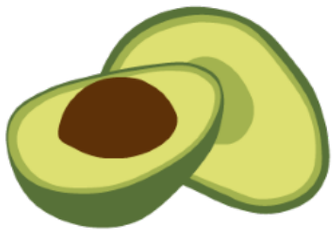
The hunt for the *right* NoSQL database

Why we ❤



ArangoDB

Andreas Jung
@MacYET
info@zopyx.com • www.zopyx.de

/about

- Python developer since 1993
- Freelancer since 2004
- Python, Zope, Plone …

- individual software development
- Electronic Publishing
  (Publishing workflows DOCX→XML→PDF | EPUB | HTML,
  XML consulting)

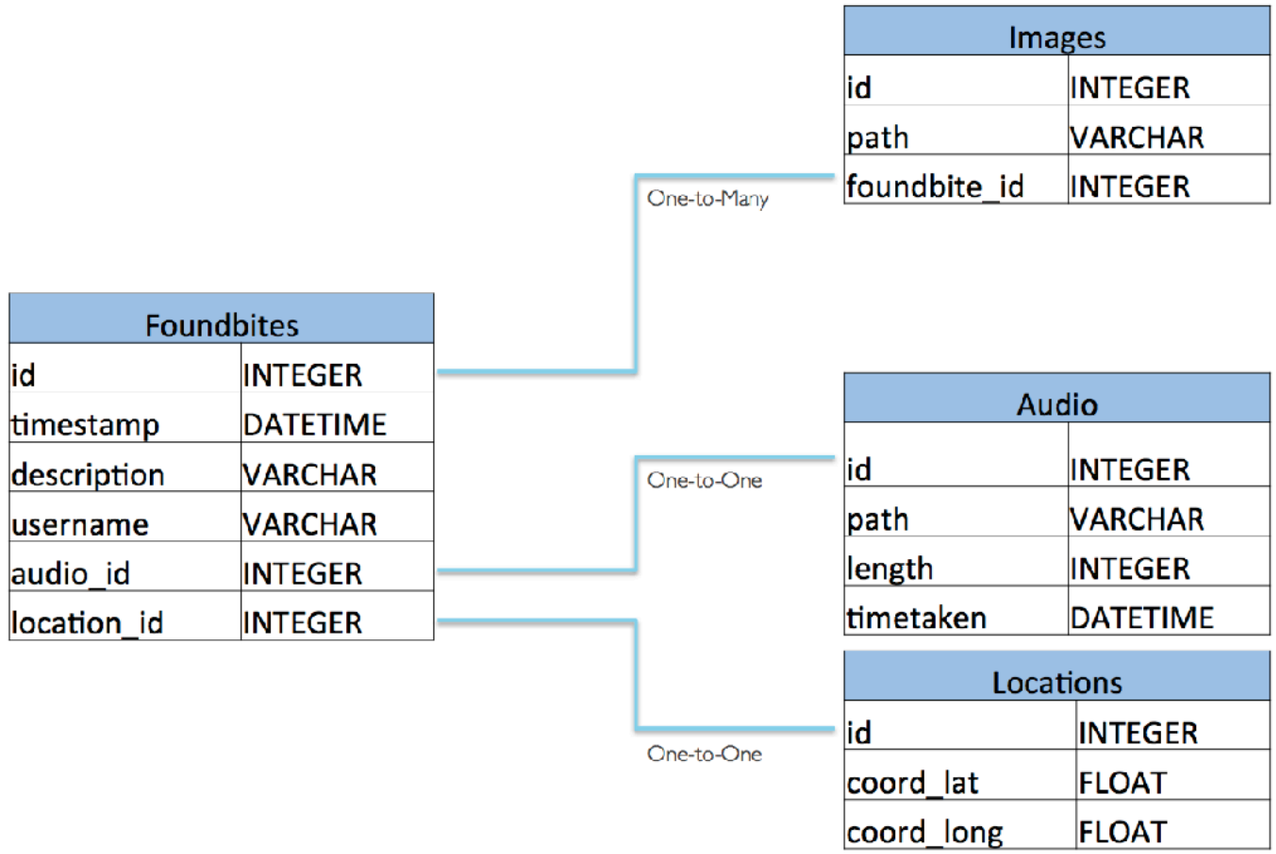
- Founded publishing projects
  - XML-Director
  - Produce & Publish

Disclaimer

- This talk is completely
  - biased
  - opinionated
  - unscientific
  - not affiliated with ArangoDB GmbH

Relational databases

- well understood
- **common** data model
- long history:
  - System R (1974)
  - Oracle (1979)



- **S**tructured **Q**uery **L**anguage (Standards: ISO/IEC 9075 + 13249)
- theoratically interoperable if you stick to the SQL standard

~~NoSQL = not SQL~~

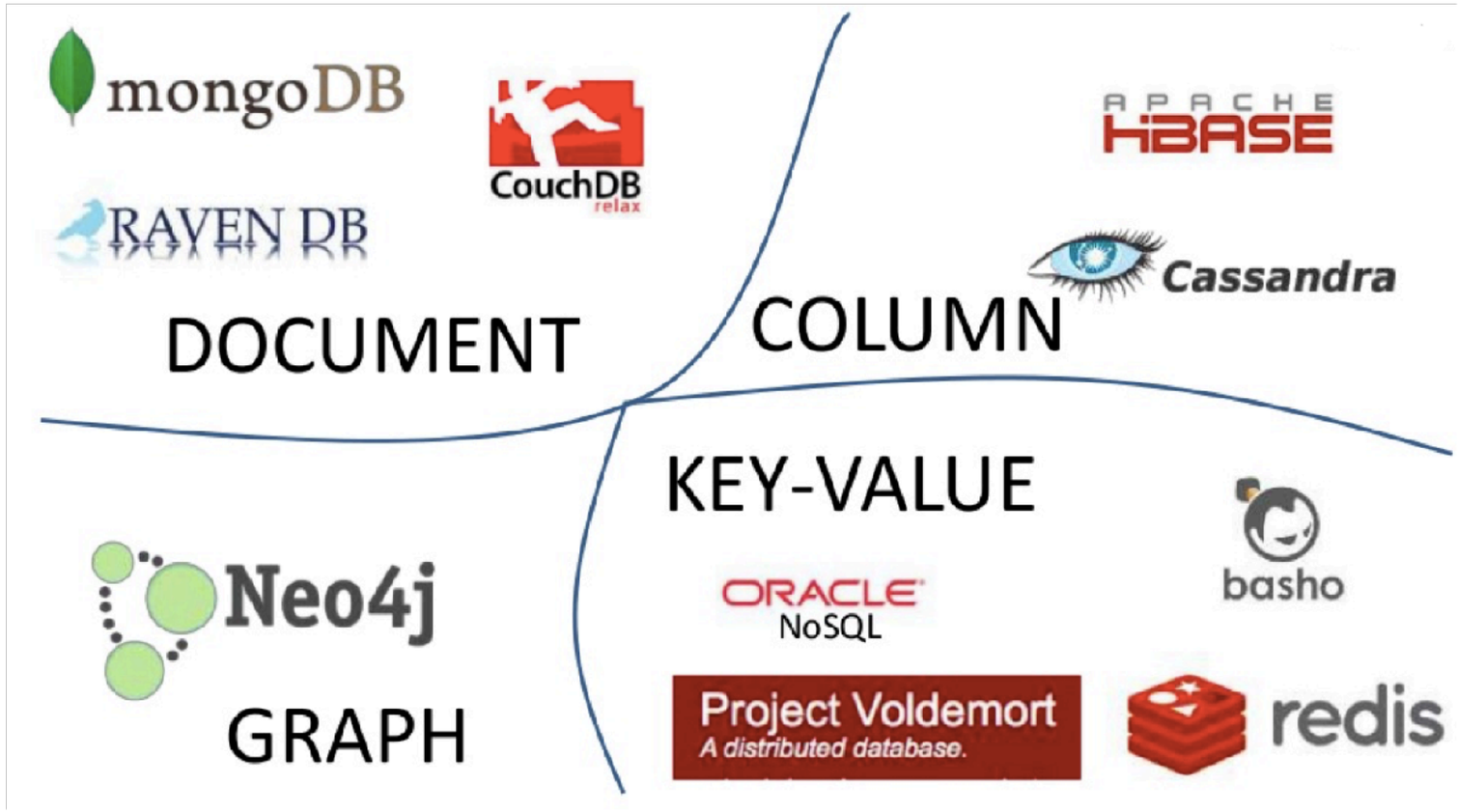
NoSQL = non-relational

*“**NoSQL is not about performance, scaling, dropping ACID or hating SQL — it is about choice.** As NoSQL databases are somewhat different it **does not help** very much **to compare** the databases by their **throughput** and chose the one which is faster. Instead—the user should **carefully think** about his **overall requirements** and weight the different aspects. Massively scalable key/value stores or memory-only systems can archive much higher benchmarks. **But your aim is to provide a much more convenient system for a broader range of use-cases—which is fast enough for almost all cases.**”*

Jan Lenhardt (CouchDB)

Categories of NoSQL databases

- **Key-Value**
  - Memcached, Redis, Riak, …
- **Column-oriented**
  - Cassandra, …
- **Document (JSON)**
  - MongoDB, CouchDB, …
- **Tabular**
  - Big Table, Hase, …
- **Graph**
  - Neo4J, …
- **XML**, **Object**…
  - eXist, BaseX, Marclogic, ZODB, …



http://martinfowler.com/articles/nosqlKeyPoints.html
https://www.youtube.com/watch?v=qI_g07C_Q5I

New challenges

- Cloud
- Replication
- massive data explosion: „Big Data“

- Globally distributed systems
- Specialized requirements

➡ **more specialized databases**

➡ **Relational databases are no longer the only option**

CAP Theorem

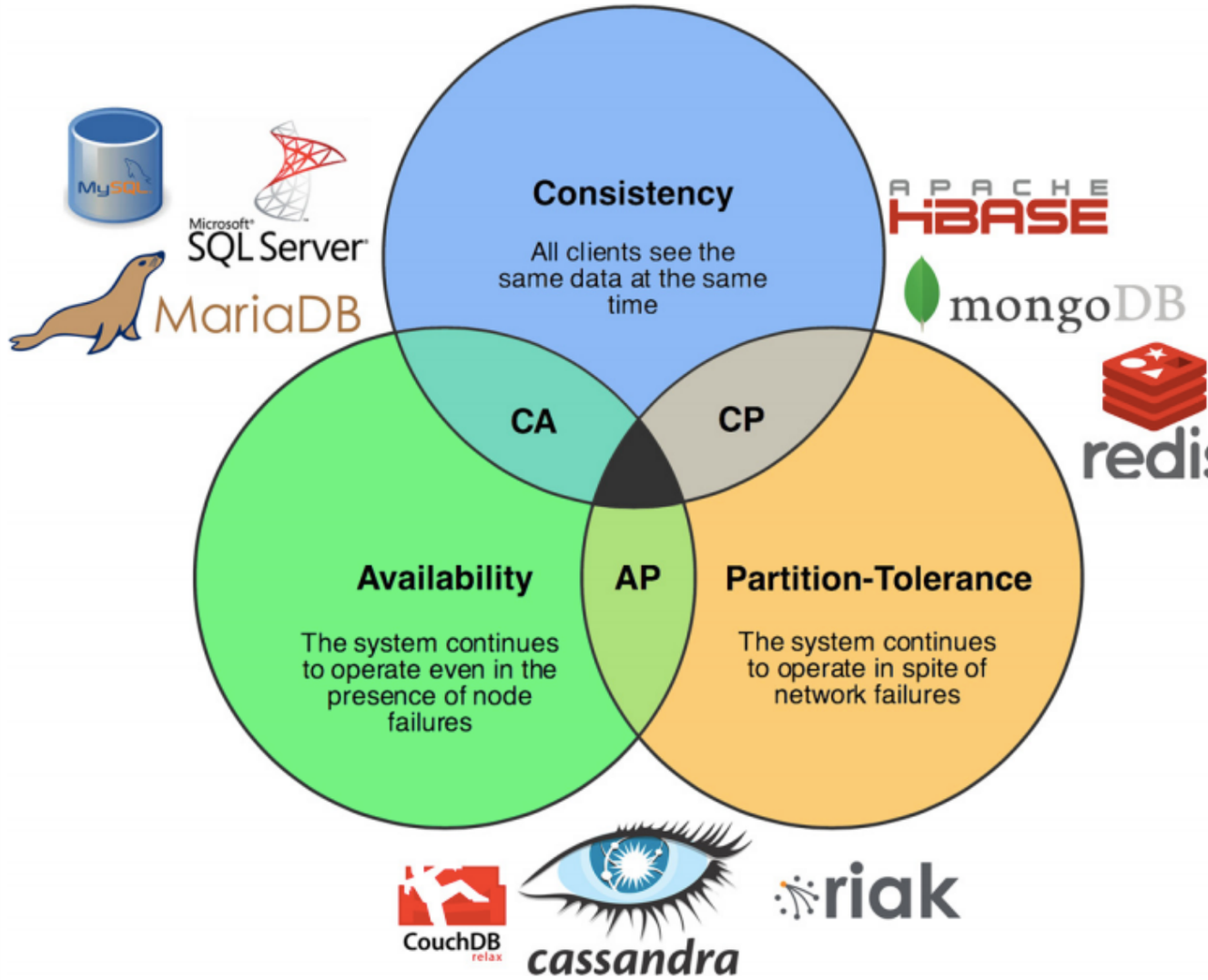
It is **impossible** for a distributed computer system to simultaneously **provide all three** of the following guarantees:

- **Consistency**
  (every read receives the most recent write or an error)

- **Availability**
  (every request receives a response, without guarantee that it contains the most recent version of the information)

- **Partition tolerance**
  (the system continues to operate despite arbitrary partitioning due to network failures)

(Eric Brewer)



PICK TWO

My personal hunt for a multi-purpose NoSQL database …

- Should fit most mid-size projects
- Document store (+ graphs)
- Arbitary query options
- Cross-table/collection relationships
- (optional) transactional integrity (ACID) across multiple documents and operations
- replication/clustering

My personal hunt...

RethinkDB

mongoDB.

FOUNDATIONDB

cassandra

ArangoDB

OrientDB®

MarkLogic®

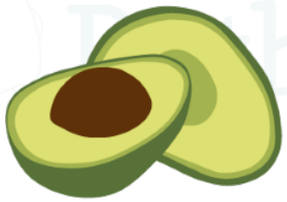
elastic

ToroDB

CouchDB

...and various others

My personal hunt…



ArangoDB

FOUNDATIONDB

MarkLogic®

…and various others

~~FOUNDATIONDB~~

bought in 2015 by

MarkLogic® The high-end $$$$ solution

- the most professional, feature-complete, feature-rich NoSQL database ever

- document (XML/JSON) store and graph database

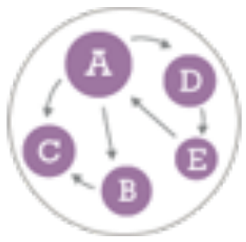
- focus on data integration and data consolidation

- expensive but worth the money **if** you need the features

- widely used in enterprises (saved „Obama-Care“ project)

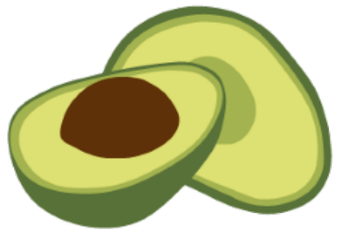
ArangoDB

A native multi-model database



- **Document store (JSON)**
  - JOINs, secondary indexes, ACID transactions
- **Key-value store**
- **Graph database**
  - integrates with document store
  - rich graph query operations
  - nodes and edges can contain complex data

➡ **all models can be combined**



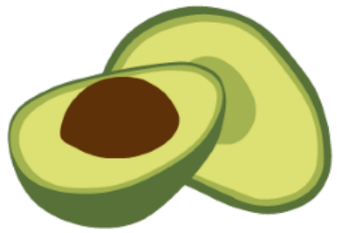
ArangoDB

foxx

Foxx framework

- implement your own REST micro-services directly with Javascript running inside ArangoDB
- unified data storage logic (decouples API from external services)
- reduced network overhead (no network latencies)
- you can use the full JS Stack
- batteries included
- build-in job queue

```
router.get((req, res) => {
  res.write(`
    Hello :D
    This is your database speaking.
  `)
})
```



ArangoDB

AQL - One Query Language to rule them all

- AQL = Arango Query Language
- declarative, human-readable DSL (*I hate JSON queries*)
- document queries, graph queries, joins, all combined in one statement
- ACID support with multi-collection transactions
- easy to understand with some SQL background

ArangoDB

```
FOR u IN users
  FOR f IN friends
    FILTER u.active == true && f.activ
    RETURN u.name
```

```
FOR u IN users
  FOR p IN products
    FILTER u._key == p.recommendedBy
    INSERT { _from: u._id, _to: p._id } IN recommendations
```

```
WITH managers, usersHaveManagers
FOR v, e, p IN (
  RETURN { v, e
```

```
FOR u IN users
  FILTER u.active == true
  LIMIT 0, 4
  FOR f IN relations
    FILTER f.type == "friend" && f.friendOf == u.userId
    RETURN {
      "user" : u.name,
      "friendId" : f.thisUser
    }
```

```
TE_NOW() }
```

ArangoDB

```
FOR u IN users
  FOR f IN friends
    FILTER u.active == true && f.active == true && u.id == f.userId
    RETURN u.name
```

```
INSERT { _from: u._id, _to: p._id } IN recommendations
```

```
WITH managers, usersHaveManagers
FOR v, e, p IN (
  RETURN { v, e
```

```
FOR u IN users
  FILTER u.active == true
  LIMIT 0, 4
  FOR f IN relations
    FILTER f.type == "friend" && f.friendOf == u.userId
    RETURN {
      "user" : u.name,
      "friendId" : f.thisUser
    }
```

```
TE_NOW() }
```

ArangoDB

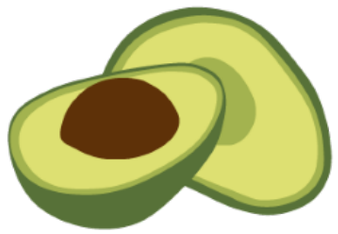
```
FOR u IN users
  FOR f IN friends
    FILTER u.a
    RETURN u.n
```

```
FOR u IN users
  FOR p IN products
    FILTER u._key == p.recommendedBy
    INSERT { _from: u._id, _to: p._id } IN recommendations
```

```
WITH managers, us
FOR v, e, p IN (
  RETURN { v, e
```

```
FOR u IN users
  FILTER u.active == true
  LIMIT 0, 4
  FOR f IN relations
    FILTER f.type == "friend" && f.friendOf == u.userId
    RETURN {
      "user" : u.name,
      "friendId" : f.thisUser
    }
```

```
TE_NOW() }
```



ArangoDB

```
FOR u IN users
  FOR f IN friends
    FILTER u.a
    RETURN u.n
```

```
FOR u IN users
  FOR p IN products
    FILTER u._key == p.recommendedBy
```

```
WITH managers, usersHaveManagers
FOR v, e, p IN OUTBOUND 'users/1' GRAPH 'userGraph'
  RETURN { v, e, p }
```

```
FOR f IN relations
  FILTER f.type == "friend" && f.friendOf == u.userId
  RETURN {
    "user" : u.name,
    "friendId" : f.thisUser
  }
```

```
TE_NOW() }
```

ArangoDB

```
FOR u IN users
  FILTER u.active == true
  LIMIT 0, 4
  FOR f IN relations
  FILTER f.type == "friend" && f.friendOf == u.userId
  RETURN {
    "user" : u.name,
    "friendId" : f.thisUser
  }
```

```
INSERT { name: 'superuser', logins: 1, dateCreated: DATE_NOW() }
UPDATE { logins: OLD.logins + 1 } IN users
```

ArangoDB

```
FOR u IN users
  FILTER u.active == true
  LIMIT 0, 4
```

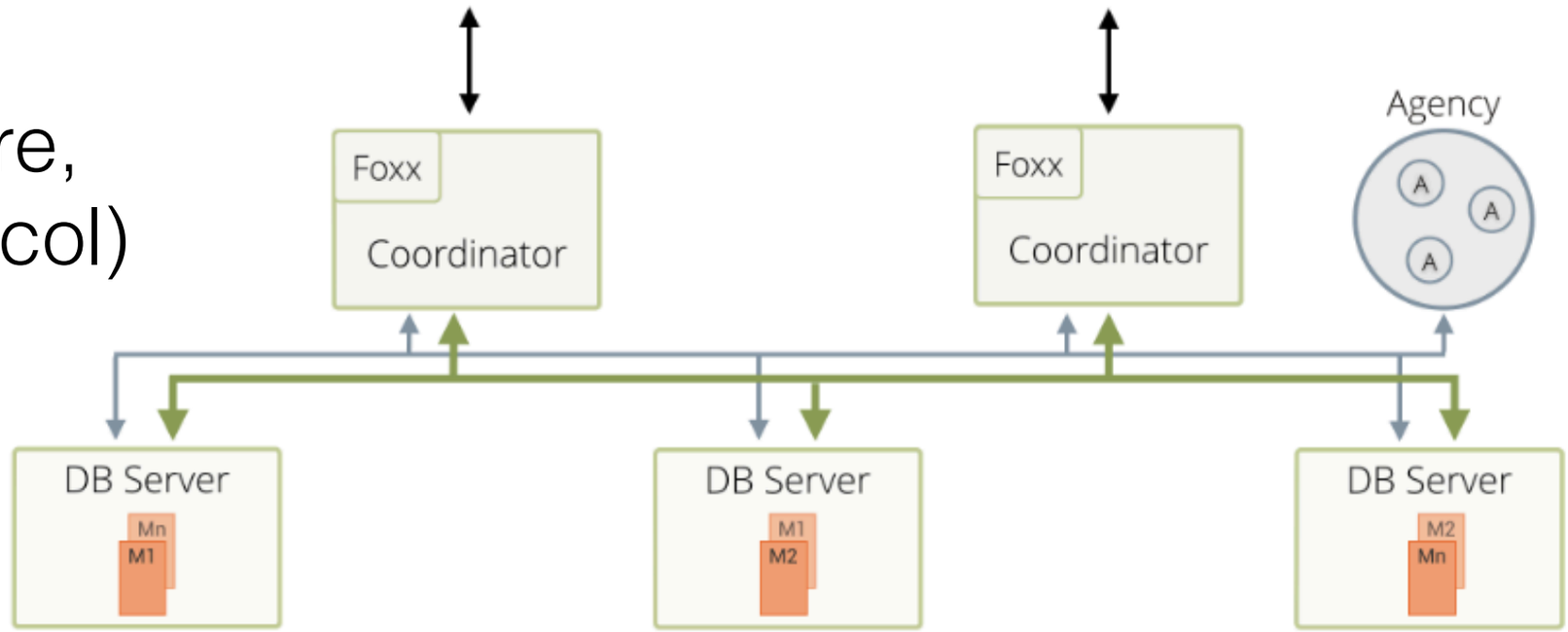
```
UPSERT { name: 'superuser' }
INSERT { name: 'superuser', logins: 1, dateCreated: DATE_NOW() }
UPDATE { logins: OLD.logins + 1 } IN users
```

```
      "user"     : u.name,
      "friendId" : f.thisUser
    }
```

ArangoDB

Custering, replication, sharding

- Agency (high-avail resilient key/value store, Raft Consensus Protocol)
- Coordinators
- Primary DB servers
- Secondaries
- Asynchronous/ synchronous replication with automatic fail-over



Benchmark (against ArangoDB V 2)

	single read	single write	single write sync	aggregation (ad-hoc query)	shortest path	neighbors* distinct 1st+2nd deg	neighbors with profile data	memory usage
ArangoDB	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %	100.00 %
2.7.0 RC2	16.962 sec.	20.530 sec.	91.816 sec.	1.250 sec.	0.061 sec.	0.464 sec.	4.327 sec.	13.142 GB
MongoDB	**86.76 %**	143.11 %	352.83 %	246.08 %		242.26 %	146.32 %	**49.20 %**
3.0.6 - WiredTiger	**14.716**	29.380	323.953	3.077		1.124	6.331	**6.466 GB**
Neo4j	522.31 %		173.02 %	225.37 %	694.41 %	511.43 %	152.59 %	**68.47 %**
2.3 M3	88.593		158.860	2.818	0.422	2.372	6.602	**8.998 GB**
OrientDB	168.55 %	116.96 %		1860.12 %	2188.82 %	1119.4 %	905.72 %	102.06 %
2.2 alpha	28.590	24.011		23.259	1.331	5.192	39.187	13.413 GB
PostgreSQL (json)	111.07 %	106.92 %	**75.12 %**	1401.22 %		461.91 %	**66.21 %**	**85.21 %**
9.4.4 - json	18.840	21.951	**68.972**	17.521		2.142	**2.865**	**11.199 GB**
PostgreSQL (tab.)	267.93 %	153.27 %	**76.87 %**	**48.77 %**		425.08 %	**43.34 %**	**77.40 %**
9.4.4 - tabular	45.445	31.465	**70.581**	**0.61**		1.972	**1.875**	**10.172 GB**

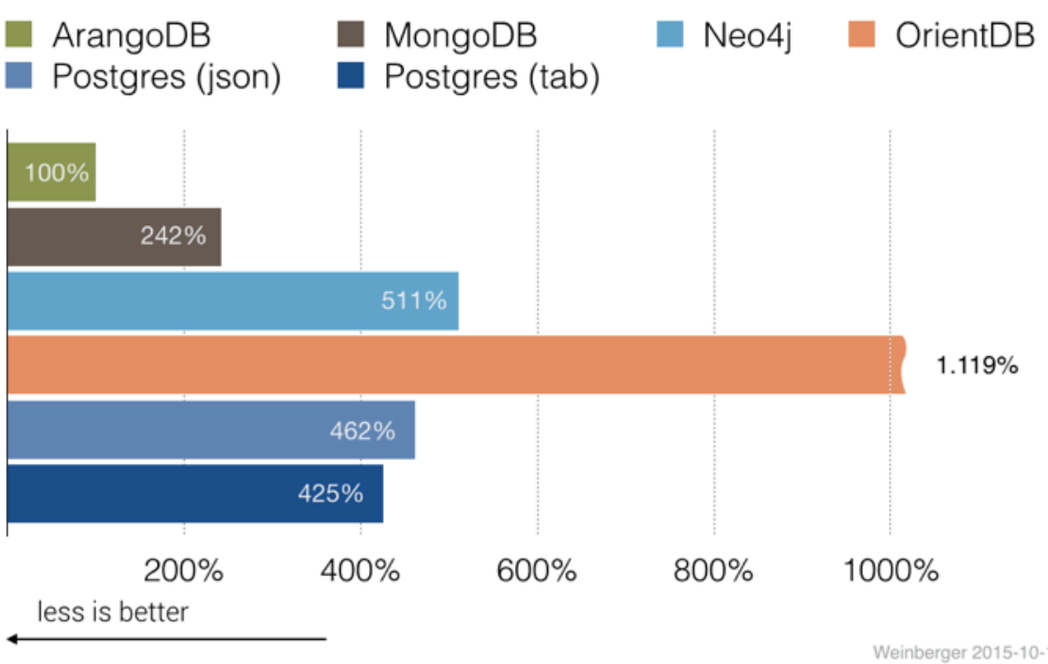
*) distance 1 and the distance 2 neighbors, each of them once.

Weinberger 2015-10-13 (r207)

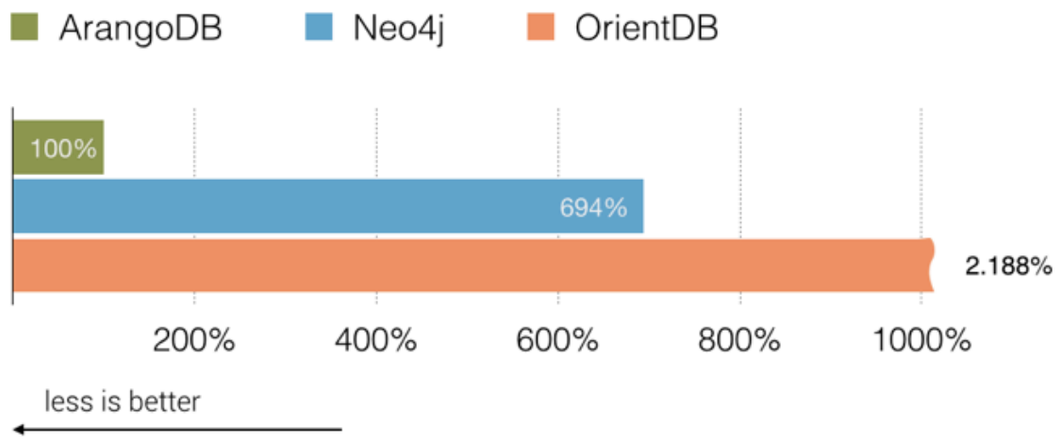
https://www.arangodb.com/2015/10/benchmark-postgresql-mongodb-arangodb/

Benchmark (against ArangoDB V 2)

Neighbour search



Shortest path



https://www.arangodb.com/2015/10/benchmark-postgresql-mongodb-arangodb/

Python bindings (PyArango)

```
from pyArango.connection import *

conn = Connection()
conn.createDatabase(name = "test_db")
db = self.conn["test_db"] #all databases are loaded automatically into the connection and are accessible i
collection = db.createCollection(name = "users") #all collections are also loaded automatically
# collection.delete() # self explanatory

for i in xrange(100) :
      doc = collection.createDocument()
      doc["name"] = "Tesla-%d" % i
      doc["number"] = i
      doc["species"] = "human"
      doc.save()

doc = collection.createDocument()
doc["name"] = "Tesla-101"
doc["number"] = 101
doc["species"] = "human"

doc["name"] = "Simba"
# doc.save() # overwrites the document
doc.patch() # updates the modified field
doc.delete()
```

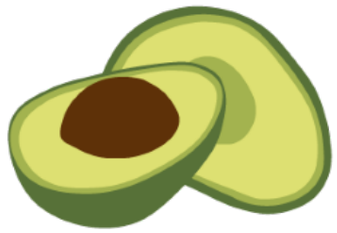
Python bindings (PyArango)

Query (AQL)

```
aql = "FOR c IN users FILTER c.name == @name LIMIT 10 RETURN c"
bindVars = {'name' : 'Tesla-3'}
# by setting rawResults to True you'll get dictionaries instead of Document objects, useful if you want to
queryResult = db.AQLQuery(aql, rawResults = False, batchSize = 1, bindVars = bindVars)
document = queryResult[0]
```

Query (by example)

```
example = {'species' : "human"}
query = collection.fetchByExample(example, batchSize = 20, count = True)
print query.count # print the total number or documents
```

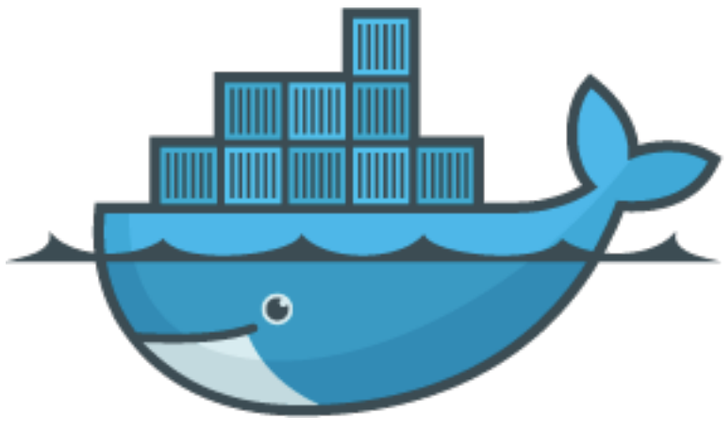
ArangoDB

Graphs

```python
from pyArango.collection import Collection, Field
from pyArango.graph import Graph, EdgeDefinition

class Humans(Collection) :
 _fields = {
 "name" : Field()
 }

class Friend(Edges) :theGraphtheGraph
 _fields = {
 "lifetime" : Field()
 }

#Here's how you define a graph
class MyGraph(Graph) :
 _edgeDefinitions = (EdgeDefinition("Friend", fromCollections = ["Humans"], toCollections = ["Humans"]), )
 _orphanedCollections = []

#create the collections (do this only if they don't already exist in the database)
self.db.createCollection("Humans")
self.db.createCollection("Friend")
#same for the graph
theGraph = self.db.createGraph("MyGraph")

#creating some documents
h1 = theGraph.createVertex('Humans', {"name" : "simba"})
h2 = theGraph.createVertex('Humans', {"name" : "simba2"})

#linking them
theGraph.link('Friend', h1, h2, {"lifetime" : "eternal"})

#deleting one of them along with the edge
theGraph.deleteVertex(h2)
```

Deployment & operations

Apache™
MESOS

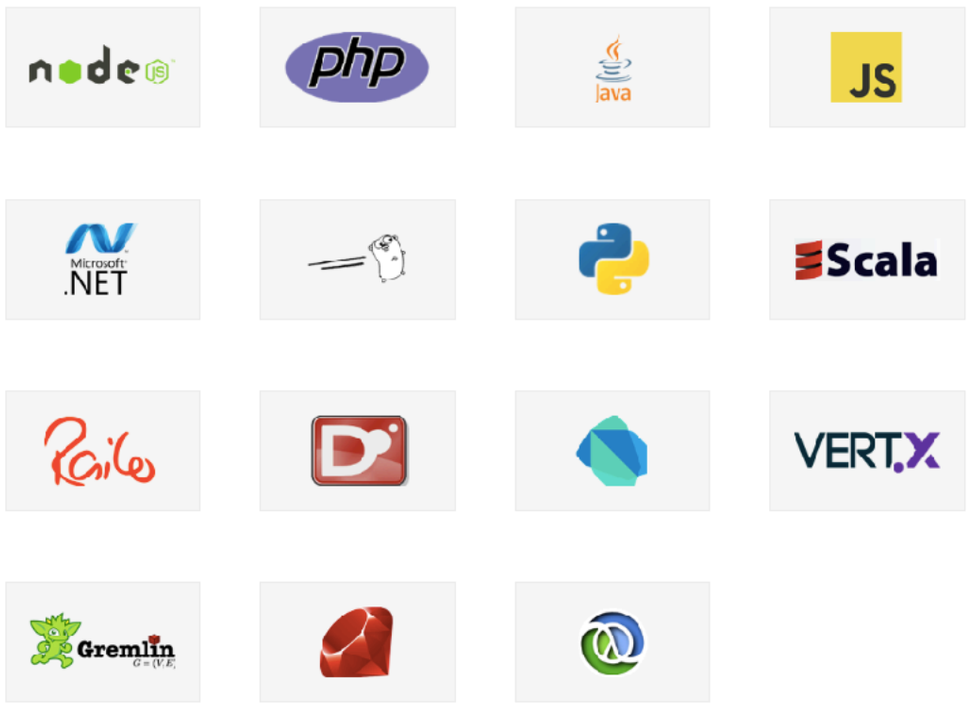


Misc

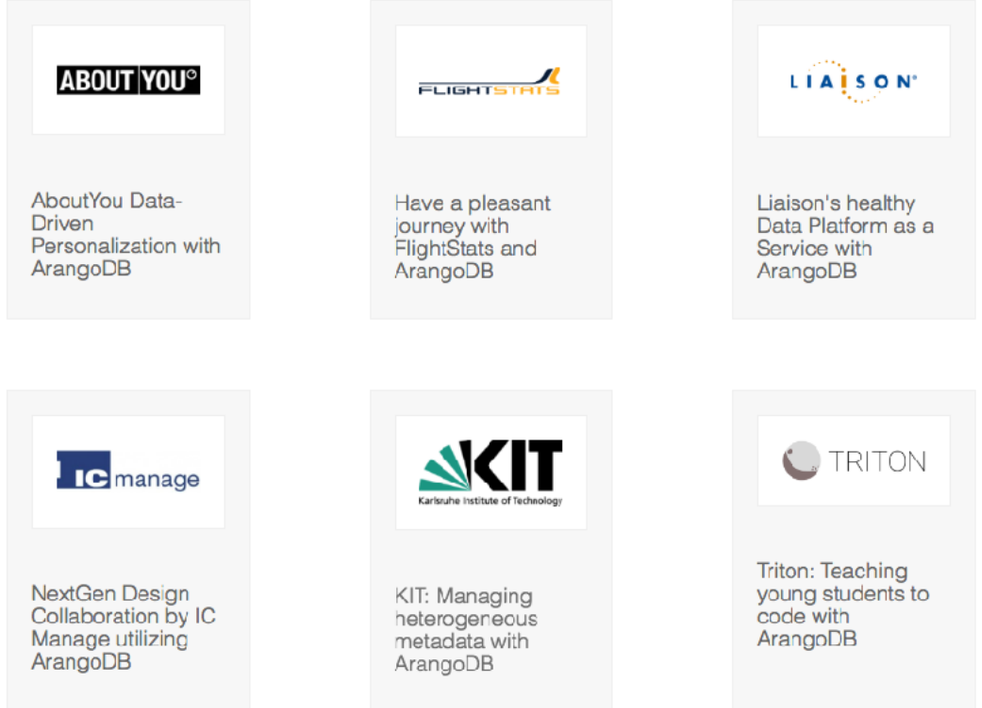
- current version 3.0, (3.1 RC3)
- good documentation
- regular updates and fixes
- nicely supported
- supported by ArangoDB GmbH in Cologne

ArangoDB

- **Community** Edition (Apache License 2.0)
- **Enterprise** Edition (SLA, support options, smart graphs, auditing, better security control)



https://www.arangodb.com/arangodb-drivers/



https://www.arangodb.com/why-arangodb/references/


Questions?
ArangoDB