Banco de dados em tempo real: Firebase

Fábio da Silva Takaki[1], **Lucas Martins Valladares Ribeiro**[1]

[1]Faculdade de Ciências e Tecnologia
Universidade Estadual Paulista
"Júlio de Mesquita Filho"
Caixa Postal 19060-900 – Presidente Prudente – SP – Brasil

fabio@takaki.me, lucasmbtos@live.com

Abstract. *With the emergence of the digital society and the Internet of Things, several concepts for data storage and manipulation have emerged with the purpose of reducing resource and maintenance costs of which Cloud Computing stands out. This article proposes a case study of a tool that addresses one of the service models of the Cloud Computing (IaaS, PaaS and SaaS) architecture, which is Firebase, a tool created by Google. Firebase is a mobile platform that provides a number of services for the rapid development of high-quality web and mobile applications, from which stand out the real-time database service that will be the focus of this work.*

Resumo. *Com o surgimento da sociedade digital e a Internet das Coisas, diversos conceitos para armazenamento e manipulação dos dados surgiram com a finalidade de diminiuir os custos de recursos e manutenção do qual destaca-se o Cloud Computing. Este artigo propõe um estudo de caso de uma ferramenta que aborda um dos modelos de serviço da arquitetura Cloud Computing (IaaS, PaaS e SaaS) que é o Firebase, uma ferramenta criada pela Google. O Firebase é uma plataforma mobile que fornece diversos serviços para o rápido desenvolvimento de aplicativos web e móveis de alta qualidade, do qual destaca-se o serviço de banco de dados em tempo real que será o foco principal deste trabalho.*

1. Introdução

A popularização da internet gerou uma sociedade digital, o que antigamente era local, tornou-se online. O volume de informações se propagou de forma abundante nos últimos anos, principalmente com o avanço das tecnologias móveis. Em decorrência disso, diversos conceitos para armazenamento e manipulação dos dados surgiram com a finalidade de diminuir os custos de recursos e manutenção. Com efeito, originaram-se recursos como [Azure 2017] [Engine 2017] [Services 2017], os quais seguem conceitos de um modelo novo, o *Cloud Computing*. Até então, não existiam serviços que supriam as necessidades essenciais de voltadas para aplicativos atuais (chats e redes sociais), que demandam desenvolvimento móvel/web com armazenamento e sincronização de dados em tempo real. O presente artigo descreve um estudo de caso do *Firebase*, que tem por finalidade fornecer um serviço de dados em nuvem em tempo real, utilizado para criação de aplicativos multiplataformas que compartilham um banco de dados não relacional que a própria ferramenta disponibiliza, do qual será o foco principal deste trabalho. Como consequência, a ferramenta possibilita aos desenvolvedores criarem aplicativos de alta qualidade de forma

rápida e fácil, sem a necessidade de se preocupar com infraestrutura e escalabilidade de recursos.
Este artigo está organizado da seguinte maneira: na Seção II será apresentado uma fundamentação teórica sobre *Cloud Computing*, em que é essencial para o entendimento do Firebase; na Seção III será apresentado a metodologia de apoio; na Seção IV um estudo de caso da ferramenta firebase em que é realizado um estudo do desenvolvimento de uma aplicação; por fim na Seção V, resultados e trabalhos futuros.

2. *Cloud Computing*

Cloud computing é uma tecnologia em que os recursos de *hardware* e *software*, tais como aplicações especiais, CPU, armazenamento e muitos outros são fornecidos aos usuários através da Internet como um serviço, e é cobrado com base no que você usa [Bokhari et al. 2016]. Como os recursos são fornecidos através da Internet, o usuário não se preocupa com a manutenção e suporte deles. Esses recursos têm a capacidade de escalabilidade automática de acordo com a demanda do cliente.

O conceito de *Cloud Computing* é classificado em três modelos de serviço: *Software as a Service* (SaaS), *Plataform as a Service* (PaaS) e *Infrastructure as a Service* (IaaS). Com isso, é possível criar uma representação dos modelos de serviço por meio de camadas de virtualização (Figura 1).

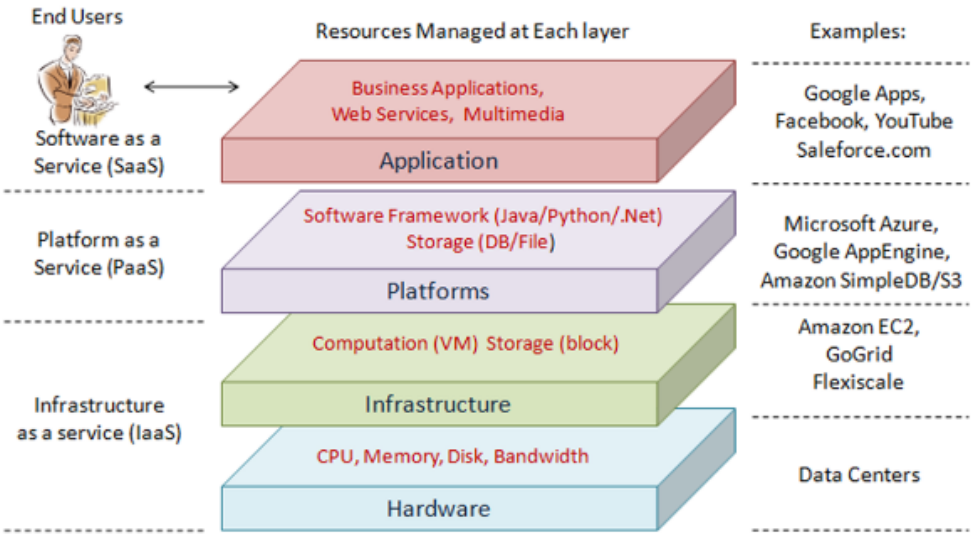

Figure 1. Arquitetura *Cloud Computing* (retirado de: [Zhang et al. 2010])

Infrastructure as a Service

O serviço oferece, como o próprio nome já diz, a infraestrutura. Compondo a base das camadas de virtualização, é responsável pelo *hardware*. Ao invés de manter hacks, servidores e roteadores, o usuário contrata sevidores virtuais, que armazenarão de forma comum á aqueles. Os tipos de cobrança mais comuns são: número de servidores e o tráfego de dados utilizado.

Platform as a Service

Plataforma como serviço, tradução literal, é o serviço entre *Infrastructure as a Service* e *Software as a Service*. É proporcionar um ambiente ou plataforma apropriada na qual o desenvolvedor pode criar as aplicações e o *software*, através da Internet, sem necessidade de instalação ou gerenciar o ambiente de desenvolvimento [Bokhari et al. 2016]. O serviço fornece ao usuário um ambiente flexível e adequado que suporta as tecnologias que deseja, como frameworks ou linguagens de programação. Como observado na figura 1, temos o modelo de serviço *Google App Engine*, o qual oferece uma plataforma de desenvolvimento para aplicações web e móvel.

Software as a Service

Entrando na última camada de serviços, observamos o *Software as a Service*. Esse modelo propõe um software como serviço fornecido através da internet, não necessitando de sua instalação. O fornecedor deste modelo de serviço é responsável por controlar e limitar o uso das aplicações. Portanto, o cliente deste modelo de serviço também fica livre da necessidade de uma infraestrutura, já disponibilizada pelo fornecedor.

Assim, com os conceitos de *Cloud Computing*, o *Firebase* é considerado *Platform as a Service*, por ser uma plataforma de desenvolvimento que oferece armazenamento e sincronização de dados em tempo real, utilizando apenas código *client-side*.

3. Metodologia de Apoio ao *Firebase*

Firebase foi criado em 2011 por Andrew Lee e James Tamplin porém foi lançado oficialmente em 2012. Originalmente, o sistema tinha o intuito de fornecer um banco de dados em tempo real, fornecendo uma API aos usuários para armazenar e sincronizar dados através de diferentes clientes [Cleveroad 2016]. Assim, a metodologia do *Firebase* mudou um pouco após a aquisição da Google. Dois anos depois de sua *release*, surgiram diversos serviços que complementam a ideia do banco de dados em tempo real, voltados para aplicativos móveis ou web: *Firebase Auth*, *Firebase Storage*, *Firebase Cloud Messaging*, *Firebase Remote Config*, *Firebase Test Lab for Android* e *Firebase Crash Reporting*. Dentre todos os serviços citados nesta seção, este trabalho foca no objetivo inicial do *Firebase* que é o *real-time database*.

3.1. Acesso e criação de projetos

O acesso à todos os serviços (citados na Seção 3) do *Firebase*, é preciso apenas de uma conta Google, que pode ser criada gratuitamente. Concedido o acesso ao console do *Firebase*, facilmente você cria um projeto apenas especificando o nome do projeto, e assim é liberado o acesso à todos os serviços. Para o uso da API, necessita apenas do desenvolvedor configurar a aplicação web ou mobile desejada com as *keys* da API do *Firebase* que é fornecida logo após a criação de um projeto, carregando a biblioteca *firebase*.

3.2. *Firebase Realtime Database*

O *Firebase Realtime Database* é um banco de dados hospedado em nuvem. Os dados são armazenados como JSON e sincronizados em tempo real para cada cliente conectado. Ao criar aplicativos multiplataforma com nossos SDKs para iOS, Android e JavaScript, todos os seus clientes compartilham uma instância do *Realtime Database* e recebem automaticamente atualizações com os dados mais recentes [Google 2017]. Portanto, o objetivo

do *Firebase Realtime Database* é fornecer um banco de dados em tempo real, em que o cliente é conectado diretamente e só é preciso enviar os dados através de uma API.

Como os dados são armazenados

Todos os dados são armazenados na nuvem do *Firebase* em objetos JSON formando uma árvore. Quando um dado é adicionado no banco de dados, o dado se torna em um nó na árvore JSON do qual o *Firebase* associa uma chave à esse nó para manipulações futuras. Nesse sentido, um exemplo de um chat em que o usuário tem seu próprio perfil com sua lista de contatos, a árvore JSON se tornaria algo como na Figura 2.

Figure 2. Exemplo de uma estrutura JSON de uma aplicação chat.

Segurança e Regras

O *Firebase Realtime Database* tem uma área (*Firebase Realtime Database Rules*) onde há configurações e regras que determinam quem tem acesso a leitura e gravação no banco de dados, como os dados estão estruturados e o qual índices existem. Todas as regras ficam armazenadas nessa área do *Firebase* e podem ser alteradas a qualquer momento na ferramenta. Portanto, todas as operações de leitura e gravação só serão concluídas se as regras armazenadas no *Firebase Realtime Database Rules* permitirem.

A estrutura em que as regras são definidas também são armazenadas em JSON que seguem uma sintaxe e são definidas em quatro tipos:

1. ***.read*** - Descreve se e quando os dados podem ser lidos pelos usuários.
2. ***.write*** - Descreve se e quando os dados podem ser escritos.
3. ***.validate*** - Define o aspecto correto de um valor formatado, se ele tem atributos filho e o tipo de dados.
4. ***.indexOn*** - Especifica uma criança a indexar para suportar pedidos e consultas.

O *Firebase Realtime Database* é pré-configurado com o *Firebase Auth*, que é responsável por realizar a segurança do app quando sua natureza são dados restritos aos usuários autenticados. Como o intuito deste trabalho é realizar um estudo de caso do serviço de banco de dados em tempo real, foi configurado as permissões de leitura e gravação como público, não necessitando de autenticação para efetuar ambas operações.

API do *Firebase*

A API do *Firebase* é manipulada através de bibliotecas desenvolvidas pela Google em que é facilmente carregada em projetos Android, iOS ou Web. Assim, o *Firebase* conta com uma variedade de métodos em sua biblioteca para a realização de qualquer tipo de operação direta com o banco de dados, com a sua principal característica que são operações em tempo real.

3.3. Implementação de um chat com *Firebase*

Para a realização do estudo de caso para o *Firebase*, foi implementado uma aplicação chat, que são aplicações muito comuns hoje em dia e necessitam de uma sincronização e armazenamento em tempo real. Nesse sentido, foi criado um projeto novo no *Firebase* utilizando a biblioteca Javascript voltado para aplicações web e mobile (híbridas) realizando as operações CRUD de um chat público. Com isso, demonstraremos como seria a mesma aplicação implementada com bancos de dados tradicionais relacionais ou não relacionais.

Enviando Mensagens (*CREATE*)

O envio de mensagens com o *Firebase* pode ser realizado através do método ***.push(objeto)*** da biblioteca do *Firebase*, em que o objeto com as informações são passadas como parâmetro. No caso do chat utilizamos uma estrutura bastante simples do qual o objeto JSON tem apenas o nome do usuário que está enviando a mensagem e a mensagem em si. Em bancos de dados tradicionais, é preciso instalar o banco de dados e utilizar de uma linguagem *backend* para conexão com o banco de dados para a inserção dos dados no banco. Se caso for um banco de dados relacional, é preciso definir a estrutura da mensagem em uma tabela com os campos de nome de usuário e a mensagem.

Recebendo Mensagens (*READ*)

Com o propósito do *Firebase* de realizar o banco de dados em tempo real, o recebimento das mensagens de uma aplicação em chat fica extremamente simples. Com o método ***.on(evento, callback)***, a aplicação interage através de eventos passados parâmetro. Assim, com o evento *child_added* é disparado quando algum novo objeto da referência que está estanciada é adicionado no banco de dados. Então, a função *callback* é chamada, em que é passada no segundo parâmetro do método **.on()**, tem seu parâmetro que tem um valor do objeto JSON adicionado. Então, é através dessa função de *callback*, é possível atualizar a lista de mensagens com a nova mensagem adicionada no banco à todos os usuários que estiverem com a aplicação de chat aberta. A implementação com bancos de dados tradicionais nessa parte do chat, é preciso implementar um *socket* entre todos os usuários que estão conectados na aplicação, que após a inserção de mensagens no banco, os usuários conectados receberiam o aviso através do socket e adicionaria a mensagem na lista de mensagens. Um *socket* é um ponto final de um link de comunicação bidirecional entre dois programas em execução na rede [Oracle 2017].

Editando Mensagens (*UPDATE*)

A edição de mensagens com o *Firebase* é implementado com o mesmo propósito do recebimento de mensagens, do qual utilizaremos do método ***.on()***, porém com o evento *child_changed*, do qual o *callback* só é disparado quando uma mensagem for alterada em algum momento da aplicação. Além disso, para alterar mensagem, é utilizado a função ***.update(objeto)*** a qual é chamada com a referência ao nó na árvore JSON que a atualização é desejada, passando como parâmetro o objeto JSON com os campos desejados e seus novos valores. Então é realizado a atualização no banco de dados *Firebase*. Em bancos de dados tradicionais a edição além da implementação do *socket* para verificar alteração de qualquer mensagem no banco, é implementado também na linguagem *backend* escolhida para realizar a operação *update* para atualização do banco de dados.

Deletando Mensagens (*DELETE*)

A função delete com o *Firebase*, é também implementado com o mesmo propósito do recebimento de mensagens, do qual utilizaremos o método ***.on()***, porém com o evento *child_removed*, do qual o *callback* só é disparado quando uma mensagem for deletada em algum momento da aplicação. Para a remoção de qualquer nó no banco de dados *Firebase*, é preciso apenas chamar o método ***.remove()*** em um objeto referenciado na biblioteca Firebase. Em bancos de dados tradicionais, além da implementação do *socket* para verificar a remoção de qualquer mensagem no banco, é implementado também na linguagem *backend* escolhida para realizar a operação *delete*.

4. Estudo de Caso

Nesta seção é realizado um estudo de caso com os conceitos da ferramenta até então apresentada, no qual é analisado as funcionalidades e facilidades com a ferramenta *Firebase Real Time Database* para a criação de um chat público em tempo real trazendo uma analogia com banco de dados tradicionais relacionais e não relacionais.

A grande questão é que se utilizassemos um banco de dados tradicional, seja ele relacional ou não relacional, para a criação de um chat, seria necessário a utilização de tecnologias como *sockets* para a implementação da sincronização de dados ao ter alguma alteração nas operações de banco de dados, o que trás um certo atraso. Portanto, o chat com os bancos tradicionais, preocuparia o desenvolvedor lidar com a sincronização das mensagens e que podem ser perdidas caso a conexão caia. Além disso, o desenvolvedor teria que se preocupar com a instalação dos bancos de dados para a manipulação e armazenamento das mensagens, do qual irá utilizar dos recursos do servidor onde a aplicação estaria hospedada. Seguindo a linha de raciocínio com os bancos de dados tradicionais, com a preocupação e dificuldade da sincronização dos dados, se o desenvolvedor não souber realizar a implementação correta, pode acontecer do *socket* mandar uma mensagem avisando todos os usuários que foi deletado ou alterado alguma mensagem sendo que no banco de dados ocorreu alguma falha durante a operação, tornando a aplicação inconsistente.

Um outro caso relevante que deve ser explicitado: pelo *Firebase* ser uma multiplataforma de desenvolvimento, facilmente os desenvolvedores conseguem adicionar novas plataformas compartilhando um mesmo banco de dados. No caso do chat, como foi exemplificado com a biblioteca Javascript web, se houvesse uma necessidade de ter uma aplicação Android, facilmente seria implementada com a biblioteca desenvolvida em Java para a manipulação do banco de dados *Firebase*. Em bancos de dados tradicionais, isso seria trabalhoso, pois haveria a necessidade de implementar não só o *socket* novamente em uma nova plataforma, como também o desenvolvimento de uma API para o acesso em multiplataformas no mesmo banco de dados.

Em termos de desempenho, o *Firebase* além de ser uma ferramenta em *Cloud Computing* permitindo escalabilidade, conta com um modelo não relacional, que foi escolhido justamente por ter diferentes otimizações e funcionalidades comparados aos bancos de dados relacionais. Com isso, possibilita rápida atualização à todos os dispositivos conectados ao banco de dados. Em aplicações com bancos de dados tradicionais, é preciso contar com uma forma de adquirir recursos para o servidor caso a demanda exigir. Além disso, depende do *uptime* do servidor em que se encontra o banco de dados para

que todos os dispositivos conectados ao chat funcione. Com o *Firebase* quem se preocupa com a infraestrutura é a própria empresa Google, sua dona.

5. Conclusões e Trabalhos Futuros

Mediante as necessidade e dificuldades de se garantir o armazenamento e manipulação dos dados em diferentes plataformas, este trabalho propôs a análise de uma ferramenta para a criação de aplicações multiplataformas que compartilham um banco de dados em tempo real, que tem por objetivo facilitar o desenvolvimento de aplicações de alta qualidade em diversos dispositivos. Dentre os desafios encontrados, destaca-se a dificuldade de implementação de uma aplicação multiplataforma com a dependência da sincronização de dados em tempo real de forma rápida. Assim, foi realizado um estudo de caso comparando a ferramenta *Firebase* com os bancos de dados tradicionais com uma aplicação muito comum nos dias de hoje, um chat.

O *Firebase* apresentou-se uma boa alternativa para soluções em tempo real quando comparado aos bancos de dados tradicionais, sendo mais rápido e simples. Também foi constatado a facilidade do uso, em termos de manipulação de dados, quanto a implementação da aplicação em diferentes plataformas. Adicionalmente, a ferramenta se destaca pela preocupação com a segurança dos dados que são configuradas facilmente e podem ser alteradas a qualquer momento.

Os trabalhos futuros devem se concentrar na análise com as outras soluções disponíveis que complementam o banco de dados em tempo real do *Firebase*. Assim, permite um aprofundamento do estudo sobre o armazenamento dos dados como o *Firebase Storage* que permite o armazenamento de arquivos, como também o *Firebase Authentication* que trata da autenticação e segurança dos dados quando a natureza do aplicativo exige dados pessoais. Além disso, pode-se expandir o estudo para os outros serviços disponíveis no *Firebase* que são voltados para aplicativos móveis que podem facilitar a implementação, como *Firebase Notifications*, *Firebase Invites*, *Firebase Cloud Messaging*, entre outros.

References

Azure, W. (2017). Windows azure. `http://www.microsoft.com/azure/`. [Online; acessado 05-Março-2017].

Bokhari, M. U., Shallal, Q. M., and Tamandani, Y. K. (2016). Cloud computing service models: A comparative study. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 890–895.

Cleveroad (2016). A story of firebase or your next favourite cloud-based service. `https://www.cleveroad.com/blog/a-story-of-firebase-or-your-next-favourite-cloud-based-service`. [Online; acessado 05-Março-2017].

Engine, G. A. (2017). Google app engine. `https://appengine.google.com/`. [Online; acessado 05-Março-2017].

Google, F. (2017). Documentation firebase real-time database. `https://firebase.google.com/docs/database/`. [Online; acessado 05-Março-2017].

Oracle (2017). What is socket ? https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html. [Online; acessado 05-Março-2017].

Services, A. W. (2017). Amazon web services. https://aws.amazon.com/. [Online; acessado 05-Março-2017].

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18.