ArangoDB

Siegen, 31 August 2017

Max Neunhöffer

Documents (JSON)

In this talk, when I say “document“, I mean JSON document:

JSON example

```
{
  "name": "Neunhöffer", "firstName": "Max",
  "address": { "street": "Im Bendchen", "number": "35a",
               "town": "Kerpen", zip: 50169 },
  "height": 1.80, "blabla": null,
  "isHere": true, "isAway": false,
  "children": ["Savina", "Phil"]
}
```

The Multi-Model Approach

Multi-model database

A multi-model database combines a document store with a graph database and is at the same time a key/value store,

The Multi-Model Approach

Multi-model database

A multi-model database combines a document store with a graph database and is at the same time a key/value store,
with a common query language for all three data models.

The Multi-Model Approach

Multi-model database

A multi-model database combines a document store with a graph database and is at the same time a key/value store,
with a common query language for all three data models.

Important:

- is able to compete with specialised products on their turf

The Multi-Model Approach

Multi-model database

A multi-model database combines a document store with a graph database and is at the same time a key/value store,
with a common query language for all three data models.

Important:

- is able to compete with specialised products on their turf
- allows for polyglot persistence using a single database technology

The Multi-Model Approach

Multi-model database

A multi-model database combines a document store with a graph database and is at the same time a key/value store,
with a common query language for all three data models.

Important:

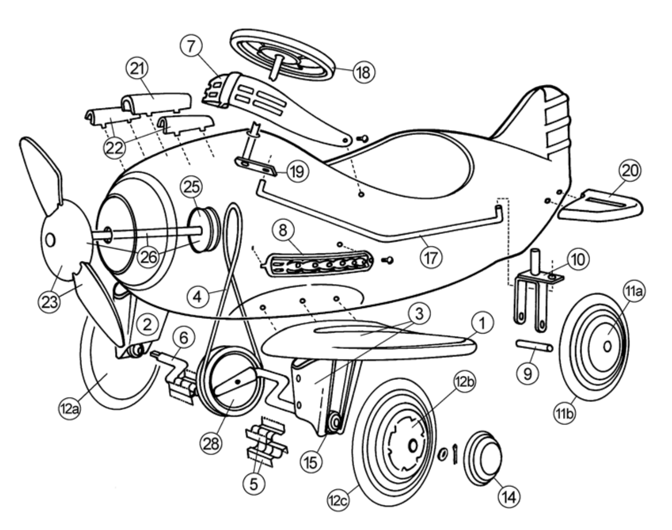
- is able to compete with specialised products on their turf
- allows for polyglot persistence using a single database technology
- In a microservice architecture, there will be several **different** deployments.

Relational database vs. document store

Comparison

Relational database	Document store
table	collection
row	JSON document
schema of columns	schema-free
SQL query	other, JSON-centric languages
standardized	wide variety
data normalization	choice between embedding and normalization
joins	many stores do not offer joins (ArangoDB does!)

Use case: Aircraft fleet management



Use case: Aircraft fleet management

One of our customers uses ArangoDB to

- store each part, component, unit or aircraft as a document
- model containment as a graph
- thus can easily find all parts of some component
- keep track of maintenance intervals
- perform queries orthogonal to the graph structure
- thereby getting good efficiency for all needed queries

http://radar.oreilly.com/2015/07/
data-modeling-with-multi-model-databases.html

Why is multi-model possible at all?

Document stores and key/value stores

Document stores: have primary key, are key/value stores.

Why is multi-model possible at all?

Document stores and key/value stores

Document stores: have primary key, are key/value stores.

Without using secondary indexes, performance is nearly as good as with opaque data instead of JSON.

Why is multi-model possible at all?

Document stores and key/value stores

Document stores: have primary key, are key/value stores.

Without using secondary indexes, performance is nearly as good as with opaque data instead of JSON.

Good horizontal scalability can be achieved for key lookups.

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

Why is multi-model possible at all?

Document stores and graph databases

Graph database: would like to associate arbitrary data with vertices and edges, so JSON documents are a good choice.

Why is multi-model possible at all?

Document stores and graph databases

Graph database: would like to associate arbitrary data with vertices and edges, so JSON documents are a good choice.

- A good edge index, giving fast access to neighbours.
  This can be a secondary index.

Why is multi-model possible at all?

Document stores and graph databases

Graph database: would like to associate arbitrary data with vertices and edges, so JSON documents are a good choice.

- A good edge index, giving fast access to neighbours.
  This can be a secondary index.
- Graph support in the query language.

Why is multi-model possible at all?

Document stores and graph databases

Graph database: would like to associate arbitrary data with vertices and edges, so JSON documents are a good choice.

- A good edge index, giving fast access to neighbours.
  This can be a secondary index.
- Graph support in the query language.
- Implementations of graph algorithms in the DB engine.

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

ArangoDB AQL: Powerful query language

AQL

The built in Arango Query Language allows

- complex, powerful and convenient queries,

ArangoDB AQL: Powerful query language

AQL

The built in Arango Query Language allows

- complex, powerful and convenient queries,
- to mix all three data models in a query,

ArangoDB AQL: Powerful query language

AQL

The built in Arango Query Language allows

- complex, powerful and convenient queries,
- to mix all three data models in a query,
- with transactional semantics,

ArangoDB

AQL: Powerful query language

AQL

The built in Arango Query Language allows

- complex, powerful and convenient queries,
- to mix all three data models in a query,
- with transactional semantics,
- to do joins (like in the relational model),

ArangoDB AQL: Powerful query language

AQL

The built in Arango Query Language allows

- complex, powerful and convenient queries,
- to mix all three data models in a query,
- with transactional semantics,
- to do joins (like in the relational model),
- AQL is independent of the driver used and

AQL: Powerful query language

AQL

The built in Arango Query Language allows

- complex, powerful and convenient queries,
- to mix all three data models in a query,
- with transactional semantics,
- to do joins (like in the relational model),
- AQL is independent of the driver used and
- offers protection against injections by design.

AQL: Powerful query language

```
FOR user IN users
  RETURN user
```

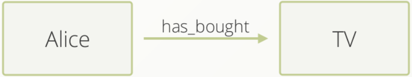
AQL: Powerful query language

```
FOR user IN users
  FILTER user.name == ’alice’
  RETURN user
```

Alice

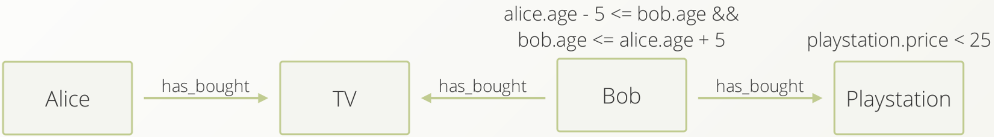
ArangoDB AQL: Powerful query language

```
FOR user IN users
  FILTER user.name == 'alice'
  FOR product IN OUTBOUND user has_bought
    RETURN product
```



ArangoDB AQL: Powerful query language

```
FOR user IN users
  FILTER user.name == ’alice’
  FOR recommendation, action, path IN 3 ANY user has_bought
    FILTER path.vertices[2].age <= user.age + 5
           AND path.vertices[2].age >= user.age - 5
    FILTER recommendation.price < 25
    LIMIT 10
    RETURN recommendation
```



Extensible through JavaScript

The Foxx Microservice Framework

Allows you to extend the HTTP/REST API by **your own routes**, which you implement in JavaScript running on the database server, with direct access to the C++ DB engine.

ArangoDB

Extensible through JavaScript

The Foxx Microservice Framework

Allows you to extend the HTTP/REST API by **your own routes**, which you implement in JavaScript running on the database server, with direct access to the C++ DB engine.

Unprecedented possibilities for data centric services:

- complex queries or authorizations, schema-validation, push feeds, etc.

ArangoDB Extensible through JavaScript

The Foxx Microservice Framework

Allows you to extend the HTTP/REST API by **your own routes**, which you implement in JavaScript running on the database server, with direct access to the C++ DB engine.

Unprecedented possibilities for data centric services:

- complex queries or authorizations, schema-validation, push feeds, etc.
- easy deployment via web interface or REST API,

ArangoDB Extensible through JavaScript

The Foxx Microservice Framework

Allows you to extend the HTTP/REST API by **your own routes**, which you implement in JavaScript running on the database server, with direct access to the C++ DB engine.

Unprecedented possibilities for data centric services:

- complex queries or authorizations, schema-validation, push feeds, etc.
- easy deployment via web interface or REST API,
- automatic API description through **Swagger** $\Longrightarrow$ discoverability of services.

ArangoDB : A distributed, fault-tolerant system

ArangoDB provides (Version 3.2, August 2017)

- Sharding with automatic data distribution,
- easy setup of replication (synchronous and asynchronous),
- fault tolerance by automatic failover,
- self-repairing and self-balancing cluster architecture,
- full integration with Apache Mesos and Mesosphere DCOS,
- easy deployment and scaling on various cloud orchestration tools.

ArangoDB : A distributed, fault-tolerant system

ArangoDB provides (Version 3.2, August 2017)

- Sharding with automatic data distribution,
- easy setup of replication (synchronous and asynchronous),
- fault tolerance by automatic failover,
- self-repairing and self-balancing cluster architecture,
- full integration with Apache Mesos and Mesosphere DCOS,
- easy deployment and scaling on various cloud orchestration tools.

Work in progress (Version 3.3, October 2017):

- asynchronous data center to data center replication,

ArangoDB : A distributed, fault-tolerant system

ArangoDB provides (Version 3.2, August 2017)

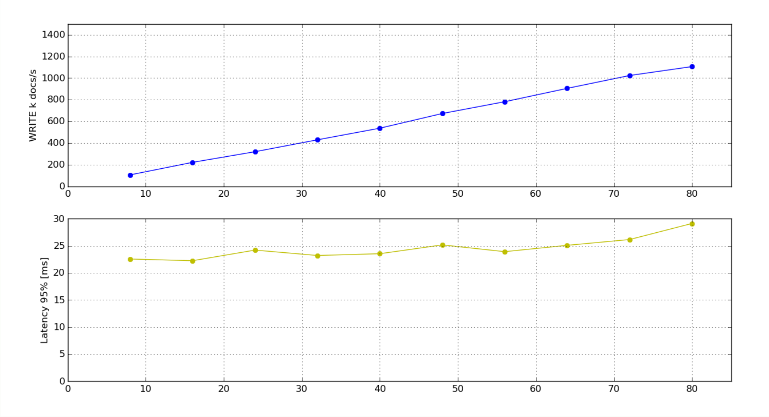
- Sharding with automatic data distribution,
- easy setup of replication (synchronous and asynchronous),
- fault tolerance by automatic failover,
- self-repairing and self-balancing cluster architecture,
- full integration with Apache Mesos and Mesosphere DCOS,
- easy deployment and scaling on various cloud orchestration tools.

Work in progress (Version 3.3, October 2017):

- asynchronous data center to data center replication,
- Distributed transactions.

ArangoDB horizontal scalability

Experiment: Single document writes (1kB / doc) on cluster of sizes 8 to 80 machines (64 to 640 vCPUs), another 4 to 40 load servers, running on AWS.



https://mesosphere.com/blog/2015/11/30/arangodb-benchmark-dcos/

Easy deployment

- Binary packages for various Linux variants, Windows and MacOS
- Docker images
- There is a tool for easy cluster deployment “ArangoDB starter“
- For Apache Mesos and DC/OS there is a framework scheduler
- Cloud orchestration tools like Kubernetes and Docker Swarm are possible

Links

`https://www.arangodb.com`

`https://docs.arangodb.com`

`http://mesos.apache.org/`

`https://mesosphere.com/`

`https://mesosphere.github.io/marathon/`