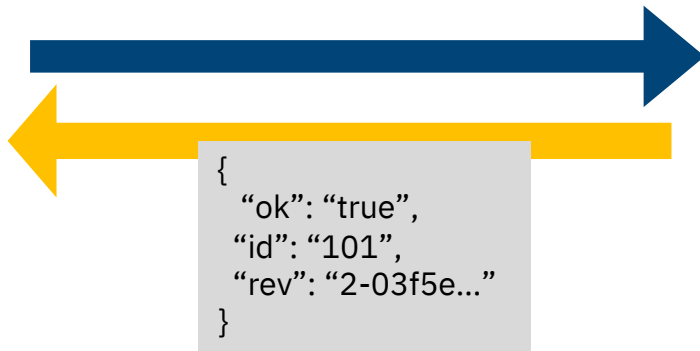


Cloudant API: Deleting a document – HTTP DELETE or PUT

using **DELETE** with `_id` and `_rev` in the document body:

DELETE

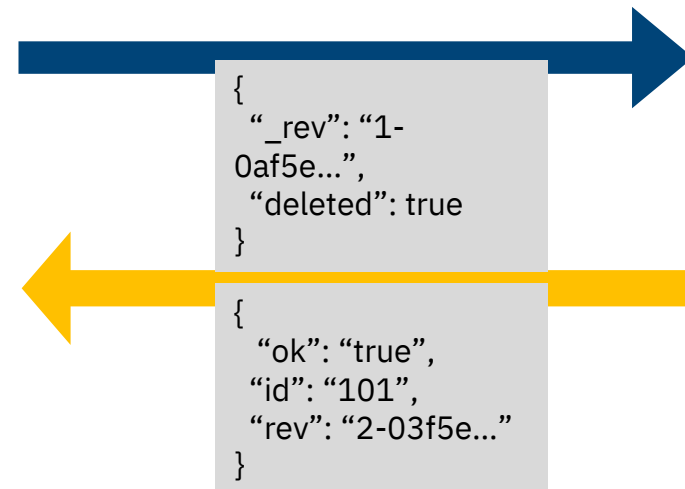
`https://<username>.cloudant.com/authors/101/?rev=...`



using **PUT** with `_id` in the URL, `_rev` and `deleted:true` in document body :

PUT

`https://<username>.cloudant.com/authors/101`



`_rev` is required or operation fails

Cloudant documents: `_id` and `_rev`

Each document has an `_id` (ID) field that is unique per database

- Any string can be supplied as an `_id`, but it is recommended that you allow Cloudant to generate a UUID (universally unique identifier)

```
{  
  "_id" : "7f123e23a328bd50ee123cd35452ae47",  
  "_rev" : "2-3123414209",  
  "title" : "IBM Cloudant Redbook",  
  "author" : "Christopher Bienko"  
}
```

There is also a unique `_rev` (revision number) field per document

- Generated by an md5 hash of the transport representation of the document
- *N*-prefix reflects the number of times this document has been updated
- Updates to existing documents must provide the latest `_rev` value
 - Otherwise the update request is rejected

References (1 of 2)

Cloudant

- <https://cloudant.com/learning-center/>

“RDBMS to NoSQL: Reviewing Some Next-Generation Non-Relational Database's”

- <http://liacs.leidenuniv.nl/~stefanovtp/courses/StudentenSeminarium/Papers/DB/3.IJAEST-Vol-No-11-Issue-No-1-RDBMS-to-NoSQL-Reviewing-Some-Next-Generation-Non-Relational-Database's-015-030.pdf>

Eventual consistency

- <http://guide.couchdb.org/draft/consistency.html>

“NoSQL Distilled”

- <http://www.amazon.com/dp/0321826620>

Extract, Transform, and Load (ETL)

- https://en.wikipedia.org/wiki/Extract,_transform,_load

Graph database introduction

- <https://www.compose.com/articles/graph-101-getting-started-with-graphs/>

References (2 of 2)

JanusGraph concepts

- <https://help.compose.com/docs/janusgraph-concepts>

Introduction to graph concepts

- <https://tinkerpop.apache.org/docs/3.2.3/reference/#intro>

Compose for JanusGraph on IBM Cloud

- <https://console.bluemix.net/docs/services/ComposeForJanusGraph/index.html#getting-started-with-compose-for-janusgraph>