Our Lessons Learned Building a Backend-as-a-Service

A Technical Deep Dive

Felix Gessert
fg@baqend.com
09.05.2017, Netlight
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

Render

Data

Update

Client

Server

Database

HTML, CSS, JS, ...

User Action

Requires business logic in the server

Needs to be scalable

Must ensure high availability

Subject to latency

Updates
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

Client

Render

Still latency-critical

Database-as-a-service

Client-side business logic

Angular
React
Vue.js

HTML, CSS, JS

Data

Update

Cloud DB

DynamoDB
Elasticache
MongoDB
Common Web Architectures

The 2-Tier Stack

Business Logic

Browser

HTML, CSS, JS

Data

Update

Cloud APIs (e.g. comments)

Static Website Files

Angular  React  Vue.js

DynamoDB  Elasticache  MongoDB

MongoDB

Common Web Architectures

The 2-Tier Stack

Business Logic

Browser

HTML, CSS, JS

Data

Update

Cloud APIs (e.g. comments)

Static Website Files

Angular  React  Vue.js

DynamoDB  Elasticache  MongoDB

MongoDB

Common Web Architectures

The 2-Tier Stack

Business Logic

Browser

HTML, CSS, JS

Data

Update

Cloud APIs (e.g. comments)

Static Website Files

Angular  React  Vue.js

DynamoDB  Elasticache  MongoDB

MongoDB
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

Scalable code execution

Backend-as-a-Service
Backend-as-a-Service

Overview

Unified API and docs

Must be fast, scalable and highly available

Reusable features

Backend-as-a-Service

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
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
Backend-as-a-Service
Comparison of Service Models

- **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

- 100 ms
- 400 ms
- 500 ms

Average: 9.3s

-20% Traffic
-9% Besucher
-1% Umsatz
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

Page Load Time as bandwidth increases

Page Load Time as latency decreases

Netzwerk
Bandbreite vs. Latenz

\[ 2 \times \text{Bandwidth} = \text{Same Load Time} \]

\[ \frac{1}{2} \text{Latency} \approx \frac{1}{2} \text{Load Time} \]
STATE OF THE ART

How to improve web performance?
What Others Do: AMP
Google’s Accelerated Mobile Pages

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


F. Gessert, N. Ritter „Scalable Data Management: NoSQL Data Stores in Research and Practice”, 32nd IEEE International Conference on Data Engineering, ICDE, 2016

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?
Shops in "Die Höhle der Löwen"

The Google Page Speed Scores for Season 3, 09/06/2016

- **STRYVE by THINKS**: 99
- **Tepping**: 68
- **REISHUNGER**: 62
- **penta sense**: Down
- **Gemütliche Top**: Down
- **Chickefen**: Down

< 1 Second Page Loads

Concurrent Users

7.8% Conversion Rate

3% Server Usage

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

At high load micro-caching is very effective

>3.2 Gigabit/s
How does this work?
Dynamic Caching in Detail

Has **Time-to-Live** (expiration)

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

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

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

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

\[ f \approx (1 - e^{-kn/m})^k \]

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

HTTP
- 3.22s

HTTPS with CDN
- 4.03s

HTTP with CDN
- 0.44s

HTTPS with CDN and h2
- 0.35s
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
Data, Queries, User Login, etc.

Backend-as-a-Service API:
Access through all Web Caches

Scalable Databases

Desktop
Mobile
Tablet

Content-Delivery-Network

Baqend

Caches

redis
mongoDB
elasticsearch
Backend Architecture

Baqend Cloud

CDN on

Baqend Cloud on

Content-Delivery-Network

Backend Architecture

Baqend Cloud

CDN on

Baqend Cloud on

Content-Delivery-Network
Swarm Manager Nimbus Zookeeper

Management and Orchestration Server

Updated Query

Storm Query Matching

Updates

Pub/Sub

Updated Query

Pub/Sub
Overview
Optimizations

- 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](http://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.
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 Modelling
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();
```

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 data
Cache-Miss or revalidation: forward 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
REST API
Access from all platforms

• Stateless and scalable
• Realized in Java/Jetty
  • ~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**
  - PUT `db/posts/{id}`
  - **User-Token**
    - JSON Object

- **Baqend**
  - `db.posts.update(..., [Allow-Deny-Conditions])`
  - **schema-level ACL**
    - Checked via token

- **Database**
  - **object-level ACL**
Backend Code

Trusted Business Logic

- Callable Methods
- For microservices and custom web APIs
• **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();
}
```
Problem: Concurrency

Lost Update

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

Lost Update

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

Safe Updates

Optimistic Concurrency Control:

```
doc, Etag: 22
```

```
doc
If-Match: 22
```

Version Check ➔ Atomic Update/Rejection

Partial Updates:

Others: push, pop, set, remove, etc.

```
doc.inc(‘counter‘,1)
```

commutative

```
doc.inc(‘counter‘,1)
```
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
Going Real-Time
Query Caching & Subscribing

How to detect changes to queries:
“Give me the most popular products that are in stock.”

Create
Update
Delete

Server

Pub-Sub

Fresh Caches

Fresh Bloom filter

InvaliDB on Storm

Real-Time

Pub-Sub
<table>
<thead>
<tr>
<th>Rank</th>
<th>Username/Screen Name (Followers)</th>
<th>Tweet</th>
<th>URL</th>
</tr>
</thead>
</table>
| 1    | Conju (conju_re, 3840 followers) | Congress Saved the Science Budget—And That's the Problem https://t.co/UdJNldakc  
https://t.co/xlNjEpKZG | [Link](https://twitter.com/conju_re/status/859767327570702336) |
| 2    | Yuuu_key (229 followers) | けいさんと PENGUIN RESEARCHのけいたんがリブのやり取りしてる... | [Link](https://twitter.com/Yuuu_key/status/859767323384623104) |
| 3    | Whitney Shackley (bschneids11, 5 followers) | holy... waiting for it so long 🙌  
https://t.co/UdXcHJb7X3 | [Link](https://twitter.com/bschneids11/status/85976731953469122) |
| 4    | Lisa Schmid (LisaMSchmid, 67 followers) | Congrats to Matthew Kent, winner of the 26th TeamSCS Coding Challenge.  
https://t.co/vXtO6WgJrZ #SCCodingChallenge | [Link](https://twitter.com/LisaMSchmid/status/859767317311500290) |
| 5    | Brian Larson (Brian_Larson, 40 followers) | Congrats to Matthew Kent, winner of the 26th TeamSCS Coding Challenge.  
https://t.co/Brian_Larson/status/859767317303001089 | [Link](https://twitter.com/Brian_Larson/status/859767317303001089) |
var query = DB.Tweet.find()
  .matches('text', /my filter/)
  .descending('createdAt')
  .offset(20)
  .limit(10);

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

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

Programming Real-Time Queries
JavaScript API
Baqend Streaming Queries
Real-Time Architecture

Keeps data up-to-date
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

Email: fg@baqend.com  Twitter: @baqendcom
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

Baqend Blog
On Building a Faster Web

https://medium.baqend.com/

Baqend

https://www.baqend.com/
>10x Faster Loads

Automatic Scaling

Faster Development

For a web without loading times.

www.baqend.com
@Baqendcom