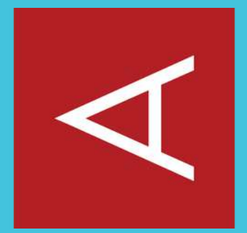
BDNR

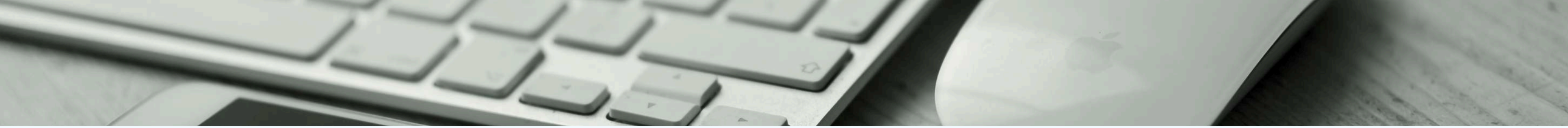
AEROSPIKE

Presented by Group 03



SUMMARY OF CONTENTS

OUR MAIN TOPICS TODAY

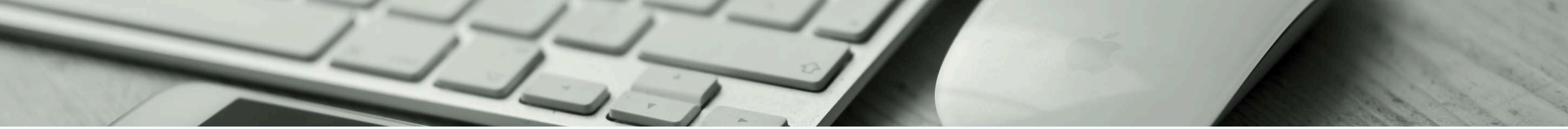


AEROSPIKE

WHY WAS AEROSPIKE CREATED?

- **High-Performance NoSQL** Database
- Designed to allow **Zero Downtime**
- Built for **Real-Time Applications**
- **Key-Value** main data model for simplicity and scalability (supports other models)

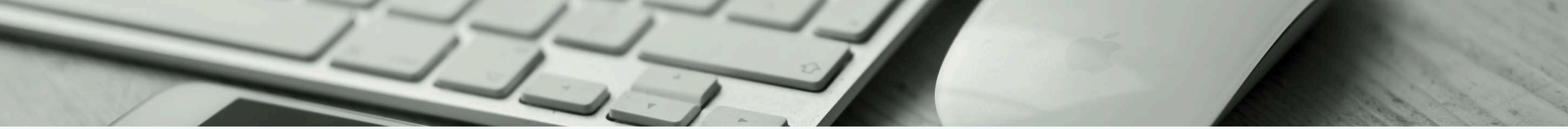
...this led us directly to our application's theme



DISTRIBUTED CHAT

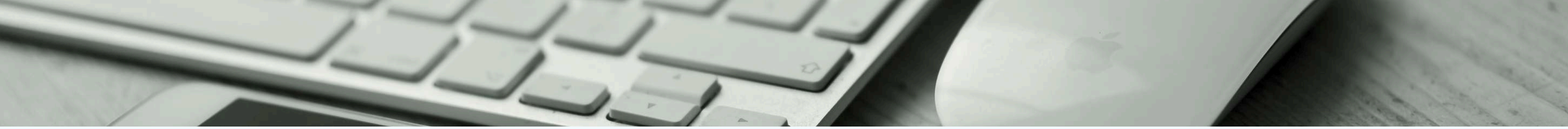
PROPOSED PROJECT THEME

- **High Concurrency:** Supports Thousands of Concurrent Users
- **Scalability:** Scales Seamlessly with Growing User Base
- **Fault Tolerance:** Ensures Continuous Availability of Chat Service



ADMINISTRATION

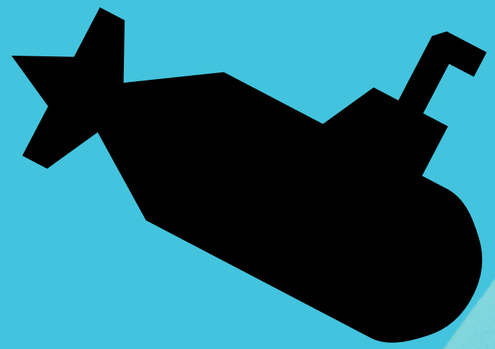
- Several CLIs available for various purposes
  - **asadm**: cluster management (allows creating Secondary Indices, which are explained further ahead, among other things).
  - **aql**, providing a minimal DML/DDL to query and modify the database's contents.
  - **asinfo**, provides information on the operational status of a node
- There seems to be an administration web dashboard but we could not find it anywhere on the official website



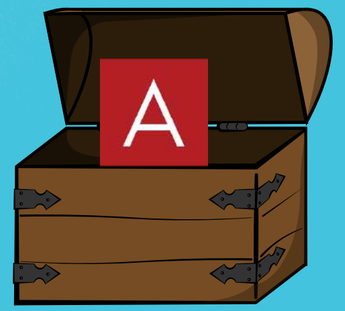
INSTALLATION

- Optimized for 64-bit Linux
- Using Docker
- RPM/Deb Packages
- Package and Source builds
- Cluster: Docker Swarm

NOW THAT THE RECAP IS DONE



LET'S DIVE **DEEPER...**



Data Model

- **Multi-Modal;**
- Predominantly **Key-Value**, supporting **Document** and **Graph** operations (the latter needing a separate service);
- **Records** identified by a **Primary Key** and having data separated into **bins**;
- **Schema-less**: the schema is derived from the usage.

DATA OPERATIONS

- Record retrieval using the PK;
- Bin querying with support of **Secondary Indices**;
- Support for GeoJSON-based queries;
- **Map** and **List** operations for a bin;
- **Record-level transactions**;

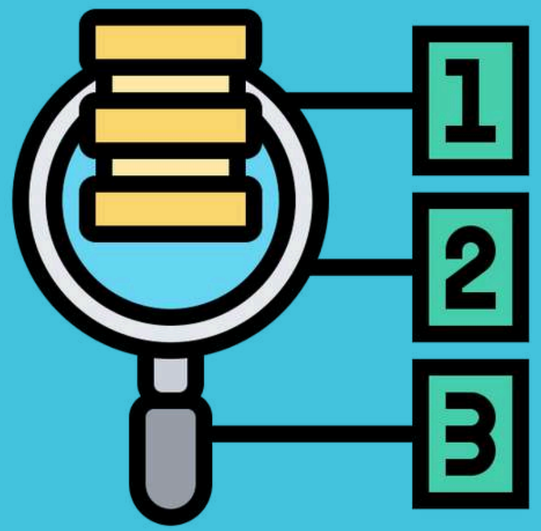
OTHER IMPORTANT FEATURES

- **Hybrid Memory Architecture** giving users the ability to configure the best storage solution for their specific needs.
- **Cross-Datacenter Replication** provides reliable and available storage on a global scale.
- Ability to run custom, user-defined Lua scripts (**User Defined Functions**) on a single record or on a Stream of records.

CLIENT LIBRARIES

- Developed in several languages, chief among them are C, Java, **Javascript** and Python.
- **Smart Clients** which have an extensive view into a cluster's topology and can intelligently route client requests to the appropriate server nodes.
- Library specific support for certain operations (e.g., JSON Document API is only available for the Java library)

PROTOTYPE OVERVIEW



DATA GENERATION

- Generated with the use of **Faker.js**, a tool that allows for generation of fake data fitting into customizable data schemas.

```
const messages = [];
const keywords = {};
for (let i = 0; i <= 450; i++) {

    const message = {

        PK: `message${i + 1}`,
        senderId: users[i % 20].PK,
        senderName: users[i % 20].PK,
        senderImage: users[i % 20].image,
        content: faker.lorem.sentence({ min: 3, max: 8 }),
        timestamp: faker.date.recent().getTime(),
        deleted: faker.datatype.boolean(0.1)

    };

    messages.push(message);

    const curr_keywords = message.content.toLowerCase().split(' ');
    for (const keyword of curr_keywords) {

        if (keywords.hasOwnProperty(keyword)) {
            keywords[keyword].push(message.PK);
        } else {
            keywords[keyword] = [message.PK];
        }

    }

}
```

Figure 1: Messages/Keywords Generation

```
const channels = [];
let offset = 0;
for (let i = 0; i < 30; i++) {

    const serverMessages = [];
    for (let j = 0; j < 8; j++) {
        serverMessages.push(messages[offset + j]);
    }
    offset += 15;

    const channel = {
        PK: `channel${i + 1}`,
        name: faker.animal.type(),
        server: `server${i % 20 + 1}`,
        messages: serverMessages.map(message => {
            return message.PK;
        }),
        members: serverMessages.map(message => {
            return [message.senderId, message.senderName, message.senderImage];
        })
    };

    channel.messages = '["' + channel.messages.join('","') + '"]'
    channels.push(channel);

}
```

Figure 2: Channel Generation

AGGREGATES

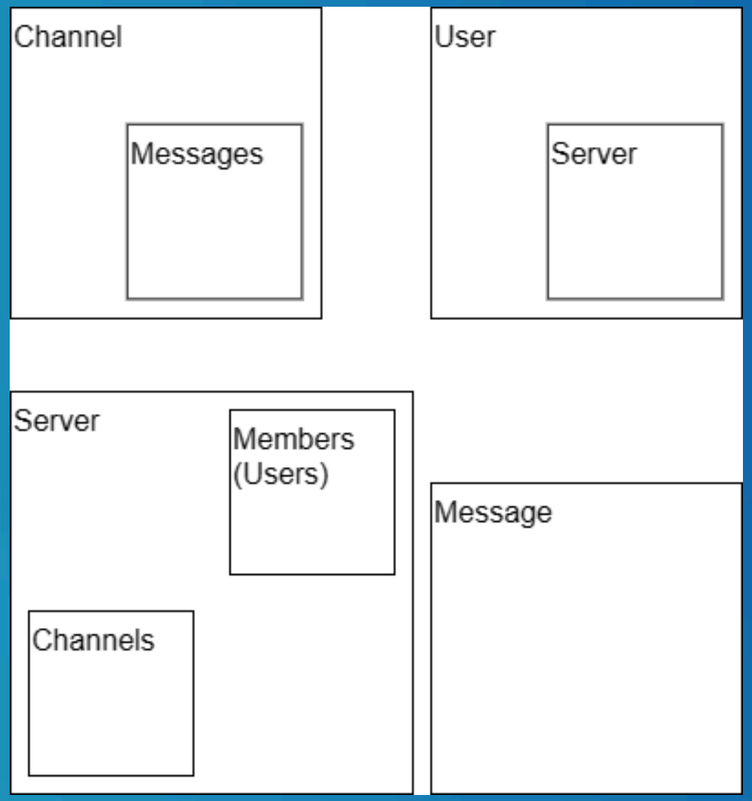


Figure 3: application aggregates

CONCEPTUAL MODEL

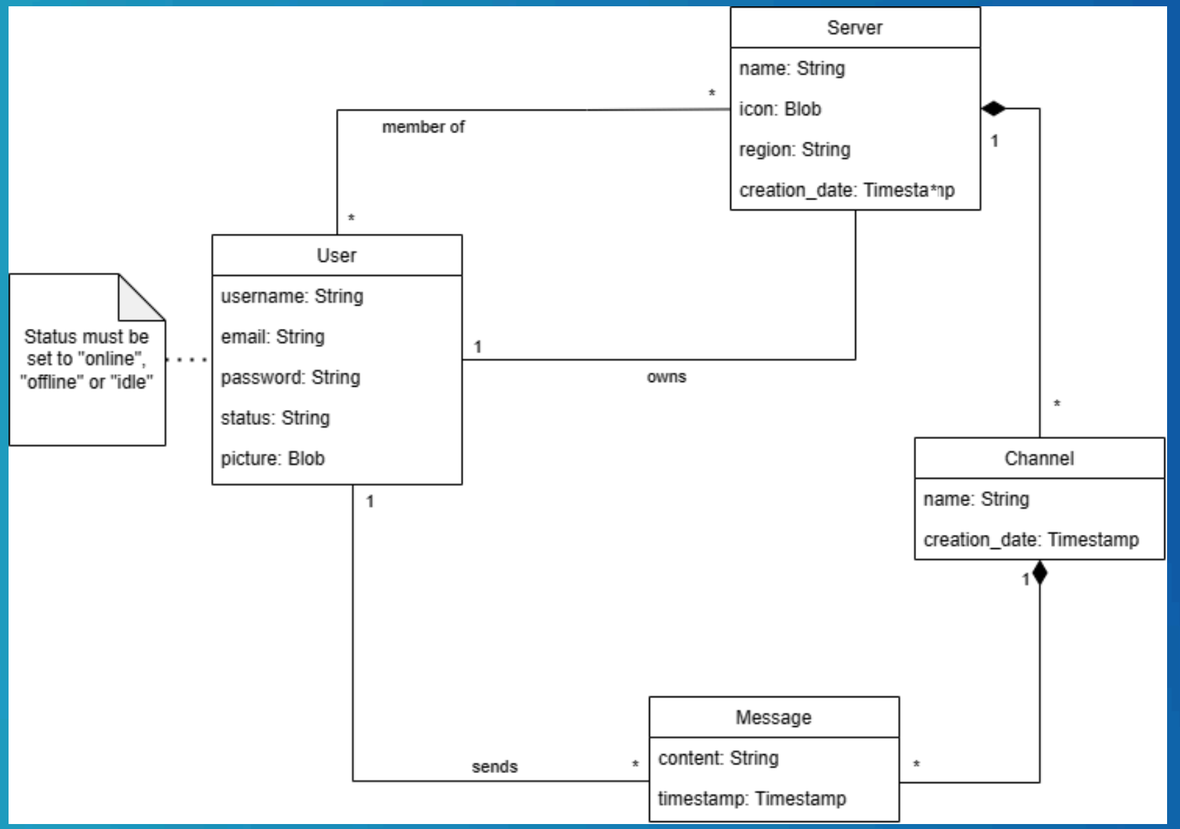


Figure 4: conceptual model

PROTOTYPE FEATURES

- Account Creation/Deletion
- Server Creation/Deletion
- Channel Creation/Deletion
- Message Deletion
- Status Change (online/offline)
- Message Searching w/ Advanced Filtering
- User Search
- Server Invitations
- Server join/leave
- User Statistics
- (...)

QUESTIONS? COMMENTS?

LET US KNOW!