How to Build a Backend-as-a-Service
Lessons Learned

Felix Gessert
fg@baqend.com
12.06.2017, Serverless Meetup
@baqendcom
Web Application Needs
Non-Functional Requirements

“Over 30 percent of web development teams deliver projects late or over-budget.”

Survey by New Bamboo
Common Web Architectures
The 3-Tier Stack

Subject to latency

Blocks delivery of website

Must ensure high availability

HTML, CSS, JS, ...

Server

Needs to be scalable

Database

Update

User Action

Requires business logic in the server

Client
Common Web Architectures
The 2-Tier Stack

**Progressive** client-side rendering

**Problem:**
- User Management
- Access Control
- Server-side Business Logic
- **Latency** and **scalability** still an issue

**Database-as-a-service**

**Client-side** business logic

**Client**

**Cloud DB**

**HTML, CSS, JS**

**Update**

**Data**

**Render**

Angular, React, Vue.js

DynamoDB, Elasticache, MongoDB

Still latency-critical

Still latency-critical

Progressive client-side rendering

Latency and scalability still an issue
The Serverless Paradigm

**FaaS and BaaS**

Many different *services* and APIs

**Function-as-a-Service (AWS Lambda)**

- Authentication
- Gateway
- Business Code
- Cloud DB
- Private DB
- Notifications

**Backend-as-a-Service**

Scalable code execution

Mike Roberts: [https://martinfowler.com/articles/serverless.html](https://martinfowler.com/articles/serverless.html)
Backend-as-a-Service Overview

Unified API and docs

Reusable features

Must be fast, scalable and highly available

Firebase

Microsoft Azure

Parse

Baqend
BAAS OVERVIEW

How Backend-as-a-Service works
Backend-as-a-Service

Feature Sets

**API for application features**
- Data Storage
- Real-Time
- Query, Search
- Backend Code
- Users, OAuth
- File Storage
- Access Control

**Hosting and Delivery**

**REST API and JS SDK**

Diagram showing the connection between Backend as a Service and various services and features.
Frontend

GET /app.html
GET /js/main-34da93.js
GET /css/main-9ad7ca3.css
Compatible with:

- AngularJS
- Ember
- React
- Ionic
- Handlebars
- Lodash
- Knockout
- Underscore.js
GET /app.html
GET /js/main-34da93.js
GET /css/main-9ad7ca3.css

db.Page.load('main')
  .done(...);

db.Page.find()
  .descending('published')
  .limit(3)
  .resultList(...);

GET /img/pic005.jpg
GET /img/pic017.jpg
GET /img/pic022.jpg
Comparison of Service Models

**Backend-as-a-Service**
- Fast time-to-market
- Low maintenance
- Often limited in expressiveness, scalability or performance

**Platform-as-a-Service**
- Full control of backend
- Tools for deployment and maintenance
- Limited in platform control

**Infrastructure-as-a-Service**
- High flexibility and full control
- Build software from scratch
- High maintenance overhead
Presentation is loading
The Latency Problem

-20% Traffic
-9% Visitors
-1% Revenue

Average: 9.3s
Performance affects many Business KPIs...what causes slow page loads?
State of the Art

Two Bottlenecks: Latency and Processing

High Latency

Processing Overhead
Network

Bandwidth vs. Latency

Netzwerk
Bandbreite vs. Latenz

![Diagram](image)

- **$2 \times \text{Bandwidth} = \text{Same Load Time}**
- **$\frac{1}{2} \text{Latency} \approx \frac{1}{2} \text{Load Time}**
STATE OF THE ART

How to improve web performance?
How AMP works:

- **Stripped down** HTML + AMP tags (e.g. `img`) → rendered asynchronously by AMP runtime
- **CSS must be inlined** + <50 KB +
- No **custom JS** (except in iframes)
- Only static sizes → **no repaints**
- **Cached** in Google CDN, as long as it is crawled the next time
  → only suited for **static** media, e.g. news

How to apply these techniques for *any website*?

https://www.ampproject.org/docs/reference/spec.html
Solution: Global Caching
Fresh Data From Distributed Web Caches
New Caching Algorithms
Solve Consistency Problem
New Caching Algorithms
Solve Consistency Problem
Typical Speedup: 15x
Impact of Global Caching

TRY THIS
benchmark.baqend.com
THINKS USE CASE

How do these techniques work for a high-traffic shop?
Performance Case Study
Thinks Online-Shop

Expected:
• 3.5 Mio. TV Viewers
• 4 Weeks for Development & Tests
• >300K Visitors

< 1 Second Page Loads

Concurrent Users

7.8% Conversion Rate

Expected:
• 3,5 Mio. TV Viewers
• 4 Weeks for Development & Tests
• >300K Visitors
High Cache Hit Rate: 99%

Logo in TV screen: sudden spike

Peak Load of over 20,000 requests per second

>3.2 Gigabit/s

At high load micro-caching is very effective
How does this work?

Dynamic Caching in Detail

False-Positive Rate: \[ f \approx (1 - e^{-k/m})^k \]

Hash-Functions: \[ k = \left\lfloor \ln(2) \cdot \frac{n}{m} \right\rfloor \]

With 20,000 entries and a 5% false positive rate: **11 Kbyte**

**Consistency**: \( \Delta \)-Atomicity, Read-Your-Writes, Monotonic Reads, Monotonic Writes, Causal Consistency
Network Optimizations

In Baqend

- Early SSL Termination
- OCSP Stapling
- Session Resumption
- Warm Backend-Connections

- Caching at Internet Exchange Points
- Backend Failover & Stale-on-error
- Automatic Scaling
- DDOS Protection
Why HTTP/2?

Typical Improvements

HTTP/2:
- **Multiplexing** over 1 TCP connection (no head-of-line blocking)
- Request **Pipelining**
- Server **Push**
- **Header Compression**

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>3.22s</td>
</tr>
<tr>
<td>HTTPS</td>
<td>4.03s</td>
</tr>
<tr>
<td>HTTP with CDN</td>
<td>0.44s</td>
</tr>
<tr>
<td>HTTPS with CDN and h2</td>
<td>0.35s</td>
</tr>
</tbody>
</table>
SCALING A BAAS

How to achieve scalability?
Backend Performance
Best Practices

- Load Balancing
- Auto-scaling
- Failover

- Stateless Sessions
- Minimize shared state
- Efficient Code & IO

- Horizontally scalable databases (e.g. “NoSQL”)
- Replication
- Sharding
- Failover
Backend Architecture
Baqend Cloud

Inclusion of all Web Caches
Access through HTTP

Data, Queries, User Login, etc.

Backend-as-a-Service API:

Scalable Databases

redis

mongoDB

elasticsearch

Desktop

Mobile

Tablet

Content-Delivery-Network

Internet

Caches

Baqend
Backend Architecture

Baqend Cloud

CDN on

Baqend Cloud on

fastly

Amazon Web Services

Baqend Cloud

Content-Delivery-Network

mongoDB

Internet

Caches

Backend

Public Search
Network & Backend Performance

Overview

- Consistent Browser Caching
- Fast CDN-Invalidations
- HTTP/2 & SSL
- APIs for SPAs

- High Availability
- Automatic Scaling
- DDoS Protection

Other Optimizations
- Scale & Optimize Images
- Minify JS & CSS
- Load Javascript async.
- Use Resource Hints
How do I use Baqend?
Including Baqend
Getting Started

Try this: www.baqend.com/tutorial.html
Development
On Baqend

Dashboard
Create schema, configurations, browse data, etc.

CLI
Develop, deploy and test frontend and backend code

REST & SDK
Website logic: load site, get data, login, etc.
Live Demo
(see: https://github.com/Baqend/thesis)
IN DEPTH

How are the Baqend BaaS APIs used?
Baqend JS SDK
For Web & Hybrid

• Completely **ES6**- and **TypeScript**-compatible
  • Support for Maps, Sets, Arrays
  • Models can be ES6-classes

• **Promise**-based

• Abstracts from **Caching** logic, intelligent **Object-Identity**

• Powerful **Query-Builder**

• Automatic **Change-** and **Dependency-Tracking**
Data Modeling
Combining Schemaful & Schemaless

- **Types**: Boolean, Integer, String, DateTime, Time, Date, GeoPoint, List, Map, Set, JSON-Object, JSON-Array
- **References, embedded types & inheritance**
CRUD
Data APIs

Insert & Delete

```javascript
var event = new DB.Event({id: 123, title: "Party"});
event.insert();
event.delete();
```

Updates

```javascript
event.title = "Grillen";
event.save();
```

Read

```javascript
Db.Todo.load(123);
```

Query

```javascript
DB.Todo.find()
.matches('name', /^My Todo/)  
.equal('active', true)  
.lessThanOrEqualTo('activities.start', new Date())  
.resultList(...)
```
From CRUD to REST/HTTP
Cache Hierarchy

DB.Posts.load GET /db/posts/{id}

Cache-Hit: deliver
Cache-Miss or revalidate request
Return data with caching information (Browser TTL, CDN TTL, ETag, LM)

Guarantee: Data never older than the Bloom filter
Guarantee: 200ms for global invalidations

Updated by Cache Sketch
Updated by server
REST API
Access from all platforms

• Stateless and scalable
• Realized in Java/Jetty
  • \textbf{\~15k RPS} on Commodity Hardware
• Formalized in Swagger Specification
  • Language bindings can be generated
  • Browsing through REST API and its resources
User Management
Login, Registration, OAuth

- Simple **registration** and **login**
  - Via Email validation
  - Via OAuth-Providers
- User can have **roles**
- **Acess rights** on user and role level
- **User-schema** is extensible
- **Sessions**, for returning users, too

```javascript
DB.User.login('john.doe@example.com', 'PW')
  .then(() => {
    // Hey we are logged in again
    console.log(DB.User.me.username);
    // 'john.doe@example.com'
  });
```
# Access Control

**Authorization**

<table>
<thead>
<tr>
<th>Role</th>
<th>Load</th>
<th>Insert</th>
<th>Update</th>
<th>Delete</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✔</td>
</tr>
<tr>
<td>admin (db/Role/1)</td>
<td>✔</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>✔</td>
</tr>
<tr>
<td>node (db/Role/2)</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

User - Token JSON Object

Database

- db.posts.update(...,
  [Allow-Deny-Conditions])

**Access Control**

**Authorization**

- **schema-level ACL**
  - Checked via token

- **object-level ACL**
Backend Code
Trusted Business Logic

- Callable Methods
- For **microservices** and custom **web APIs**
Backend Handler

Hooking Into Updates

- **Handler**: OnUpdate, OnDelete, OnInsert, OnValidate
- Node.js code, same SDK

```javascript
function onInsert(DB, obj) {
  var post = new DB.Post(
    {message: "New Data: " + this.name + "."},
    post.save();
  )
}
```

User → Create Object → Baqend → Database
Problem: Concurrency

Lost Update

\[
doc\text{.counter} = \text{doc}\text{.counter} + 1
\]

Write \text{doc}

Load \text{doc}

\[
doc\text{.counter} = \text{doc}\text{.counter} + 1
\]

Write \text{doc}

Load \text{doc}

http://profi.co/all-the-limits-of-parse/
Concurrency in Baqend

Safe Updates

Optimistic Concurrency Control:

Partial Updates:

Others: push, pop, set, remove, etc.
Concurrency in Baqend
Multi-Object Transactions

- Optimistic **ACID-Transactions** (*the* missing NoSQL-feature)
- Status: soon part of the SDK (currently in the REST-API)
LIVE QUERIES

Going Real-Time
Try it: twoogle.app.baqend.com
**Going Real-Time**

Query Caching & Subscribing

How to detect changes to queries:
„Give me the tweets for Java, ordered by Date by people with more than X followers.“

Real-Time Queries (Websockets)

Create Update Delete

Server

Pub-Sub

Fresh Caches

Fresh Bloom filter

InvaliDB on Storm

How to detect changes to queries:
„Give me the tweets for Java, ordered by Date by people with more than X followers.“

Real-Time Queries (Websockets)

Create Update Delete

Server

Pub-Sub

Fresh Caches

Fresh Bloom filter

InvaliDB on Storm
var query = DB.Tweet.find()
  .matches('text', /my filter/)
  .descending('createdAt')
  .offset(20)
  .limit(10);

query.resultList(result => ...);

query.resultStream(result => ...);
InvaliDB
Filter Queries: Distributed Query Matching

Two-dimensional partitioning:
• by Query
• by Object
→ scales with queries and writes

Implementation:
• Apache Storm & Java
• MongoDB query language
• Pluggable engine
SERVICE WORKERS

How to accelerate legacy systems?
New Standard: Service Workers

What they do:

- **Proxy** any HTTP request
- **Offline Cache** for “progressive web apps”
Service Workers + Baqend Caching

**Baqend Worker:**
- **Redirect** requests to Baqend for faster delivery by including a **snippet**
- **Update** of cached data: refresh of stored data against origin

Public Beta in 3 weeks
commerce.codetalks.de
(Rails)

commerce.codetalks.de
Loaded Through
Baqend Worker
DEMO

In action:
makefast.baqend.com
Lessons Learned
Building a Scalable BaaS

Frontend
- Single-page applications are fast for navigation
- Good Tooling for optimizations (Inlining, Above-the-fold, Minification, etc.)

Network
- Caching in the CDN and browser
- Dynamic data should also be cached
- Minimize latency, SSL and HTTP tuning

Backend
- Horizontal scaling with stateless web servers
- NoSQL databases
- Cloud-hosted
- Load-Tests important
- Failover and autoscaling

fg@baqend.com   @baqendcom
Our Lessons Learned Building a Backend-as-a-Service: A Technical Deep Dive

In this post we will share our technical learnings about building a multi-tenant Backend-as-a-Service (BaaS). We will cover how a BaaS works, how it fits into the serverless world and how performance and scalability can be achieved with tools such as AWS, Docker Swarm, MongoDB, Redis, Varnish and CDNs.

At Baqend, we are building a Backend-as-a-Service (BaaS) service that is geared towards scalability and web performance. Coming from research, we try to be open about the architecture of our platform and therefore would love to share how everything works at a technical level. This post is a combined writeup of different talks we gave.
Further Reading

Resources

Web Performance in a Nutshell: HTTP/2, CDNs and Browser Caching

High Performance Website Hosting with SSL and HTTP/2 Made Simple

The AWS and MongoDB Infrastructure of Parse: Lessons Learned

NoSQL Databases: a Survey and Decision Guidance

1. Start a free app
2. Check out the starter kits and tutorials
3. Develop web application
>10x Faster Loads

Automatic Scaling

Faster Development

For a web without loading times.

www.baqend.com
@Baqendcom