Revolutionizing Data Management

A Comprehensive Study of FaunaDB's Impact on Modern Database Technologies

DS4300 || Luke Abbatessa, Yuting Zheng

Abstract

This report evaluates FaunaDB, a distributed, multi-model database designed for modern applications. It offers a unique combination of flexibility and structure, making it stand out as a document-relational database. We explore its foundational principles, diverse use cases, and provide a detailed tutorial for practical understanding. Additionally, we assess its utility, scalability potential, licensing, and community support. Our goal is to equip development teams with insights to make informed decisions about integrating FaunaDB into their projects. We aim to highlight its suitability, capabilities, and potential challenges, empowering teams to navigate database selection confidently.

Introduction

FaunaDB offers a distributed document-relational database that blends document flexibility with relational power, ensuring ACID compliance while scaling seamlessly across regions, clouds, or globally [FaunaDB]. It leverages user-defined functions (UDFs) with the Fauna Query Language (FQL), modern security measures, event streaming support, and data import/export capabilities. Developers benefit from declarative queries, multi-region ACID consistency, flexible JSON document imports, and a combination of document storage and entity modeling akin to relational databases.

Under the Fauna Services Agreement, licensing terms are delineated across five subparts: License for Paid Use, Order Form, License for Free Tier, Support and Service Level Agreements, and Updates to Services. The License for Paid Use grants limited access for internal business use, while the Order Form facilitates longer subscriptions. The License for Free Tier offers a trial period, and Support and Service Level Agreements apply to subscription purchases. Updates to Services allow FaunaDB periodic service updates.

Globally, FaunaDB boasts a user base of 80,000+ development teams across 180 countries, collectively creating over 300,000 databases [FaunaDB]. This report will showcase FaunaDB's use cases, provide a tutorial on installation and FQL queries, discuss strengths and weaknesses (especially in comparison to MongoDB), and offer concluding remarks on the effectiveness of the database.

Use-Cases

Software as a Service (SaaS) Company - CloudConnect CRM:

- Use Case 1: Multi-tenancy Management
  - FaunaDB supports multi-tenancy out of the box, enabling efficient hosting of data from multiple tenants within a single database instance while ensuring data privacy and compliance with regulatory requirements.
- Use Case 2: User Data + Metadata Management
  - Fauna's flexible schema and document-relational model facilitate easy management of metadata without schema rigidity constraints, enabling seamless introduction of new data types or relationships and ensuring granular control over user permissions.
- Use Case 3: Global Distribution + Data Compliance
  - Fauna's distributed transaction engine ensures auto-replication across distributed nodes and regions, guaranteeing high availability and low latency access for users worldwide while adhering to data protection and privacy regulations.
- Use Case 4: Microservice Support
  - FaunaDB seamlessly integrates with CloudConnect CRM's microservices architecture, enabling each service to interact with the database via secure HTTP requests and ensuring scalable and responsive performance for users.

Retail and E-Commerce Company - E-Shop Emporium:

- Use Case 1: Storing Comprehensive Product and User Metadata
  - Fauna's document-relational model allows flexible storage and management of detailed product information, user profiles, and transactional data at scale, accommodating evolving e-commerce data structures and access patterns seamlessly.
- Use Case 2: Efficient Order Processing
  - Fauna's distributed transaction engine ensures strong consistency and low-latency transactions globally, enabling E-Shop Emporium to efficiently and reliably process orders, even during peak demand periods, thereby enhancing customer satisfaction and trust.
- Use Case 3: Inventory Management Across Regions
  - Fauna's strong consistency across regions simplifies inventory management for E-Shop Emporium, ensuring accurate stock levels and minimizing the risk of over-selling or stockouts, without the need for extensive engineering operations associated with legacy databases.
- Use Case 4: Personalized Shopping Experience
  - Fauna's API delivery model, coupled with flexible querying and indexing capabilities, enables E-Shop Emporium to build powerful personalization engines, delivering context-aware content, product recommendations, and targeted marketing campaigns with ultra-fast response times to enhance customer engagement and drive sales.

Strengths and Weaknesses

Strengths:

- FaunaDB combines document and relational features for flexibility
- FaunaDB ensures strict serializability, providing strong transactional consistency
- Enables easy scaling for modern applications
- Requires minimal custom building for distribution

Weaknesses:

- No open-source option, limited to a 30-day trial
- Cannot be self-hosted
- Certain operations in FaunaDB may experience latency issues, with "get" and "match" operations sometimes taking longer compared to MongoDB
- Unlike MongoDB Atlas, FaunaDB does not offer built-in real-time data analytics functionality

Tutorial

Create

Let's say you wanted to create a user with the name "William T" and the email "ryker@enterprise.com"; assuming you created a User collection, the FQL code snippet for this that uses the create() function would look like this:

```
User.create(name:  "William T", email:  "ryker@enterprise.com")
```

and the resulting success message would look something like this:

```
>> SUCCESS id:  "360940140394184781", coll:
User, ts:  Time("2023-04-02T23:29:51.680Z"),
name:  "William T", email:  "ryker@enterprise.com",
```

We can see that the user is automatically assigned an id, and the success message also shows the timestamp of the transaction.

Read

Let's say you first create a to-do list with a Todo collection using Create; the list could look something like this:

```
Todo.create(title:  "Get Milk", status:  "Pending")
Todo.create(title:  "Do Laundry", status:  "Pending")
```

and you'd like to simply read your to-do list; the FQL code snippet for this that uses the .all() function would look like this:

```
Todo.all()
```

and the code that would return would look something like this:

```
data:  [id:  "360941936862822477", coll:  Todo, ts:
Time("2023-04-02T23:58:24.920Z"), title:  "Get Milk", status:  "Pending",,
id:  "360942191116288077", coll:  Todo, ts:
Time("2023-04-03T00:02:27.385Z"), title:  "Do Laundry", status:
"Pending",,],
```

Update

Let's say you wanted to update the status of the "Get Milk" task from "Pending" to "Done"; the FQL code snippet for this that uses the update() function would look like this:

```
let todo = Todo.byId("360941936862822477") todo.update(status:  "Done")
```

and the code that would return would look something like this:

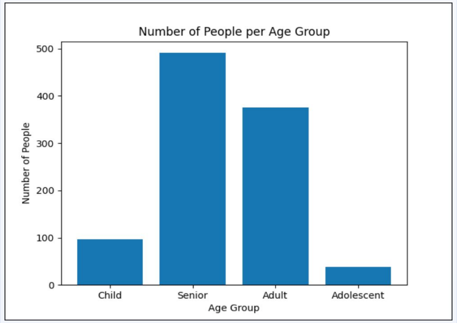
```
id:  "360941936862822477", coll:  Todo, ts:
Time("2023-04-03T03:43:10.010Z"), title:  "Get Milk", status:  "Done",
```

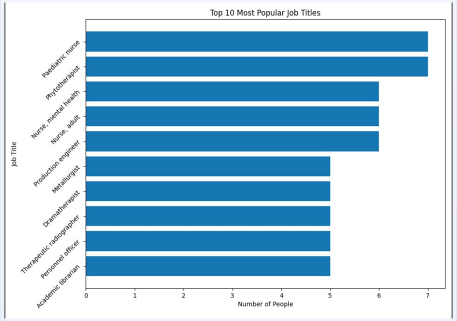
Delete

Let's say you wanted to delete the "Get Milk" task since it's now done; the FQL code snippet for this that uses the delete() function would look like this [FaunaDB]:

```
let todo = Todo.byId("360941936862822477") todo.delete()
```

Tutorial (cont.)





Conclusions

In essence, FaunaDB offers a compelling solution for organizations seeking a flexible, scalable, and globally distributed database for their modern applications. Its innovative features and robust performance make it a strong contender in the ever-evolving landscape of database technologies. By carefully evaluating its strengths and weaknesses, development teams can make informed decisions regarding its adoption, ensuring that FaunaDB aligns with their project requirements and objectives.