Firebase

Burgstaller
Mobile Computing
Fachhochschule Hagenberg
Hagenberg, Austria
email address

Schneeweiß
Mobile Computing
Fachhochschule Hagenberg
Hagenberg, Austria
S1710237029@students.fh-hagenberg.at

***Abstract*—There are many different frameworks available to create a web application. One of them is Node Js which has exploded in popularity in recent years [15] [9]. Another option is using a service, which allows the developer to not touch any server side code, for example Database as a Service (DaaS). To study and compare a server side framework to DaaS, we used benchmarks for performance metrics and scalability. Representatives are Node Js for web application frameworks and Firebase Realtime Database for DaaS. The goal is to visualize the advantages and disadvantages for each.**

***Index Terms*—Database as a Service, web application, Node Js, Firebase**

I. Introduction

The current web and mobile development forces developers to react to new problems, such as multiuser and high concurrency requests. One popular client side scripting language is JavaScript. With Node Js it was possible to create JavaScript code outside of the browser environment and therefore open it up to server side development. Node Js is based on the V8 engine developed by Google and allows for fast scalable web applications [5]. Node Js uses an event-driven, non-blocking I/O which makes it perfect for scalable web applications which support high concurrency [15].

Another option to overcome this is by outsourcing your network and backend to a cloud provider, by using services to get the desired functionality. This allows the customer to be certain the functionality is always given (over 99% [1]) and it is only billed for the usage.

The aim of this paper is to compare both options in different categories and show the advantages and disadvantages for each, and which usecases fit them best.

II. Cloud Database

Cloud computing and cloud hosted databases outsources the management of hardware and software resources to the network. This allows the customer to treat it as a utility, which provides him with massive advantages [20]

- Elasticity of resources
- Pay-as-you-go pricing model
- Unlimited ressources
- Inifite scaling

Similar to locally hosted database systems, cloud hosted database systems offer many different types. The three generic terms are platform storage service (NoSQL systems), relational database as a service (DaaS) and virtualized database servers [20] [16].

1) Platform Storage Service: NoSQL systems are designed for high throughput and high availability. This is only possible by giving up features relational database systems adhere to. Mainly simpler query level, no standardized query language and do not fulfill ACID properties [20]. They provide BASE properties, which includes for example eventual consistency, meaning a weaker consistency management protocol [20]. Firebase Realtime Database and Firebase Cloud Firestore are both NoSQL database systems [3], though there are also systems which can be hosted locally, for example MongoDB.

2) Relational Database as a Service: A third party service provides a relational database and manages all maintenance, replication and upgrade options for their users [20]. This frees the customer from buying a server, networking and applying patches to the database software. Two offerings are Amazon Web Services Relational Database System, which opens up MySQL, PostgreSQL and others to the customer. Additionally there are replication, snapshot and performance upgrade available, each costing an additional amount of money. The other is Microsoft SQL Azure [16]. By choosing this approach the customer can use all the benefits of relational databases and SQL, like complex queries, joins, ACID properties. While still having the path to scale and replication as a simple option.

3) Virtualized Database Server [20]: This approach takes an application which has been designed for use in a data center, and puts it on a virtual server in the public cloud. This is beneficial, because the data base system is now able to control the physical resources. The downside with this is, it is necessary to monitor the system usage and acting accordingly.

A. Goals

As said by Sakr [18]. Cloud databases should achieve at least this 4 goals to be successful.

- Availability
  They must always be online, even in the case of a network failure or a any other event. This forces the provider to have nodes across different data centers and redundancy.
- Scalability
  They must support high amount of requests, while having low latency. Additionally they must be able to automatically replicate and redistribute data across newly added nodes.

- Elasticity
  They must scale quickly in input as well as output direction.
- Performance
  The consumer pays only for what they actually use. Therefore the provider reduces costs by keeping their system efficient.

All this goals create unprecedented challenges to the developer and provider of cloud hosted database systems.

B. Challenges [18]

Database systems are critical to most applications, for this reason they face a lot of challenges to ensure the goals mentioned before.

1) True Elasticity: Customers expect cloud hosted services to quickly scale up and down to unpredictable workloads. This creates a challenge for the database system and provider as internet scaled applications can have a high amount of users with a high variance of requests. The demand for services occurs depending on time, day and time of year, as well as popularity of the application. Providers overcome this challenge by having a pay-as-you-go pricing model and on-demand allocation of server resources. Therefore cloud database systems should either allow the customer to allocate more or less resources to their application or have the service dynamically scale up or down, to keep the service responsive.

2) Data Replication: The goal of data replication is to increase the availability, scalability and performance of a system. For database systems this is more complex as database should hold consistent data. The CAP theorem allows database systems to only have 2 of its properties. Most cloud database systems provide data replication by loosening up the consistency guarantee [18].

- Performance
  Usually the goal is to improve the read performance or the write performance [10]. A common technique to scale linearly read performance with more nodes is the master-slave pattern. As long as the master is able to handle all writes, each slave offers an additional node to read from [10].
- Availability
  To increase availability it is necessary to decrease downtime. This can be done with Hot-Standby, where a slave is in sync with the master and on failure of the master, the slave takes over preferably no downtime [10]. Another option is WAN replication, in this system the replication nodes are asynchronously updated and off site. This reduces the risk against cluster wide disasters [10].

3) Live Migration [18]: It increases the efficiency and therefore decreases the operation costs of the provider, because it is the process of moving applications between physical machines. This allows the provider to allocate resources to the application it is needed. The two main benefits are

- Performance by moving low demand applications to a low performance system.
- Availability by moving applications from systems which are scheduled for maintenance.

Their are two main techniques to migrate [18]:

- Stop and Copy
  This is the simplest form. It works by stopping the database on the source system. Taking a snapshot and moving the snapshot to the destination system. Starting up the database on the destination system. The problem with this approach is the downtime of the database scales linearly with the size of the database.
- Iterative State Replication
  In this approach the source database creates a checkpoint, which is copied to the destination database. The source database keeps serving requests and creating checkpoints for the destination to copy. When the difference between source and destination is small enough, a stop and copy is made and the destination database takes over. This comes at the cost of higher computational resources, as both systems are running for the longer migration process.

III. Reasons to choose Cloud Based Services

This section is going to compare cloud based services with self hosted ones. The focus lies hereby on Firebase, cloud based services, and Node Js with Express as the self hosted server.

A. Node Js

Node Js, or Node, opens up Javascript for server side development. Its engine based on the V8 engine, which is the runtime of Google. In contrast to the V8 engine, which supports Javascript in the browser, Node is oriented towards long running sessions [19]. Node on the contrary to most other development languages does not need multithreading for concurrent execution, because it is running a so called asynchronous event loop. Most web servers other than Node based ones, are making great use of multiprocessing. This allows for true parallelism, each thread core executing a different thread. This increases performance by a lot [19]. On single core processors on the other hand, the processor must make context switches. For example while writing to a TCP socket, the processor is going to switch to another thread to keep himself busy [19]. When working with multithreaded applications, there are a lot of things which can go wrong. There are deadlocks, resource sharing, race conditions, all in all it is a rather complex task [19].

An event based programming system on the other hand works based on events. An application registers interest in a specific event and when this event happens the operating system notifies the application, allowing the application to execute the desired code [19]. An important part of an event based system is asynchronous I/O [11]. Using the same example from above, the application is writing to a socket and the underlying socket buffer is filled. If the I/O is synchronous, the whole application would be frozen and not able to respond to any events. On the other hand if the I/O is asynchronous the applications registers the buffer is full and can not be

filled further, this allows the application to execute other tasks while waiting for the event to fill the buffer further [19]. Like multithreaded programs implementing an asynchronous event-driven application has its downsides too. For example not every communication can be tied into an event system [19].

In Node all I/O operations are asynchronous [11]. Excluding a few operations like renaming a file. This means the control is immediately returned to the caller and therefore not blocking the application [19]. Therefore the flow of the application is determined by the events in which the application is interested in, each event invokes a handler. This handlers never blocks the application on I/O or network requests [13].

A web server implemented in Node is capable of serving many users all while running single threaded. This is possible because the main loop only registers to the desired events and never does I/O or business logic processing [19]. The I/O or network request events trigger the actual processing where only a few bytes are written, and as soon as an I/O or network request has to be made, the app asynchronously executes it and immediately returns the control. Therefore, even though a Node application runs single threaded the event-loop is never blocked and is capable of serving many users, all while waiting for I/O [19].

B. Firebase

Firebase offers many different services for mobile apps, web apps and more programming environments [12]. Firebase works especially well for mobile developers, who do not want to create their own backend. Some of these services are

- Realtime Database
  This is the most mature database service offered by Firebase. It is a NoSQL key-value store, which pushes updates on the server side down to the clients. This push process happens really fast [3].
- Cloud Firestore
  This is the second database offering by Firebase. In contrast to the Realtime Database, Cloud Firestore implements a NoSQL document store and allows for more complex queries [3].
- Authentication
  Firebase authentication service allows developers to offer either email and password, or other common authentication processes, like Google and Facebook, for signing in. Developers can then use the premade UI elements or make their own and use the Firebase API for authentication. Either way all users are visible in the Firebase project homepage, excluding the password which is hashed [3].
- Cloud Functions
  Cloud Functions makes Firebase really competitive against custom server solutions. This service allows developers to create functions written in TypeScript, which are triggered by either other Firebase services or HTTP requests [3]. For this reason, it is possible to use all Firebase services, all while offering custom functionality.

Firebase offers three different pricing plans.

- Spark Plan
  Is the free plan and good starting point, as it is always possible to upgrade later. Offers almost all functionality of Firebase at a limited amount. For example Realtime Database offers 1GB stored with 100 simultaneous connections. Firebase themselve quote them as ”Generous limits for hobbyists” [4].
- Flame Plan
  Is the fixed pricing plan, costing $25/month. Increases the limitation up from the Spark Plan, without providing more functionality. The storage limit for Realtime Database goes up to 10GB now, with 100k simultaneous connections [4].
- Blaze Plan
  This is the pay-as-you-go pricing plan, while including the free usage from the Spark Plan. Offers nearly unlimited scaling and all available functionality from Firebase. As pay-as-you-go suggests, the customer is only billed for actual resource usage and not for a fixed amount, in contrast to the Flame Plan [4]. This allows customers to be more flexible and help them against the high variance in load, mentioned before. Realtime Database for example costs $5 per GB/month of storage.

C. Comparison

1) Performance: Omar Almootassem et al. [8] compared Firebase to other NoSQL database system by creating a cloud based service to let users run the test suite and find the database most suited for their application. This cloud based service is built using Node Js. The comparison has two different simulated files, a large one with 200kb and a small one with 5kb. They run each test 30 times to create a sustainable result.

	MongoDB	DynamoDB	Firebase	CouchDB
Upload Small Data	250	210	70	470
Upload Large Data	1200	680	500	2800
Retrieve Small Data	160	150	55	366
Retrieve Large Data	740	300	540	700
Update Small Data	250	210	40	520
Update Large Data	1280	680	380	2800

TABLE I
AVERAGE TIME IN MS FOR EACH TEST

As seen in the table above I Firebase was the fastest in every test. In their [8] test Firebase was the only contender to directly connect to the client, rather than going through the server. For this reason Firebase performed as good as it did. Additionally Firebase remains the connection open while the

application is running, therefore it saves time by avoiding the connection phase.

2) Scalability: Robert Ryan McCune [17] performed several tests comparing scalability of Node Js to Apache and EventMachine. The methodology used is testing the ability to handle an incresing amount of concurrent and total requests. This test environment simulates the web of the time of his writing well, as this stresses high concurrency with low data throughput. In his tests Node Js outperformed both Apache and EventMachine. The performance delta increases with higher load on the server and more cores. Because Node Js is single threaded, to scale it across many cores a cluster is needed [6]. This works by instantiating the application on any amount of workers, each running in its own process. Additionally all share the same port.

Firebase on the other hand is a cloud based system, where the consumer does not know the underlying data structure, but expects good response times and always on time. Firebase themselves advertise their limits on the Realtime Database, consisting of one single database, of up to 100000 responses per second. Additionally they support up to 100000 simultaneously connected user [3]. The Blaze pricing plan allows customers to create more Realtime Databases and increase the scalability by sharding between them.

3) Advantages and Disadvantages: Firebase does offer a lot of benefits by providing an easy to API which allows the developer to synchronize their client data to the Firebase database service [14]. Additionally, Firebase has SDKs for Android, iOS, JavaScript and other applications [3]. This allows Firebase to work well on across platforms, because each client always gets the same data no matter the application they are on.

The customer does not know any server side programming language to create and use a powerful backend. Neither does he have to know about server side deployment, autoscaling and many more considerations that come with the use of a custom backend.

All this comes with the cost of being locked in the Firebase environment and their pricing structure.

Node Js on the other hand, allows the developer to create a backend and choose every little detail about it. This forces the developer to have knowledge in a lot more things, than just client development. Deploying on the server, making sure all dependencies are installed, setting the server up for long running sessions, creating a backup solution.

4) Use Cases: Firebase advertises for a lot of different use cases. Ranging from easy sign up, sign in all the way to machine learning to drive user retention [2]. Firebase tries to create services for every possible functionality of the application, all while keeping it simple and easy-to-use. Firebase shines when it comes to developers who want features and functionality fast and simple.

Node Js on the other hand offers third party packages (over 1 million [7]). Writing code is still necessary and it is not as easy as it is in Firebase to add functionality to your application. Node Js is great for developers, who like to create backends and server side code. But not so great for mobile developers, which do not want to spend money or time on creating a backend.

REFERENCES

[1] Firebase service level agreement. https://firebase.google.com/terms/service-level-agreement. Accessed: 2019-06-30.

[2] Firebase usecases. https://firebase.google.com/use-cases. Accessed: 2019-07-01.

[3] Google firebase documentation. https://firebase.google.com/docs. Accessed: 2019-06-25.

[4] Google firebase pricing. https://firebase.google.com/pricing. Accessed: 2019-06-26.

[5] Node js. https://nodejs.org/en/. Accessed: 2019-06-30.

[6] Node js cluster. https://nodejs.org/api/cluster.html. Accessed: 2019-06-26.

[7] Npm module counts. http://www.modulecounts.com/. Accessed: 2019-07-01.

[8] Omar Almootassem, Syed Hamza Husain, Denesh Parthipan, and Qusay H Mahmoud. A cloud-based service for real-time performance evaluation of nosql databases. *arXiv preprint arXiv:1705.08317*, 2017.

[9] Hudson Borges, Marco Tulio Valente, Andre Hora, and Jailton Coelho. On the popularity of github applications: A preliminary note. *arXiv preprint arXiv:1507.00604*, 2015.

[10] Emmanuel Cecchet, George Candea, and Anastasia Ailamaki. Middleware-based database replication: the gaps between theory and practice. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 739–752. ACM, 2008.

[11] Ioannis K Chaniotis, Kyriakos-Ioannis D Kyriakou, and Nikolaos D Tselikas. Is node. js a viable option for building modern web applications? a performance evaluation study. *Computing*, 97(10):1023–1044, 2015.

[12] Fu Cheng and Fu Cheng. *Build Mobile Apps with Ionic 4 and Firebase*. Springer, 2018.

[13] Fernando Doglio. *Pro REST API Development with Node. js*. Apress, 2015.

[14] Chunnu Khawas and Pritam Shah. Application of firebase in android app development-a study. *International Journal of Computer Applications*, 975:8887, 2018.

[15] Kai Lei, Yining Ma, and Zhi Tan. Performance comparison and evaluation of web development technologies in php, python, and node. js. In *2014 IEEE 17th international conference on computational science and engineering*, pages 661–668. IEEE, 2014.

[16] Vladimir Mateljan, D Cisic, and D Ogrizovic. Cloud database-as-a-service (daas)-roi. In *The 33rd International Convention MIPRO*, pages 1185–1188. IEEE, 2010.

[17] Robert Ryan McCune. Node. js paradigms and benchmarks. *Striegel, Grad Os F*, 11:86, 2011.

[18] Sherif Sakr. Cloud-hosted databases: technologies, challenges and opportunities. *Cluster Computing*, 17(2):487–502, 2014.

[19] Stefan Tilkov and Steve Vinoski. Node. js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, 14(6):80–83, 2010.

[20] Liang Zhao, Sherif Sakr, and Anna Liu. A framework for consumer-centric sla management of cloud-hosted databases. *IEEE Transactions on Services Computing*, 8(4):534–549, 2013.