Fundação Faculdade de Filosofia, Ciências e Letras de Mandaguari

Curso de Ciência da Computação

IoT e integração com Micro-serviços:

Willian Marques Freire

Mandaguari, 2017

Willian Marques Freire

IoT e integração com Micro-serviços

Monografia apresentada ao curso Ciência da Computação
do Centro de Informática, da Fundação Faculdade de Filosofia Ciencias e Letras de Mandaguari,
como requisito para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Munif Gebara Júnior

Julho de 2017

Ficha catalográfica: elaborada pela biblioteca do CI.

Será impressa no verso da folha de rosto e não deverá ser contada.
Se não houver biblioteca, deixar em branco.

CENTRO DE INFORMÁTICA
Fundação Faculdade de Filosofia, Ciências e Letras de Mandaguari

Trabalho de Conclusão de Curso de Ciência da Computaçãointitulado ***IoT e integração com Micro-serviços*** de autoria de Willian Marques Freire, aprovada pela banca examinadora constituída pelos seguintes professores:

Prof. Mr. Nome do Professor B
Instituicao do Professor A

Prof. Dr. Nome do Professor B
Instituicao do Professor B

Prof. Dr. Nome do Professor C
Instituicao do Professor C

Coordenador(a) do Departamento Nome do Departamento
Nome do Coordenador
CI/UFPB

Mandaguari, July 26, 2017

Fundação Faculdade de Filosofia, Ciências e Letras de Mandaguari
Rua Rene Taccola, 152 Centro, Mandaguari, Paraná, Brasil CEP: 86975-000
Fone: +55 (44) 3233-1356

**** O sucesso é ir de fracasso em fracasso sem perder entusiasmo.*
*Winston Churchill ****

DEDICATÓRIA

A Deus, que nos criou e foi criativo nesta tarefa. Ao fôlego de vida que me tem sustentado na coragem de questionar o sentido da vida e propor soluções nas possíveis evoluções e revoluções que tem chegado à humanidade. A meu pai e mãe que me são meu motivo para viver e crescer. A meu orientador e professor, que não tem somente me mostrado um novo mundo tecnológico, mas também têm me ensinado a viver e crescer no mesmo. E a todos que me apoiam e motivam a acreditar em um mundo melhor.

AGRADECIMENTOS

Agradeço principalmente a DEUS e segundo a meus pais que tem me apoiado nas dificuldades, e a meu mentor e orientador, que tem me ensinado e auxiliado no desenvolvimento do meu trabalho, com críticas construtivas e com grande sapiência. Agradeço ao mesmo em particular por ter me ensinado a diferença entre dedução e indução, que difere o mundo ciêntifico dos demais. Agradeço também a meus colegas, que se sobrevelaram suas atitudade sociais comigo, demonstrando que são bons não somente em virtude da ciência, mas também na socialização e benevolência. Agradeço pela oportunidade do desenvolvimento deste trabalho, pois não tem aberto somente meus olhos para o mundo científico, mas também tem me ensinado a vivê-lo, e como posso melhorá-lo.

RESUMO

Este trabalho têm por objetivo a apresentação dos conceitos IoT *(Internet of Things)* e Micro-serviços, e provar que é possível desenvolver uma estrutura, que possibilita a integração entre estas tecnologias. Uma das justificativas para o desenvolvimento deste trabalho, tende para o fato que atualmente tem surgido diversos estudos sobre os assuntos citados, e têm beneficiado as aplicações desenvolvidas, com alta coesão e baixo acoplamento quando se trata de micro-serviços, e propiciado o compartilhamento de informações e integração a rede mundial de Internet quando se trata de IoT. Associando estas duas tecnologias que são direcionadas a aplicações e dispositivos conectados a internet, teve-se a idéia de organizar os dispositivos IoT, utilizando o conceito distribuido dos micro-serviços, concebendo assim, o conceito de coreografia aplicado aos micro-serviços, a estrutura IoT. Foram realizados diversos testes, e pesquisas em termo de tecnologias disponíveis para esta integração, e foi comprovado a possibilidade da mesma. Todo este assunto é tratado gradativamente durante três artigos, e ao final de cada um, é apresentado os resultados do mesmo.

Palavras-chave: <IoT>, <Internet>, <Micro>, <Interação>, <serviços>.

ABSTRACT

This paper aims to present the IoT *(Internet of Things)* concepts and Micro-services, and to prove that it is possible to develop a structure that allows the integration between these technologies. One of the justifications for the development of this work, tends to the fact that several studies have already appeared on the mentioned subjects, and have benefited the applications developed, with high cohesion and low coupling when it comes to micro-services, and favoring the sharing of Information and integration into the global Internet network when it comes to IoT. Associating these two technologies that are directed to applications and devices connected to the Internet, the idea was to organize the IoT devices, using the distributed concept of micro-services, thus conceiving the concept of choreography applied to the micro-services, the structure IoT. Several tests, and researches in term of technologies available for this integration were carried out, and the possibility of the same was proven. This whole subject is treated gradually during three articles, and at the end of each, the results of the same are presented.

Key-words: <IoT>, <Internet>, <Micro>, <Interation>, <Services>.

Sumário

1 INTRODUÇÃO

IoT e Micro-serviços, dois assuntos distintos, mas que de certa forma pode haver uma conexão entre os mesmos. Os dois serão foram explanados durante dois artigos, primeiramente IoT - A Internet das coisas e posteriormente Micro-serviços de Marques e Munif (2017). Neste contexto, aplica-se a integração dos mesmos, pois de fato, como são assuntos que que serão estudados nestes artigos, obteve-se a idéia de fazer com que os dois trabalhem em conjunto. Micro-serviço é um padrão têm originado muitos projetos, e os resultados têm sido positivos. Segundo Fowler (2014), Micro-serviço é mais um novo termo na área de arquitetura de software que descreve um estilo de sistemas de software, que tem se tornando o estilo padrão para o desenvolvimento de aplicações corporativas. Algumas características como alta coesão, autonomia, resiliência, observáveis, automatização e centralização no domínio de negócio fazem parte da arquitetura de micro-serviços.

Em um podcast realizado pela empresa Hipsters.tech que faz publicação de podcasts sobre tecnologias, no qual se encontrava funcionários da empresa Netflix, dentre eles Fabio Kung (senior software engineer), cita como a empresa está crescendo, e que um dos objetivos da mesma é ter uma estabilidade mais palatável. O mesmo também fala que atualmente, ainda grande parte dos sistemas da empresa, funcionam de forma molítica, e estão trabalhando no desacoplamento dos mesmos, para que um não afete os outros, e tenha a possibilidade de escalar facilmente partes específicas do sistema. Estimativas apontam que a empresa Netflix têm faturado somente no Brasil no ano de 2015 algo em torno de R$ 260 milhões com streaming, e para gerar tal tráfego de streaming de dados, é necessário uma arquitetura robusta, para que atenda o mesmo (FELTRIN, 2016).

Outra empresa que tem trabalhado com micro-serviços é a Amazon, uma das primeiras empresas em que migraram suas aplicações de um enorme sistema monolítico, para uma estrutura de micro-serviços, à procura de um modelo mais perspicaz quando se trata de atualização e suporte a aproximadamente 2 milhões de solicitações de 800 tipos diferentes de dispositivos. Grandes empresas atuais estão a desmontar os modelos arquiteturais monolíticos, privilegiando componentes mais pequenos e independentes que trabalham em conjunto para resolver determinados problemas (WORLD, 2016).

Aproveitando-se deste contexto tecnológico de distribuição de dados, um assunto que também está em dicussão é o IoT. O mesmo refere-se a uma revolução tecnológica que tem como objetivo, conectar itens utilizados no dia a dia à rede mundial de computadores. Segundo uma pesquisa realizada pelo IDC (Corporação Internacional de dados), em 2016 foi movimentado em média de US$41 bilhões somente nesta área.

O objetivo deste trabalho é desenvolver três artigos, o primeiro sobre IoT, o segundo sobre micro-serviços e o terceiro sobre integração entre IoT e micro-serviços. Considerando isto, será desenvolvido os mesmos, e cada artigo estará organizado da seguinte forma:

uma introdução sobre o assunto, uma revisão bibliográfica sobre o mesmo, a parte de desenvolvido de cada um e finalmente conclusão individual dos mesmos. Ao encerrar a escrita dos três artigos, será feito uma conclusão geral do trabalho e apresentado os resultados gerais.

IoT - A Internet das coisas

Willian Marques Freire e Munif Gebara Junior

***Resumo*—Este artigo têm por objetivo apresentar de maneira prática o desenvolvimento de uma estrutura IoT *(Internet Of Things)*, que tenha por finalidade a fácil configuração de dispositivos integrados. Um dos maiores problemas quando se trata de IoT, são as configurações iniciais complexas que são necessárias de ser feitas. Sendo assim, surgiu a necessidade de utilizar tecnologias atuais que possibilitam o desenvolvimento do mesmo, de maneira prática e simples. Foram realizados diversos teste, e pesquisa de dispositivos IoT que atendam os requisitos mínimos de recursos para desenvolvimento da arquitetura, e optou-se a utilização do dispositivo integrados NodeMCU ESP8266 pois contém módulo Wi-Fi, além dos recursos necessários. Todo este assunto, e a arquitetura será tratado durante este artigo, e ao fim do mesmo será apresentado os resultados da pesquisa e desenvolvimento.**

***Palavras-chave*—IoT, Internet, Coisas.**

I. Introdução

A Internet têm transcorrido por diversas etapas evolucionárias distintas. A primeira fase existente foi quando a Web foi chamada de ARPANET *(Advanced Research Projects Agency Network)*. A segunda fase da Web foi caracterizada pela concentração de todas as empresas, para compartilharem informações na Internet com intuito de divulgação de produtos e serviços. Seguido por uma terceira evolução, que mudou a Web, de um estágio com informações estáticas, para informações transacionais, nas quais produtos e serviços são comercializados totalmente online. Após todas estas evoluções, surge a quarta etapa, onde é criado o conceito de Web social e experiência do usuário, na qual empresas como Facebook e Twitter se tornaram famosas e profícuas, ao permitir que pessoas se comuniquem e compartilhem informações [1, p. 6].

Considerando estas evoluções tecnológicas, IoT ou Internet das coisas, algumas vezes referida como a Internet dos objetos, também têm revolucionado, mudando o modelo de *hardware* computacional, que nasceu a aproximadamente 40 anos atrás. Dentre todas as fases diferentes de modelos de *hardware* que existiram, comprova-se que a cada revolução IoT, surge a necessidade de modificar este modelo [3, p. 6]. Devido a diversos estudos visando o melhor desenvolvimento e aproveitamento do modelo cliente-servidor, surgiram vários protocolos para transmissão de dados, e o protocolo principal utilizado atualmente é o HTTP *(Hypertext Transfer Protocol)*, encontrando-se por padrão em praticamente quase todos os navegadores atuais. Existem ainda outros protocolos como FTP *(File Transfer Protocol)* para transferência de arquivos, IMAP *(Internet Message Access Protocol)* para envio de Mensagens, entre outros.

Faculdade de Filosofia Ciências e Letras de Mandaguari é uma fundação situada em Mandaguari no Paraná região sul brasileira, na rua Rene Taccola, 152 - Centro Site: (see http://www.fafiman.br/index.html).

Artigo realizado em 2017.

Aproveitando-se destes protocolos, também surgiram especificações e arquiteturas, para que aplicações possam ser disponibilizadas como serviços. Dentre elas encontra-se o REST *(Representational State Transfer)* ou Transferência de Estado representacional, que segundo Fielding [20], é uma abstração da arquitetura World Wide Web, um estilo arquitetural, que consiste em um conjunto coordenado de restições aplicadas a componentes, conectores e elementos de dados dentro de um sistema de hipermídia distribuído, entretanto, é um assunto para um próximo trabalho.

Considerando que o IoT representa a próxima evolução da Internet, melhorando a capacidade de coletar, analisar e distribuir dados, faz-se com que os mesmos possam ser transformados em informação. Atualmente existem projetos IoT em desenvolvimento, que prometem fechar a lacuna entre ricos e pobres, melhorando a distribuição dos recursos mundiais para aqueles que precisam deles, e ajudando a entender a sociedade atual para que possa ser mais proativa e menos reativa [1, p. 2]. Em 2003, haviam aproximadamente 6.3 bilhões de pessoas vivendo no mundo, e aproximadamente 500 milhões de dispositivos conectados à internet. O crescimento de smartphones e tablets, elevou o número de dispositivos conectados a Internet para aproximadamente 12,5 bilhões em 2010 [1, p. 3].

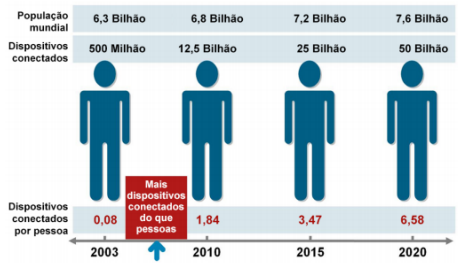


Figura 1. "Nascimento"do IoT entre 2008 e 2009 [1].

Anteriormente, em janeiro de 2009, uma equipe de pesquisadores chinesa, finalizou um estudo sobre os dados de roteamento da internet em intervalos de seis meses, e foi descoberto pelos mesmos, que a Internet dobra de tamanho a cada 5,32 anos (EVERS, 2003). Na figura 1, é possível observar o refinamento desta pesquisa. Devido a este crescimento no número de dispositivos conectados a internet, no artigo Internet of Things volume III, publicado pela empresa Dzone (empresa que faz publicações online sobre tecnologia) no ano de 2017, foi feito uma pesquisa interna na área de IoT. Na mesma, encontram-se 797 profissionais que são responsáveis por esta área, e foi questionado sobre as maiores preocupações quando se trata de IoT. O centro de preocupação estava pela segurança e privacidade, pois os mesmos, estavam

interessados em IoT para o contexto empresarial de suas empresas. A falta de padrões, era um terço na coluna "muito preocupada" chegando a 34% dos envolvidos. Dentre eles, 25% dos entrevistados também desconfiam da conectividade e do baixo custo de energia, sendo que 24% se preocupam com a manutenção de *hardware* e *software*, e 14% estão apreensivos com o desenvolvimento imaturo e paradigmas de redes [1, p. 4].

O objetivo deste trabalho, é exemplificar o processo de criação de uma estrutura IoT, e demonstrar a resolução da lacuna de configuração de um dispositivo IoT de forma simples. Será desenvolvido durante este trabalho uma estrutura IoT flexível, que seja capaz de se comunicar através do protocolo HTTP com serviços WEB, que ficarão encarregados de compartilhar as informações recebidas dos dispositivos IoT pela Internet. Este artigo está organizado da seguinte forma: primeiramente será feito uma revisão bibliográfica sobre os assuntos que serão tratados, posteriormente será desenvolvido uma estrutura IoT que vise a fácil configuração da mesma, será feito também uma apresentação de resultados e por fim a conclusão deste artigo.

A. Revisão Bibliográfica

1) IoT: Atualmente, existem diversos tipos e marcas de placas com circuito integrado e computadores, para desenvolvimento IoT. Em uma nova pesquisa realizada pelo Dzone, foram entrevistadors diversas pessoas, para obter um percentual de preferência entre dispositivos. Dentre os mesmos, 53% preferem Rapberry Pi, 28% preferem o Arduino, e 19% não apresentam nenhuma preferência. Nesta pesquisa, foram relatados protocolos mais utilizados dentro do ramo IoT. Dentre os entrevistados, 14% disseram ter preferência por Wifi-Direct em produção, 8% preferem utilizar Bluetooh LE em ambientes que não são de produção. No total, 24% já havia utilizado Wifi-Direct e 23% Bluetooth LE. Um protocolo também citado, é o MQTT *(Message Queue Telemetry Transport)*, que segundo o site FilipeFlop [22], é um protocolo de mensagens leve, criado para comunicação M2M *(Machine to Machine)*, que obteve 33% de preferência de utilização em produção [21, p. 4].

Assim como aplicações web ou móveis, dispositivos físicos também precisam ser consensuais. No entanto, existem desafios adicionais para o IoT. As arquiteturas atuais ainda deixam a desejar em questão de terminais e configuração. Uma segunda preocupação na área de IoT é o tempo necessário para atualização de software para o dispositivo. Dentre as complicações que poder existir, a vulnerabilidade de um dispositivo, pode ser alvo para ataques e invasões. Uma das maneiras de gerenciar a testabilidade do software no dispositivo, é testar isoladamente. Reproduzir o ecossistema em torno do dispositivo e como os usuários interagem garantem uma boa testabilidade. Estudos avançados reconhecem que não se podem identificar, e muito menos testar cada possível cenário de falha. Por este fato, estes estudos concentram cada vez mais na confiabilidade e segurança IoT. O ultimo desafio para esta área, são as identificações e dignosticação de falhas. Tomar ações rapidamente para resolução de problemas, é um aspecto fundamental no desenvolvimento de dispositivos IoT. Técnicas para integração continua são utilizados para resolver estes problemas, e uma delas é a *Canary Release*, que Segundo Danilo Sato [14], é uma técnica para reduzir o risco de introduzir uma nova versão de software na produção, lançando mudanças lentamente para um pequeno subconjunto de usuários, antes de lança-la em toda a infra-estrutura e torná-la disponível para todos [21, p. 9].

Nas últimas duas décadas, houveram avanços tecnológicos na área computacional, surgindo processadores com mais capacidade, armazenamentos, memória e dispositivos de rede com baixo custo. Atualmente, dispositivos físicos estão sendo desenvolvidos com mais capacidade computacional, e interligados através da internet de maneira efetiva. A adoção generalizada das tecnologias IoT enriquecem a idéia de computação ubíqua, um conceito que surgiu no final dos anos 80. Mark Weiser, criador deste conceito, escreveu em seu artigo The Computer for the 21st Century, que o computador se integra a vida das pessoas de modo que elas não o percebam, todavia, o utilizam. Motivado por esta convicção, Weiser percebeu que em sua época não haviam tecnologias necessárias para que a Computação Ubíqua se tornasse realidade, fazendo com que assim, dedicasse esforços para desenvolver estes meios. [11].

Após alguns anos, no início de 1926, uma revista chamada Collier publicou uma entrevista com Nikola Tesla, no qual ele falou sobre suas previsões para as próximas décadas. Entre elas, ele falava de um mundo com máquinas voadoras, energia sem fio e superioridade feminina. Segundo palavras de Tesla,s quando a tecnologia sem fio estivesse perfeitamente estabelecida em todo o mundo, o planeta se tornaria um enorme cérebro. Nesta entrevista, ele disse que inclusive os seres humanos seriam capazes de se comunicar uns com os outros de imediato, independente da distância [12].

Além disto, em um artigo feito pelo professor Michael Wooldridge (chefe do Departamento de Ciência da Computação da Universidade de Oxford), definiu um termo utilizado no meio tecnológico, que são os agentes. Segundo ele, um agente é um computador que está situado em algum ambiente, capaz de realizar ações autônomas sobre este, para cumprir seus objetivos delegados. O mesmo torna-se inteligente com as seguintes propriedades: Reatividade (a percepção de seu ambiente, e a capacidade de respota em tempo hábil sobre mudanças que ocorrem), proatividade (antecipação de problemas futuros), habilidade social (capacidade de interação entre agentes para satisfazer seus objetivos de design) [13].

2) NodeMCU: Neste trabalho, será utilizado o NodeMCU para desenvolvimento das aplicaçoes IoT, pelo fato do mesmo conter um módulo WiFi, o que facilitará na comunicação via interface de rede com outros dispositivos. NodeMCU é um firmware baseado em eLua (implementação completa utilizando programação Lua para sistemas embarcados) [16]. O mesmo foi projetado para ser integrado com o Chip WiFi ESP8266 desenvolvido pela empresa Espressif, situada em Shangai, especializada no ramo de IoT [17]. O NodeMCU utiliza sistema de arquivos SPIFFS *(SPI Flash File System)* e seu repositório no Github consiste em 98.1% de código na linguagem C - criada em 1972 por Dennis Ritchie [18] e o demais existente em código escrito na linguagem Lua - criada em 1993 por Roberto Ierusalimschy, Luiz Henrique de

Figueiredo e Waldemar Celes [19].

ESP8266 é um chip com arquitetura 32 bits, e o seu tamanho está em 5mm x 5mm. Existem diversos módulos parecidos. Dentre eles estão, o ESP-01 que contém 8 conectores, e surgiu para ser utilizado como um módulo para o Arduino, o ESP-07 que contém 16 pinos, antena, cerâmica e conector para antena externa, e o ESP-12E, que conta com 22 pinos, que possibilita a ligação de diversos módulos ao ESP como *displays*, cartões SD, dentre outros. Um ponto importante nestes módulos, é que eles utilizam 3,3V. Comparado ao Arduino UNO, o NodeMCU se destaca por ter um processador Tensilica LX106, que pode atingir até 160MHZ, e possui uma memória RAM de 20KB e uma memória flash de 4MB. Já o Arduino UNO possui um micro controlador de 16MHZ, possui uma memória RAM de 2KB e uma memória flash de 32KB. Outra questão a ser levado em conta, é o custo benefício, pois atualmente é possível encontrar o ESP8266 por até US$1,70, e um Arduino UNO ultrapassa os 20 dólares[10].

3) Módulo Wi-Fi: Devido ao elevado número de cabos necessários para interconectar computadores e dispositivos, surgiu o WiFi. É muito utilizado cabos, entretanto possuem algumas limitações. Um exemplo é o deslocamento dos mesmos, é trabalhoso pelo fato de possuir limite de alcance, e em ambientes com muitos computadores, são necessárias apropriações estruturais. O uso do Wi-Fi tem se tornando comuns nao somente em residências, mas também em ambientes corporativos e públicos. Dentre os padrões de Wi-Fi existentes, estão o 802.11b que tem como possibilidade p estabelecimento de conexões nas seguintes velocidades de transmissão: 1 Mb/s, 2Mb/s, 5,5 Mb/s e 11 Mb/s, o 802.11g que surgiu em 203 e é totalmente compatível com o 802.11b, e possui como atrativo a possibilidade de trabalhar com taxas de transmiss ao de até 54 Mb/s, o 802.11n que tem como principal característica a utilização de um esquema chamado MIMO ou *Multiple-Input Multiple-Output*, que é capaz de atingir taxas de transmissão de até 600 Mb/s, e finalmente o 802.11ac, também chamado 5G WiFi, que sua principal vantagem está em sua velocidade, estimada em até 433 Mb/s no modo mais simples, fazendo com que, é possível fazer a rede suportar 6 Gb/s de velocidade na trasmiss ao de dados. Resumidamente, Wi-Fi é um conjunto de especificaçoes para redes locais sem fio baseada no padrão IEEE 802.11, uma abreviatura para *"Wireless Fidelity"*. [4]

4) Segurança Wi-Fi: Apesar de todas as facilidades que se encontram ao utilizam tecnologias sem fio como Wi-Fi, assim como todo sistemas, medidas de segurança devem ser utilizadas para previnir ataques e invasões. Existem protocolos de segurança que auxiliam na segurança de conexões Wi-Fi, dentre eles se encontram: WEP, WPA, e o WPA2. *Wired Equivalent Privacy* (WEP) é um algoritmo de segurança que foi criado em 1999 e é compatível com prativamente todos os dispositivos Wi-Fi disponível no mercado. Este padrão se torna mais inseguro à medida que o poder de processamento dos computadores aumenta. Pelo fato de conter um numero máximo de combinações de senha totalizando 128 bits, é possível descobrir a palavra-passe em poucos minutos por meio de softwares de ataques. *Wi-Fi Protected Access* ou WPA, surgiu quando o WEP saiu de circulação, e entrou como protocolo-padrão industrial. Adotado em 2003, trazia como novidade a encriptação 256 bits e como segurança adicional, fazia análise de pacotes, para verificar alterações e invasões. Atualmente é utilizado o *Wi-Fi Protected Access II* ou WPA2, pelo fato de ser o mais seguro. Foi implementado pela Wi-Fi Alliance em 2006, e possui como diferencial a maneira como lida com senhas e algoritmos, excluindo totalmente a possibilidade de um ataque de força bruta. Segundo especialistras, o riscos de intrusos em redes domésticos é quase zero. Isto se deve a utilização de duas tecnologias que são utilizadas neste algoritmo, o AES (Advanced Encryption Standard) que é um novo padrão para segurança das informações, e o CCMP *(Counter Cipher Mode)*, um mecanismo de encriptação que protege os dados que passam pela rede. Devido a complexidade do mesmo, muitos dispositivos, mesmos recentes, não são compatíveis com ele. [8]

5) Sistema de arquivos: O NodeMCU utiliza o sistema de arquivos SPIFFS, um sistema de arquivos destinado a dispositivos *flash* embarcados. SPIFFS é projetado para projetos que são pequenos e memória sem pilha [6]. O projeto pode ser clonado do repositório do mesmo no github.

6) Ferramenta para desenvolvimento: Neste projeto será utilizado uma ferramenta chamada ESPlorer. ELa é utlizada para facilitar o desenvolvimento de aplicações LUA para dispostivos ESP8266. Basicamente, é uma IDE (Ambiente de desenvolvimento integrado), que possui diversos facilitadores para melhor visualização do código e envio do código fonte para o dispositivo. Esta ferramenta possui um terminal para *debug*, e acessos rápidos para executar comandos. Nela também é possível executar comandos de forma rápida. Todos estes fatores beneficiaram na escolha de utilização desta ferramenta para o desenvolvimento do trabalho [7]. Na figura 2 é possível observar estas características.

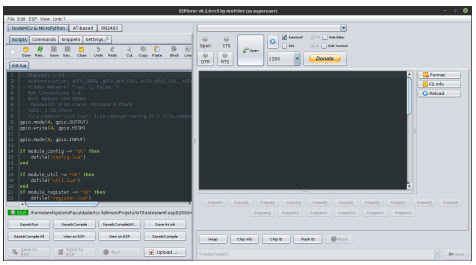


Figura 2. Esplorer [7]

7) Protocolos TCP e UDP: O protocolo TCP ou Protocolo de controle de transmissão, é um dos protolocos mais utilizados. Ele é complementado pelo IP (Protocolo da Internet), sendo normalmente chamado de TCP/IP. A versatilidade e robustez do mesmo tornou-o mais adequado para utilização em redes globais, já que são feito verificação de dados para confirmação de recebimento e entrega sem erros. O TCP é um protocolo que se encontra no nível da camada 4 ou camada de transporte do modelo OSI (sistema de interconexão aberto), que é um modelo de referência da ISO (Organização Internacional para Padronização) dividido em camadas de funções, que são: Aplicação que fornece serviços às aplicações do cliente, Apresentação que fornece encriptação e compressão de dados, além de garantir a compatibilidade entre camadas

de aplicação de sistemas diferentes, Sessão que controla as sessões entre aplicações, transporte que controla o fluxo de informação e controle de erros, rede que encaminha pacotes e possui esquema de endereçamentos, dados que controla o acesso o meio físico de tansmissão e erros da camada física, e finalmente a camada física, que define as características do meio físico de transmissão da rede, conectores, interfaces, codificação ou modulação de sinais [23][24].

Outro protocolo que também será utilizado, é o UDP *(User Datagram Prococol)*. Antes de entender a diferença do UDP para o TCP é necessário saber o que é um datagrama. Segundo a RFC 1594 [25] é uma entidade de dados completa e independente que contém informações suficientes para ser roteada da origem ao destino sem precisar confiar em trocas anteriores entre essa fonte, a máquina de destino e a rede de transporte. O UDP permite que a aplicação envie um datagrama encapsulado em um Pacote IPv4 (ou IPv6, e então enviado ao destino. Ao contrário de TCP, no UDP não há qualquer tipo de garantia que o pacote será entregue.

B. Desenvolvimento

1) Configurações iniciais no NodeMCU ESP-8266: Primeiramente, para se utilizar este dispositivo, é necessário fazer o build (compilação) do firmware, conforme a figura 3, pois o mesmo é distribuido sem nenhum sistema operacional. Atualmente, pode-se fazer o build do mesmo utilizando o site *https://nodemcu-build.com* ou clonar o repositório do projeto do github. No caso deste projeto, o build do projeto será feito manualmente. Para incluir bibliotecas neste módulo, é necessário editar o arquivo que se encontra no diretório *app/include/user_modules.h* do projeto e retirar os comentários das bibliotecas que será utilizado.

```
#define LUA\_USE_MODULES\_MQTT
// #define LUA\_USE\_MODULES_COAP
// #define LUA\_USE\_MODULES\_U8G
```

Figura 3. Compilação do firmware

Duas possibilidades disponível pelo módulo NodeMCU, são ativar o suporte TLS e o debug (depurador) quando em execução. Para utilizar o suporte TLS, é necessário descomentar a opção *CLIENT_SSL_ENABLE* que se encontra dentro do arquivo *user_config.h* no diretório app/include, e para ativar o modo debug em tempo de execução, e descomentar a linha em que se encontra *#define DEVELOP_VERSION*.

Por padrão, o build do firmware é gerado com suporte a variáveis com ponto flutuante, entretanto, isto ocupa mais memória, e para reduzir a quantidade de memória utilizada, deve-se comentar em que é definido a constante *LUA_NUMBER_INTEGRAL* dentro so arquivo *user_config.h*.

2) Gerando e gravando o firmware: Após ajustar as devidas configurações inicial, é necessário compilar e gravar o *firmware* no dispositivo, conforme a figura 4. Neste trabalho, será utilizado a ferramenta esptool para gravar o mesmo. Esta ferramenta pode ser encontrada no repositório da Espressif (empresa que desenvolveu o NodeMCU) no github, e foi iniciada por Fredrik Ahlbeg, um dos envolvidos no projeto, e atualmente é mantido por Angus Gratton e pela comunidade. E para compilar os códigos fontes, deve-se primeiramente executar os seguintes comandos dentro do diretório do projeto.

```
$ docker pull marcelstoer/nodemcu-build
$ sudo docker run --rm -ti -v `pwd`:/opt
/nodemcu-firmware
marcelstoer/nodemcu-build
```

Figura 4. Compilação do firmware

Como este projeto utiliza o docker, uma infraestrutura independente de plataforma [5], primeiramente é atualizado o container e posteriormente é executado um comando para compilar o projeto. Após compilado, deve ser executado os comandos conforme a figura 5, utilizando a ferramenta esptool. Estes comandos serão utilizados para gravar o firmware no ESP8266.

```
$ sudo python esptool.py
    --port="/dev/ttyUSB1"
    erase_flash

$ sudo python ./esptool.py
    --port /dev/ttyUSB0
    write_flash
    0x00000
    ../nodemcu-firmware/bin/0x00000.bin
    0x10000
    ../nodemcu-firmware/bin/0x10000.bin

$ sudo python ./esptool.py
    --port="/dev/ttyUSB0"
    write_flash -fm=dio
    -fs=16m 0x00000
    ./../nodemcu-firmware/bin
    /nodemcu_integer__20170510-0106.bin
```

Figura 5. Compilação do firmware

Os comandos da figura 5 estão divididos em 3 partes. Primeiramente é removido todo o firmware do dispositivo, posteriormente é gravado nos endereçamentos 0x00000 e 0x10000 os arquivos com bibliotecas padrões utilizadas no firmware. Após isto é gravado o firmware com as bibliotecas extras selecionadas para inclusão no dispositivo.

3) Access Point para configuração: Assim como todo dispositivo, neste trabalho será desenvolvido uma central de configuração. Como este dispositivo foi desenvolvido com foco na utilização e conexão Wi-fi, será criado uma central para configuração sobre qual dispositivo Wireless o mesmo se conectará, e para isto, como a maioria destes dispositivos possuem regras de segurança e senhas, deverá conter também a opção para que seja configurado a senha do dispositivo.

Um dos recursos que o ESP8266 provê, é o desenvolvimento de um AP (*Access Point* ou ponto de acesso), que permite interligar duas ou mais redes sem fio. Dentre as diversas funções de um AP estão: repetir um sinal e transformar um sinal de um cabo em sinal sem fio [9]. Entretanto, neste trabalho será utilizado somente como central para configuração do dispositivo.

Primeiramente é necessario fazer o NodeMCU trabalhar com o padrão Access Point. Ele provê um facilitador para que o dispositivo trabalhe como Access Point e como cliente. Para isto é necessário fazer conforme a figura 6. Neste trabalho será criado inicialmente um arquivo chamado *init.lua*, onde se encontrará o código fonte que será executado ao iniciar o dispositivo.

```
wifi.setmode(wifi.STATIONAP)
```

Figura 6. Configurando modo Access Point.

Uma outra possibilidade ao utilizar o NodeMCU, é a criação de um pequeno servidor em uma de suas portas. Neste caso, será criado um servidor TCP conforme a figura 7, para que quando alguém se conectar no dispositivo em uma determinada porta padrão, esteja disponível a central de configuração caso o dispositivo não esteja conectado em nenhum roteador Wireless.

```
srv = net.createServer(net.TCP)
```

Figura 7. Criando Servidor TCP.

Uma das questões encontradas neste trabalho, foi a forma de como manter estas configurações em funcionamento, pois se ocorrer algo que faça com que o dispositivo desligue, seria interessante que o mesmo mantivesse as configurações salvas, para que não seja necessário a reconfiguração do Wi-Fi. Para isto, será utilizado o sistema de arquivos do ESP8266, para criação de um arquivo config.lc, conforme a figura 11, e sempre que o dispositivo iniciar, será feito a verificação da existência do arquivo de configurações para conexão do Wi-FI, e se caso não existir este arquivo, será iniciado a central de configuração do dispositivo.

Para configurar um Access Point, é necessário algumas considerações como: ssid (Nome da conexão), pwd (Senha da conexao), auth (Tipo de autenticação), channel (Canal que será liberado no dispositivo Wi-Fi), hidden (indica se é visível), max (máximo de conexões simutâneas) e beacon (intervalo de tentativas de conexão do dispositivo). Quando é encontrado o arquivo, é feito uma leitura do mesmo, e neste trabalho, está sendo salvo no padrão usuário e senha separados por um espaço em branco. Os demais códigos desenvolvidos, podem ser encontrados no apêndice deste trabalho. No arquivo *init.lua* deverá ser feito a importação dos mesmos, pois serão divididos da seguinte forma: *init.lua* (arquivo inicial), *config.lua* (arquivo que terá o código de configuração) e util.lua (arquivo que terá funções utilitárias). Para importar os demais módulos, foi feito conforme a figura 8.

```
dofile("config.lua")
dofile("util.lua")
dofile("register.lua")
```

Figura 8. Importação dos módulos

II. Resultados

O NodeMCU ESP8266 possui um módulo Wi-Fi, o que possibilitou a conexão à rede de internet. O mesmo possui um sistema de arquivos, o que permitiu as configurações serem salvas no dispositivo, para que se caso ocorra alguma falha, ou até mesmo o dispositivo seja desligado, não seja necessário fazer novamente as configuraçãos iniciais. Foi implementado uma regra no dispositivo, que faz a verificação de conexão a internet dentro de 20 segundos, e se ocorrer algulma falha na conexão, será deletado o arquivo de configurações e reinicializado o dispositivo, liberando assim, o acesso à configuração de rede.

Como este dispositivo tem suporte a protocolos TCP e UDP, é aberto possibilidades para integração com outros dispositivos ou sistemas que utilizam este protocolo. Foram realizados diversos testes com o mesmo, dentre eles utilizando o GPIO [26] para utilização de pinos digitais, SJSON para serialização de Objetos utilizandos textos no padrão JSON [27] e o Timer para executar funções em determinado tempo [28].

III. Conclusões e trabalhos futuros

Utilizando-se destes recursos disponíveis no NodeMCU ESP8266, é possível montar uma estrutura IoT flexível e de simples configuração. Atualmente existem outros dispositivos embarcados semelhantes que são acessíveis. Entretanto, neste trabalho foi utilizado este, pelo fato de ter um custo-benefício acessível, e de estar sendo utilizado pela comunidade. Foi utilizado a linguagem LUA para desenvolver neste dispositivo, entretanto, existem alguns projetos que possibilitam o desenvolvimento em outras linguagens. Como neste trabalho foi desenvolvido uma estrutura IoT de fácil configuração e com tecnologias como TCP e UDP, foi aberto possibilidades para o desenvolvimento de sistemas que se integrem com o mesmo, utilizando a rede de internet, e trabalhos futuros podem visar esta possibilidade. Sistemas complexos de automação residencial ou industrial podem ser resolvidos acoplando estas tecnologias, integrando o dispositivo com a rede de internet de forma fácil, foi resolvida.

APÊNDICE A
CÓDIGO FONTE CONFIGURACIONAL: CONFIG.LUA

```
function configureWifi()
  local ap = {}
  wifi.sta.getap(
    function(t)
      i = 0
      for ssid, v in pairs(t) do
          ap[i] = ssid
          i = i + 1
      end
    end)
  srv:listen(
    80,
    function(conn)
      conn:on(
        "receive",
        function(conn, payload)
          listaAp = "SSID: <select name='ssid'>"
          print("Request Configuration")
          for i = 0, tablelength(ap), 1 do
            if ap[i] ~= nil then
              listaAp = listaAp
                .. "<option value='"
                .. ap[i] .. "'>"
                .. ap[i] .. "</option>"
            end
          end
          listaAp = listaAp .. "</select><br/>"

          local _GET = getParamsUrl(payload)

          if (_GET ~= nil) then
            if (_GET.ssid ~= nil
              and _GET.password ~= nil) then
              print("Connecting to wifi
              and creating configuration...")
              print("SSID: "
                .. _GET.ssid)
              print("Password: "
                .. _GET.password)

              for i = 0, tablelength(ap), 1 do
                if ap[i] ~= nil then
                  if (string
                    .find(ap[i], _GET.ssid)) then
                    user = ap[i]
                  end
                end
              end

              password = _GET.password
              wifi.sta.config(user, password)
              wifi.sta.connect()

              if file.open("config.lc", "w") then
                file.writeline(user
                  .. " " .. password)
                file.close()
              end

              node.restart()
              node.chipid()
            end
          end
          conn:send([[... HTML Whatever...]]))
        conn:on("sent", function(conn)
          conn:close()
          collectgarbage()
        end)
    end)
end
```

Figura 9. Módulo Utilitário

APÊNDICE B
CÓDIGO FONTE UTILITÁRIO: UTIL.LUA

```
function split(str, pat)
    local t = {}
    -- NOTE: use {n = 0} in Lua-5.0
    local fpat = "(.-)" .. pat
    local last_end = 1
    local s,
          e,
          cap = str:find(fpat, 1)
    while s do
        if s ~= 1 or cap ~= "" then
            table.insert(t, cap)
        end
        last_end = e + 1
        s,
        e,
        cap = str:find(fpat, last_end)
    end
    if last_end <= #str then
        cap = str:sub(last_end)
        table.insert(t, cap)
    end
    return t
end

local unescape = function(s)
    s = string.gsub(s, "+", " ")
    s = string.gsub(
        s,
        "%%(%x%x)",
        function(h)
            return string.char(tonumber(h, 16))
        end
    )
    return s
end

function tablelength(T)
    local count = 0
    for _ in pairs(T) do
        count = count + 1
    end
    return count
end

function getParamsUrl(request)
    local buf = ""
    local
      _, _,
      method,
      path,
      vars = string
        .find(request, "([A-Z]+) (.+)?(.+) HTTP")
    if (method == nil) then
        _, _,
        method,
        path = string.find(request, "([A-Z]+) (.+)
    HTTP")
    end
    local _GET = {}
    if (vars ~= nil) then
        for k, v in string
          .gmatch(vars, "(%w+)=(%w+)&*") do
            _GET[k] = unescape(v)
        end
    end
    return _GET
end
```

Figura 10. Módulo Configuracional

APÊNDICE C
CÓDIGO FONTE INICIAL: INIT.LUA

```
1
2  srv = net.createServer(net.TCP)
3
4  if file.exists("config.lc") == true then
5
6      print("Open configuration...")
7
8      if file.open("config.lc") then
9
10         local corte =
11           split(file.read(), " ")
12         user = corte[1]:gsub("%s+", "")
13         password = corte[2]:gsub("%s+", "")
14
15         print("Connecting in ip with user: "
16         .. user
17         .. " and password: "
18         .. password)
19         wifi.sta.config(user, password)
20         wifi.sta.connect()
21
22     end
23
24 else
25
26     wifi.ap.config({
27       ssid = "NodeMcuEsp8266"
28         ..node.chipid(),
29       pwd = nil,
30       auth = AUTH_OPEN,
31       channel = 6,
32       hidden = 0,
33       max = 4,
34       beacon = 100
35     })
36
37     wifi.ap.setip({
38       ip = "192.168.10.1",
39       netmask = "255.255.255.0",
40       gateway = "192.168.10.1"
41     })
42
43     wifi.ap.dhcp.config(
44         {start = "192.168.10.2"}
45     )
46
47     tmr.alarm(
48         1,
49         1000,
50         1,
51         function()
52             if wifi.ap.getip() == nil then
53                 print("Connecting...")
54             else
55                 print("Connected in "
56                   .. wifi.ap.getip())
57                 configureWifi()
58             end
59             tmr.stop(1)
60         end
61     )
62
63 end
```

Figura 11. Central de Configuração

REFERÊNCIAS

[1] Dave Evans. *T*he Internet of Things How the Next Evolution of the Internet Is Changing Everything, vol 1, pp. 2-4, 2011

[2] Willian Marques Freire e Munif Gebara Júnior. *M*icro-serviços, vol 1, pp. 1-3, 2017

[3] John Esposito. *T*he Dzone Guide to Internet of Things, vol 3, pp. 1-6, 2016

[4] Emerson Alecrim. *O* que é Wi-Fi (IEEE 802.11)? 2013 [Online] Disponível: https://www.infowester.com/wifi.php#80211. [Acesso: 20-Mai-2017]

[5] Docker. *D*ocker 2017 [Online] Disponível: https://www.docker.com. [Acesso: 11-Jun-2017]

[6] Peter Andersson. *S*PIFFS (SPI Flash File System) 2013 [Online] Disponível: https://github.com/pellepl/spiffs. [Acesso: 11-Jun-2017]

[7] ESPlorer. *E*SPlorer Integrated Development Environment (IDE) for ESP8266 developers 2013 [Online] Disponível: https://github.com/4refr0nt/ESPlorer. [Acesso: 11-Jun-2017]

[8] Felipe Demartini. *W*EP, WPA, WPA2: o que as siglas significam para o seu WiFi? 2013 [Online] Disponível: https://www.tecmundo.com.br/wi-fi/42024-wep-wpa-wpa2-o-que-as-siglas-significam-para-o-seu-wifi-.htm. [Acesso: 03-Jun-2017]

[9] Mundo Max. *Q*ual a diferença entre um Roteador Wireless e um Access Point? 2011 [Online] Disponível: http://www.mundomax.com.br/blog/informatica/qual-a-diferenca-entre-um-roteador-wireless-e-um-access-point/. [Acesso: 03-Jun-2017]

[10] Irving Souza Lima. *N*odeMCU (ESP8266) o módulo que desbanca o Arduino e facilitará a Internet das Coisas. 2016 [Online] Disponível http://irving.com.br/esp8266/nodemcu-esp8266-o-modulo-que-desbanca-o-arduino-e-facilitara-a-internet-das-coisas/ [Acesso: 05-Jun-2017]

[11] Mark Weiser. *T*he Computer for the 21st Century, vol 1, pp. 1-2, 1991

[12] John B. Kennedy. *W*HEN WOMAN IS BOSS, 1926 [Online] Disponível: http://www.tfcbooks.com/tesla/1926-01-30.htm. [Acesso: 10-Mai-2017]

[13] Michael Wooldrige e Nicholas R. Jennings. *I*ntelligent agents: theory and practice, vol 1, pp. 1-3, 1995

[14] Danilo Sato. *C*anaryRelease. 2014 [Online] Disponível: https://martinfowler.com/bliki/CanaryRelease.html. [Acesso: 10-Abr-2017]

[15] NodeMCU. *N*odeMCU 2.0.0. [Online] Disponível: https://github.com/nodemcu/nodemcu-firmware. [Acesso: 10-Dez-2017]

[16] ELua. *W*hat is eLua. [Online] Disponível: http://www.eluaproject.net/overview. [Acesso: 19-Nov-2017]

[17] Systems. *E*xpressif Systems. [Online] Disponível: http://espressif.com/company/contact/pre-sale-questions. [Acesso: 11-Nov-2016]

[18] William Stewart. *C* Programming Language History. [Online] Disponível: http://www.livinginternet.com/i/iw_unix_c.htm. [Acesso: 15-Mai-2017]

[19] Lua. *A*uthors. [Online] Disponível: http://www.lua.org/authors.html. [Acesso: 5-Fev-2017]

[20] Roy Thomas Fielding. *R*epresentational State Transfer (REST. [Online] Disponível: http://www.ics.uci.edu/ fielding/pubs/dissertation/-rest_arch_style.htm. [Acesso: 29-Mai-2017]

[21] Cate Lawrence. *I*nternet of Things Applications, Protocols, and Best Practices, vol 4, pp. 1-10, 2017

[22] Filipeflop. *C*ontrole e monitoramento IoT com NodeMCU e MQTT. 2016 [Online] Disponível: http://blog.filipeflop.com/wireless/controle-monitoramento-iot-nodemcu-e-mqtt.html. [Acesso: 10-Abr-2017]

[23] Pedro Pinto. *R*edes – Sabe o que é o modelo OSI? 2010 [Online] Disponível: https://pplware.sapo.pt/tutoriais/networking/redes-sabe-o-que-e-o-modelo-osi. [Acesso: 17-Jun-2017]

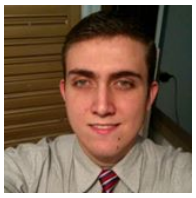
[24] Vinton G. Cerf, Robert E. Kahn. *A* Protocol for Packet Network Intercommunication, IEEE Transactions on Communications, vol. 22, pp. 637-648, 1974

[25] April N. Marine et Al. *R*FC 1594, 1994 [Online] Disponível: https://tools.ietf.org/html/rfc1594. [Acesso: 17-Jun-2017]

[26] NodeMCU. *G*PIO Module? 2014 [Online] Disponível: https://nodemcu.readthedocs.io/en/master/en/modules/gpio/ [Acesso: 17-Jun-2017]

[27] NodeMCU. *S*JSON Module 2017 [Online] Disponível: https://nodemcu.readthedocs.io/en/master/en/modules/sjson/ [Acesso: 17-Jun-2017]

[28] NodeMCU. *T*imer Module 2014 [Online] Disponível: https://nodemcu.readthedocs.io/en/master/en/modules/tmr/ [Acesso: 17-Jun-2017]



Willian Marques Freire Possui Ensino Médio completo pelo Colégio Estadual Rosa Delúcia Calsavara (2013). Atualmente é Desenvolvedor de Software da Gumga Tecnologia da Informação S/A. Tem experiência na área de Ciência da Computação.

Munif Gebara Júnior Possui graduação em Ciência da Computação pela Universidade Estadual de Maringá (1997) e mestrado em Engenharia Elétrica e Informática Industrial pela Universidade Tecnológica Federal do Paraná (2001). Atualmente é professor da Fundação Faculdade de Filosofia Ciências e Letras de Mandaguari e professor de ensino superior da Faculdade de Tecnologia e Ciências do Norte do Paraná Ltda e desenvolvedor.

Micro-serviços

Willian Marques Freire e Munif Gebara Junior

***Resumo*—Este artigo têm por objetivo a apresentação do conceito Micro-serviço, e provar que é possível desenvolver uma estrutura, que vise a alta coesão e o baixo acoplamento que é empregado quando se trata de micro-serviços. Dentre as justificativas para o desenvolvimento deste artigo, está o fato de que, o micro-serviço além de privilegiar o desenvolvimento de aplicações de acordo com o que foi citado anteriormente, o mesmo é muito utilizado quando se trata de aplicações distribuidas. Aplicações com escopos bem definidas, coreogradas, e específicas, têm surgido como o objetivo de facilitar o desenvolvimento de sistemas. Neste artigo, é desenvolvido um exemplo desta arquitetura para prova de conceito, e são feitos diversos testes de tecnologias atuais. A tecnologia principal utilizada é o Eureka da empresa Netflix. Todo este trabalho será desenvolvido durante este artigo, e ao final será apresentado os resultados do mesmo.**

***Palavras-chave*—Microservice, Micro-serviço, Eureka.**

I. Introdução

MICRO SERVIÇOS é um novo paradigma que influecia diretamente o modo em que são desenvolvidas, e distribuídas as aplicações. Há certas características relacionadas à sua organização, à capacidade de negócio independentes, ao *deploy* automatizado, à inteligência e controle descentralizado de liguagens e de dados [4]. Em uma publicação feita por Sampaio [3], o mesmo definiu através de estudos que Micro-serviços são componentes de alta coesão, baixo acoplamente, autônomos e independentes, que representa um contexto de negócio de uma aplicação.

Para exemplificação sobre a motivação do uso de micro-serviços, pode-se citar os sistemas ERP (*Enterprise Resource Planning* ou sistemas para Planejamentos de Recursos Empresariais), que são desenvolvidos para cuidar de setores empresariais, desde o financeiro, recursos humanos, produção, estoque, dentre outros. Em um sistema para Planejamento de Recursos Empresariais, todas as funcionalidades do mesmo são agrupadas dentro deste grande sistema, fazendo com que seja uma aplicação monolítica, ou seja, uma aplicação feita em somente uma unidade.

Por consequência, um dos principais pontos negativos, é que se houver um grande ponto de falha em algum módulo do sistema, isto poderá afetar a aplicação inteira, incluindo funcionalidades não relacionadas com o mesmo. Outro ponto negativo, é a base de código fonte, que se torna exponencialmente extensa de acordo com o tempo de desenvolvimento, tornando assim, novos membros do projeto improdutivo durante algum tempo, já que a complexida do código é bem maior [5].

Um fato que ocorreu no ano de 2014, foi que o Docker (*container* portátil padronizado) começou a ser utilizado amplamente pela comunidade de desenvolvedores. Uma razão importante para sua utilização generalizada que Adrian (Membro e fundados da eBays Research Labs) observa, é sua portabilidade e aumento na velocidade com *container* de aplicações, que entregava algo em minutos ou horas e passou a entregar em segundos [6]. Na figura 1 é apresentado sua utilização entre os anos 2012 e 2016.

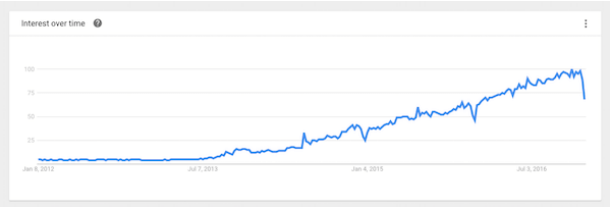


Figura 1. Gráfico de utilzação do docker entre os anos 2012 e 2016.

Semelhantemente, a velocidade de desenvolvimento e implantação de um micro-serviço, permite e incentiva a implementação e estudo dos mesmos. Segundo Adrian micro-serviços possuem também outras características em comum, como implantação com pouca frequência, novas versões implantadas automaticamentes, orquestração de uso geral não é necessário (uma vez que, sistemas inteiros são implantados com todas as partes ao mesmo tempo), arquiteturas utilizam centenas de micro-serviços, e cada publicação, é altamente customizada [7].

Sendo assim, implantar a Arquitetura de Micro-serviços em empresas, irá proporcionar diferentes benefícios para a estrutura de negócio como, usufruir de liberdade maior para o desenvolvimento de serviços de modo independente, implantar aplicações automaticamente através de ferramentas de integração contínua (como Hudson, Jenkins e outras), possibilitar utilização de códigos escritos em linguagens diferentes para diferentes serviços, facilitar a ampliação e integração de micro-serviços com serviços terceirizados (atráves de APIs), organizar o código em função de capacidades de negócio, dando assim, mais visão das ofertas e necessidades dos clientes. Dentre todos os benefícios citados, também é possível fazer o gerenciamento de falhas, visto que, o particionamento favorece uma visão mais detalhada de cada serviço, o que significa que se um serviço venha a falhar, os outros continuarão funcionando [8].

Este trabalho têm por objetivo, preencher a lacuna a respeito do desenvolvimento de uma simples estrutura para micro-serviços, fazendo com que, utilizando-se de ferramentas existentes, seja fácil a criação da mesma. Durante este trabalho, serão criados micro-serviços, que posteriormente abrirão possibilidades para integração entre eles e outros serviços existentes. O mesmo está organizado entre uma revisão biliográfica sobre

Faculdade de Filosofia Ciências e Letras de Mandaguari é uma fundação situada em Mandaguari no Paraná região sul brasileira, na rua Rene Taccola, 152 - Centro Site: (see http://www.fafiman.br/index.html).

Artigo realizado em 2017.

o assunto, uma parte com desenvolvimento dos micro-serviços, e a apresentação de resultados pouco antes da conclusão.

A. Revisão bibliográfica

Segundo dados de Chris Richardson [9], diversas empresas estão utilizando micro-serviços, dentre as citadas estão: Comcast Cable, Uber, Netflix, Amazon, Ebay, SoundCloud, Karma, Groupon, Hailo, Gilt, Zalando, Lending Club, AutoScout24. Os problemas associados ao desenvolvimento de software em larga escala ocorreram em torno da década de 1960. Posteriormente, na década de 1970 viu-se um enorme aumento de interesse da comunidade de pesquisa, para o design de software em suas aplicações e no processo de desenvolvimento. Nesta década, o design foi muitas vezes considerado como uma atividade não associada com a implementação em si, e portanto requerendo um conjunto especial de notações e ferramentas. Por volta da década de 1980, a integração do design nos processos de desenvolvimento contribuiu para uma fusão parcial dessas duas atividades, tornando assim mais difícil fazer distinções puras.

Da mesma forma, as referências ao conceito de arquitetura de software também começaram a aparecer década de 1980. No entanto, uma base sólida sobre o tema foi estabelecida apenas em 1992, por Perry Wolf (autor do livro *"Foundations for the study of software architecture"*). Sua definição de arquitetura de software era distinta do design de software, e desde então tem-se gerado uma grande comunidade de pesquisadores estudando as aplicações práticas da arquitetura de software com base em micro-serviços, permitindo dddassim que os conceitos sejam amplamente adotados pela indústria e pela academia.

O advento e a difusão da orientação a objetos, a partir dos anos 80 e, em particular, a década de 1990, trouxe sua própria contribuição para o campo da Arquitetura de Software. O clássico por Gamma et ai. abrange a concepção de software orientado a objetos e como traduzi-lo em código que apresenta uma coleção de soluções recorrentes, chamados padrões. Esta ideia não é nova nem exclusiva à Engenharia de Software, mas o livro é o primeiro compêndio a popularizar a idéia em grande escala. Na era pré-Gamma, os padrões para soluções orientada a objetos já estavam sendo utilizado. Um exemplo típico de um padrão de projeto arquitetônico em programação orientada a objetos, é o *Model-View-Controller* (MVC), que tem sido um dos insights seminais no desenvolvimento precoce de interfaces gráficas de usuário [17].

Cerca de sete anos atrás a empresa Netflix (provedora global de filmes e séries de televisão via streaming - distribuição de dados, geralmente de multimídia em uma rede através de pacotes) começou a migrar suas aplicações legadas para uma arquitetura baseada em APIs, (Interface de programação de aplicativos) hospedadas na nuvem (local para armazenamento de dados online) da Amazon (empresa transnacional de comércio electrónico dos Estados Unidos com sede em Seattle), influenciando assim, o crescimento de uma ideologia na área de desenvolvimento de softwares que foi batizada pelo nome de "micro-serviço".

Pouco antes, uma investigação realizada pela empresa Cisco (Companhia sediada em San José, Califórnia, Estados Unidos da América) em 2016 revela que, apesar de toda a euforia sobre a Internet das Coisas, o consumo de de vídeo via internet gera 63% do tráfego global. A expectativa é que essa marca chegue a 79% até 2020 e o tráfego de dados gerado por vídeos em resolução Ultra HD subirá de 1.6% para 20.7% do total em 2020. Um levantamento realizado pela Cisco VNI *Mobile* em 2016 também mostra que os dispositivos IoT mais simples, geram uma quantidade de dados equivalentes a 7 vezes o que é produzido por um celular comum (não um smartphone). Demandando pouco das redes de telecomunicações, os dispositivos IoT não representarão um grande pessoa para os provedores de infraestrutura na América Latina [10].

A fim de informação sobre a utlização de *internet* nos últimos anos, segundo o relatório "The State of Internet" de 2016, da Akamai (Empresa de *Internet* americana, sediada em Cambridge, Massachusetts), o país melhor colocado na faixa de redes com banda igual ou maior a 15 Mb/s é o Chile - 4,4% de seus serviços de Internet atingem essa marca. Entretanto, para chegar a essa posição, o Chile investiu pesadamente entre 2014 e 2015, conseguindo crescer 150% de um ano para outro. O Uruguai fica logo abaixo, com 4,1% de sua Internet na faixa dos 15 Mb/s. Atualmente no Brasil, somente 1,1% dos serviços atingem esta marca.

Na arquitetura de micro-serviços, quando é feito a implantação ou atualização dos mesmos, não é afetado outros serviços, e cada micro-serviço pode ser desenvolvido em uma linguagem diferente. Governança granular também é possível para cada micro-serviço, já que o mesmo não tem dependência com outros, e pode ser monitorado e coreografado separadamente. Essa arquitetura descentraliza o gerenciamento de dados, uma vez que cada micro-serviço pode armazenar seus dados de uma maneira que se adapte a ele, e são mais resistentes do que as aplicações tradicionais, devido ao fato de que uma única aplicação pode ser retirada de um monte de aplicações, considerando que são independentes uns dos outros.

1) Tecnologias para gerenciamento de Micro-serviços: Neste trabalho foi utilizado diversas tecnologias para desenvolvimento dos micro-serviços. A principal utilizada que é utilizado para registro centralizado das aplicações, é a tecnologia da Netflix *Service Discovery (Eureka)*. Existem outras bibliotecas que podem trabalhar em conjunto com o Eureka, e serão utilizadas neste trabalho, algumas são: Circuit Breaker (Hystrix), Intelligent Routing (Zuul) and Client Side Load Balancing (Ribbon).

2) Zuul: Zuul é a "porta da frente" para todas as requisições de dispositivos e sites para o *back-end*. O mesmo foi construído para permitir roteamento dinâmico, monitoramento, resiliência e segurança. O Zuul foi desenvolvido pela Netflix pelo fato de que, o volume e a diversidade do tráfego da API do mesmo, resultam em problemas de produção, surgem rapidamente e sem aviso prévio, e a empresa necessitava de um sistema que permita os mesmos mudarem rapidamente o comportamento e reagir a estas situações. O Zuul utiliza diferentes tipos de filtros, que permitem aplicar rapidamente funcionalidades aos serviços de ponta.

Em suma, esses filtros ajudam a executar as seguintes funções: Autenticação e segurança, identificação de requisitos

de autenticação para cada recurso, negação de solicitações indesejadas, insights e monitoramento, rastreamento de dados significativos e estatísticas, a fim de dar uma visão precisa da produção, roteamento dinâmico, encaminhamento dinâmico solicitações para diferentes clusters de backend conforme necessário, *stress Testing*, aumento gradual de tráfego para um cluster, a fim de avaliar o desempenho, *load Shedding*, alocação de capacidade para cada tipo de solicitação e liberação de pedidos que excedem o limite, manipulação de resposta estática e construção de respostas diretamente na ponta ao invés de encaminhá-las para um cluster interno. Dentre os vários componentes que integram a biblioteca do Zuul, estão: Zuul-core que contém funcionalidades a fim de compilar e executar filtros, Zuul-simple que demonstra como construir um aplicativo com zuul-core e Zuul-netflix que adiciona componentes Netflix utilizando Ribbon para solicitações de roteamento [13].

3) Ribbon: Ribbon oferece suporte à comunicação entre processos na nuvem, e inclui balanceadores de carga desenvolvidos pela netflix. A tecnologia citada fornece os seguintes recursos: regras de balanceamento de carga múltiplas e conectáveis, integração com a descoberta de serviços, resiliência de falhas incorporada e clientes integrados com balanceadores de carga. O Ribbon é composto pelos seguintes projetos: *Ribbon-core* que inclui definições de interface e balanceamento de carga e cliente, implementações de balanceador de carga comuns, integração de cliente com balanceadores de carga e fábrica de clientes. *Ribbon-eureka* que inclui implementações do balanceador de carga com base no *Eureka-client* (biblioteca para registro e descoberta de serviços). *Ribbon-httpclient* que inclui a implementação de balanceamento de carga baseada em JSR-311 [12].

4) Hystrix: Em um ambiente distribuído, inevitavelmente algumas das muitas dependências de serviços falharão, e esta biblioteca ajuda a controlar as interações entre serviços distribuídos, adicionando tolerância de latência e lógica de tolerância a falhas. O mesmo faz isso isolando pontos de acesso entre os serviços, interrompendo falhas em cascata através deles, todas as quais melhoram a resiliência geral do sistema. Atualmente, dezenas de bilhões de threads isoladas e centenas de bilhões não isoladas, são executadas utilizando o Hystrix todos os dias na Netflix. Isso resulta em uma melhoria dramática no tempo, atividade e resiliência das aplicações. Hystrix é um projeto desenvolvido também para proteger e controlar a latência e falhas, de dependências acessadas por meio de bibliotecas de terceiros, monitoramento em tempo real, alertas e controle operacional. Quando se trata de micro-serviços, os mesmos contém dezenas de dependências com outros serviços, o que ocasiona que se um deles falhar, e o mesmo não estiver isolado destas falhas externas, corre o risco de também ser afetado. Como exemplo, um aplicativo que dependa de 40 serviços, em que cada serviço tem 99,99% de disponibilidade, pode se esperar: $99{,}99^{40} = 99{,}6\%$ de tempo de atividade, 0.4% de 1 bilhão de falhas resulta em 4 milhões de falhas. Mesmo que pequena a possibilidade de falha, se somar a quantidade de micro-serviços ao tempo de indisponibilidade que pode surgir por pequenas falhas, o problema pode ser facilmente escalável fazendo com que assim serviços importantes fiquem até mesmos horas indisponíveis. Quando toda a aplicação está funcionando e configurada de maneira correta, o fluxo de solicitações ocorrer conforme a figura 2.

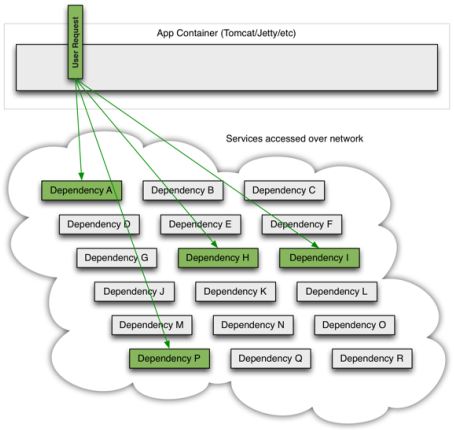


Figura 2. Wiki Hystrix (Internet Overtime) - 2015).

Quando um dos muitos serviços se torna latente, ele pode bloquear toda a solicitação do usuário, conforme apresentado na figura 3.

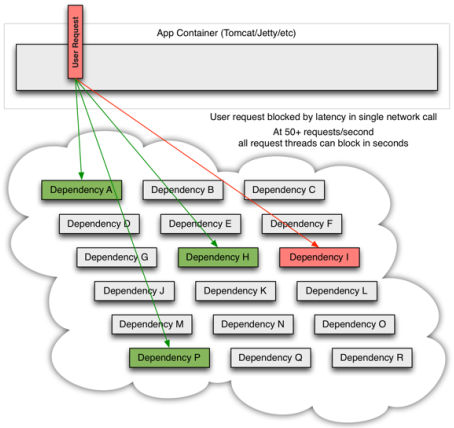


Figura 3. Wiki Hystrix (3 dimensions to scaling) - 2015).

Com tráfego de alto volume, uma única dependência com latência excessiva, pode fazer com que todos os recursos fiquem saturados em segundos. Cada ponto em um aplicativo que atinge a rede, ou em uma biblioteca cliente que pode resultar em solicitações de rede, é uma fonte de falha potencial. Esses aplicativos também podem resultar em latências entre os serviços, causando ainda mais falhas em cascata em todo a aplicação, conforme a figura 4.

A biblioteca Hystrix subjaz os seguintes princípios de design: impedir que qualquer dependência única utilize todas as threads de usuários de um container (como Tomcat) desperdiçando carga do sistema, fornecer soluções sempre que possível para proteger os usuários contra falhas, utilizar técnicas de isolamento para limitar o impacto de qualquer

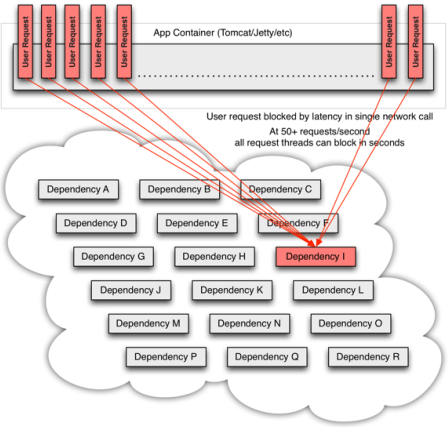


Figura 4. Wiki Hystrix (Container) - 2015).

dependência, otimizar o tempo de descoberta através de métricas, monitoramento e alertas em tempo real, otimizar o tempo de recuperação por meio de propagação de baixa latência de alterações de configuração e, oferecer suporte para alterações de propriedade dinâmicas na maioria dos aspectos do Hystrix, o que permite fazer modificações operacionais em tempo de execução com loops de realimentação de baixa latência, protegendo contra falhas em toda a execução do cliente, não apenas no tráfego da rede. (Hystrix, 2015)

5) Eureka + Spring Cloud: Spring Cloud fornece integrações Netflix OSS para Spring Boot por meio de autoconfiguração, e vinculação ao Spring Environment e outros padrões de programação Spring. Dentre os produtos Spring Cloud encontra-se, os clientes Eureka ou Service Discovery, que é um dos princípios fundamentais de uma arquitetura baseada em micro-serviços. Configurar um micro-serviço é trabalhoso, pois envolve diversas técnicas de descoberta e registro de serviços, e com o Service Discovery da Netflix, torna-se eficiente este trabalho pois com poucas anotações Java consegue-se criar uma aplicação simples Eureka.

Eureka também vem com um componente de cliente baseado em Java, o cliente Eureka, que torna as interações com o serviço muito mais fácil. O cliente também tem um balanceador de carga incorporado, que faz balanceamento de carga *round-robin* (Algoritmos simples de agendamento e escalonamento de processos) básico. Quando um cliente se registra no Eureka, o mesmo fornece metadados como host e porta, dentre outras informações que podem ser encontradas na documentação. Se o registro falhar durante a configuração, a instância da aplicação é removida do registro. Em resumo, o Eureka é um serviço baseado em REST (Representational State Transfer), que é utilizado principalmente na AWS *(Amazon Web Services)*, para localizar serviços com a finalidade de balanceamento de carga, e *failover* (tolerância a falhas) de servidores de camada intermediária.

Semelhantemente, a Amazon possui um produto chamado AWS ELB *(Amazon Web Services Elastic Load Balancer)*, que é uma solução de balanceamento de carga para serviços de ponta, expostos ao tráfego web do usuário final, e a diferença entre o mesmo e o produto da Netflix, é que o Eureka preenche a necessidade de balanceamento de carga médio. Embora teoricamente pode-se colocar serviços de nível intermediário junto com o AWS ELB, no EC2 *classic (Elastic Compute Cloud)*, pode-se expor à rede externa, e perder toda a utilidade dos grupos de segurança AWS. O AWS ELB também possui uma solução de balanceamento de carga em proxy (servidor intermediário para requisições entre cliente e servidor final) tradicional, enquanto no Eureka, o balanceamento ocorre no nível da instância, servidor e host. As instâncias do cliente sabem todas as informações sobre quais aplicações precisam conversar.

Na Netflix, além de desempenhar um papel crítico no balanceamento de carga de nível médio, o Eureka é utilizado para os seguintes fins: implementações com Netflix Asgard, um serviço para fazer atualizações de serviços de forma rápida e segura, registro e exclusão de instâncias e transporte de metadados específicos de aplicativos adicionais sobre serviços. Dentre os motivos para utilizar o Eureka está o fato de que, o mesmo provê uma solução para balanceamento de carga round-robin simples, e quando não pode-se expor o tráfego das aplicações externamente com o AWS ELB, o Eureka resolve este problema.

Sendo assim, com o Eureka a comunicação é transparente, pois o mesmo fornece informações sobre os serviços desejados para comunicação, mas não impõe quaisquer restrições sobre o protocolo ou método de comunicação. Exemplificando, pode-se utilizar o Eureka para obter o endereço do servidor destino e utilizar protocolos como thrift, http(s) ou qualquer outro mecanismos RPC *(Remote Procedure Call)* que permite fazer conexões ou chamadas por espaço de endereçamento de rede.

6) Modelo Arquitetural Eureka: O modelo arquitetural implantado na Netflix utilizando o Eureka é descrita na figura 5. Existe um cluster por região que conhece somente instâncias de sua região. Há pelo menos um servidor Eureka por zona para lidar com falhas da mesma. Os serviços se registram e, em seguida, a cada 30 segundos enviam os chamados ”batimentos cardíacos” ou requisições para renovar seus registros. Se o cliente não renovar o registro, ele é retirado do servidor em cerca de 90 segundos. As informações de registro e renovações são replicadas para todos as conexões no cluster. Os clientes de qualquer zona podem procurar as informações do registro para localizar seus serviços que podem estar em qualquer zona e fazer chamadas remotas.

Para serviços não baseados em Java, têm-se a opção de implementar a parte do cliente, utilizando o protocolo REST desenvolvido para o Eureka ou executar um *”side car”* que é uma aplicação Java com um cliente embutido Eureka, que manipula os registros e conexões. Quando se trabalha com serviços em nuvem, pensar em resiliência se torna ímprobo. Eureka se beneficia dessa experiência adquirida, e é construído para lidar com falha de um ou mais servidores do mesmo.

B. Desenvolvimento

C. Iniciando com Eureka Client

Este trabalho tem por objetivo o a pesquisa e desenvolvimento de uma estrutura de micro-serviços. Como o projeto

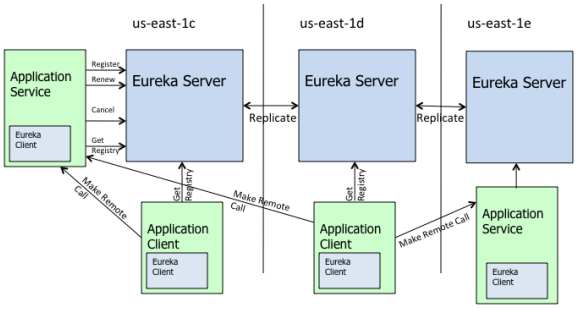


Figura 5. Wiki Eureka - 2015).

será baseado em Java, inicialmente será criado um projeto que utilizará maven, para gerenciamento de dependências, e o Eureka *Client* para descoberta de serviços. Após a criação de um projeto utilizando Maven, será incluso o groupId org.springframework.cloud e o artifactId spring-cloud-starter-eureka no arquivos de configuração das dependências.

Como resultado, quando um cliente se registra com o Eureka, ele fornece meta-dados sobre si, indicador de estado ou saúde, página inicial, dentre outros. O Eureka recebe mensagens heartbeat (disponibilidade) de cada instância pertencente a um serviço, e se algum heartbeat falhar, a instância é removido do registro. Para inicializar um projeto com Eureka Client, será utilizado algumas anotações Java fornecidas pelo Eureka descritas a seguir: *@Configuration*, para utilizar recursos do projeto Spring Config, para facilitar configurações de projetos Spring baseado em Java, *@ComponentScan* para buscar componentes em pacotes java, *@EnableAutoConfiguration* para ativar as configurações automáticas internas, *@EnableEurekaClient* para ativar a descoberta de serviços do Eureka, *@RestController* para criar um controlador Rest *(Representational State Transfer)*, *@RequestMapping* para mapear as rotas da aplicação. As mesmas podem ser visualizadas na figura 6.

```
@Configuration
@ComponentScan
@EnableAutoConfiguration
@EnableEurekaClient
@RestController
public class Application {

    @RequestMapping("/")
    public String home() {
        return "Hello world";
    }

    public static void main(String[] args) {
        new SpringApplicationBuilder
          (Application.class).web(true)
          .run(args);
    }
}
```

Figura 6. Inicio de utilização do Eureka

Para que possa surtir efeito na aplicação, é necessário fazer ajustes nas configurações do Eureka, dentro do diretório resources da aplicação Java. Esta configuração é feita dentro de um arquivo *Application.yml* conforme a figura 7.

```
Eureka
 cliente:
   ServiceUrl:
     DefaultZone:
       http://localhost:8761/eureka/
```

Figura 7. Configuração do Eureka

Neste arquivo de configuração, encontra-se uma peculiaridade. O DefaultZone é a URL do serviço Eureka para qualquer cliente. O nome do aplicativo padrão (ID de serviço), o host e a porta podem ser acessadas respectivamente pelas variáveis de ambientes: *$spring.application.name* , *$spring.application.name* e *$server.port*. A anotação Java *@EnableEurekaClient* faz com que, a aplicação corrente se registre no Eureka, para que assim possa localizar outros serviços.

D. Status e Saúde do serviço

Com a página de status e os indicadores de integridade de uma instância do Eureka, é possível visualizar informações do serviço. Para acessar os indicadores de saúde, deve-se configurar as rotas padrões de acesso a mesma. Por padrão, o eureka utiliza a conexão do cliente, para determinar se está ativo. Caso não seja utilizado o Discovery Client, não será propagado o status de verificação de integridade atual do serviço. Para que os indicadores de saúde e status da aplicação funcionem corretamente, devem ser feito configurações conforme a figura 8.

```
eureka:
  instance:
    statusPageUrlPath:
      ${management.context-path}/info
    healthCheckUrlPath:
      ${management.context-path}/health
  client:
    healthcheck:
      enabled: true
```

Figura 8. Configuração de indicadores de Status

Para conseguir utilizar mais recursos e obter mais informações sobre o status da aplicação, a aplicação deve implementar seu próprio controle de integridade que se encontra no pacote *com.netflix.appinfo.HealthCheckHandler*

E. Alterando o ID da instância Eureka

Uma instância registrada no Eureka possui seu ID, que identifica o serviço que está no mesmo. O Spring Cloud Eureka fornece o seguinte padrão de configuração: *${spring.cloud.client.hostname}:${spring.application.name}*: ${spring.application.instance_id:${server.port}}. Como exemplo a URL fica da seguinte maneira: myhost:myapp:8080

F. Iniciando com EurekaClient

O próximo passo para utilizar o Eureka Server para coreografar os micro-serviços, é utlizar o Eureka Client, que pode ser utilizado para descobrir instâncias do mesmo. Para fazer isto utilizando o framework, primeiramente é necessário incluir a dependência do EurekaClient, e criar um método que busque as instâncias registradas no Eureka conforme a 9.

```
@Autowired
private EurekaClient discoveryClient;

public String serviceUrl() {
  InstanceInfo instance =
    discoveryClient
    .getNextServerFromEureka
      ("STORES", false);
    return instance.getHomePageUrl();
}
```

Figura 9. Busca de instâncias

Não necessariamente é preciso utilizar o EurekaClient. Também pode-se utilizar o DiscoveryClient. A diferença entre os dois, está na maneira de como é utilizado. A mesma pode ser vista na figura 10.

```
@Autowired
private DiscoveryClient discoveryClient;

public String serviceUrl() {
    List<ServiceInstance> list =
      discoveryClient.getInstances
        ("STORES");
    if (list != null && list.size() > 0 ) {
        return list.get(0).getUri();
    }
    return null;
}
```

Figura 10. Busca de instâncias com DiscoveryClient

G. Performanece de registro no Eureka

Registrar um serviço no Eureka pode ser considerado um pouco lento, pelo fato de que, ser uma instância também envolve um heartbeat periódico para o registro, com duração padrão de 30 segundos. Um serviço não estará disponível para descoberta por clientes, enquanto uma intância tenha todos os metadados em seu cache local. Para alterar o período em que isto ocorre, pode ser configurado através da propriedade *eureka.instance.leaseRenewalIntervalInSeconds*. Entretanto, em produção, não deve ser alterado este padrão pelo fato de que, existem alguns cálculos internos do Eureka, que fazem suposições de renovação de locação.

H. Zonas Eureka

Primeiramente, para se configurar uma zona Eureka, é necessário ter certeza de que existem servidores Eureka implantados em cada zona e que eles são pares uns dos outros. Em seguida, precisa-se informar em qual zona o mesmo está. Para fazer isto será utilizado a propriedade *metadataMap*. E isto pode ser feito conforme a figura 11.

```
eureka.instance.metadataMap.zone = zone1
eureka.client.preferSameZoneEureka = true
```

Figura 11. Compilação do firmware

I. Primeiros passos com Eureka Server

Como este projeto é baseado em Java no backend, e utiliza maven como gerenciador de dependências, será incluso nas configurações de dependências Maven o groupId *org.springframework.cloud* e o artifactId *spring-cloud-starter-eureka-server*. Com esta dependência adicionada, será possível utilzar o Eureka Server.

Após adicionar esta dependência, será criado a classe principal, que se encarregará de iniciar a aplicação Eureka Server. Utilizando-se da anotação *@EnableEurekaServer* fornecida pelo framework, e seguindo o padrão utilizado no Eureka Client para iniciar a aplicação, é possível ver um resultado. Um exemplo de código pode ser na figura 12.

```
@SpringBootApplication
@EnableEurekaServer
public class Application {
  public static void main(String[] args) {
      new SpringApplicationBuilder
        (Application.class).web(true)
        .run(args);
  }
}
```

Figura 12. Código inicial Eureka Server

J. Modo Autônomo

A combinação entre o cliente e servidor Eureka, e as pulsações para verificação de disponibilidade entre os mesmos, tornam o servidor Eureka resiliente à falhas, contanto que haja algum tipo de monitoramento para mantê-lo funcionando. No modo autônomo, pode-se prefirir desativar o comportamento padrão do lado do cliente, para que ele não continue tentado alcançar seus pares caso haja falha. Para isto será feito diversas configurações como pode ser visto na figura 13.

```
server:
  port: 8761

eureka:
  instance:
    hostname: localhost
  client:
    registerWithEureka: false
    fetchRegistry: false
    serviceUrl:
      defaultZone:
        http://${eureka.instance.hostname}
        :${server.port}/eureka/
```

Figura 13. Confiugração Eureka Server

Acima foi apresentado as seguintes configurações: port que configura a porta em que será instalado a aplicação, hostname

para identificar o nome do host, *registerWithEureka* que indica se a própria aplicação do Eureka se registrará em si mesma, e fetchRegistry que diz se o Eureka buscará registros associados a eles que ainda estão executando, porém ainda não registradas no Eureka. Com o Eureka, os registros podem ser ainda mais resistentes e disponíveis, executando várias instâncias e pedindo-lhes para se registrarem uns com os outros. Tudo o que precisa para fazê-lo é configurar o serviceUrl dos pares, conforme a figura 14.

```
---
spring:
  profiles: peer1
eureka:
  instance:
    hostname: peer1
  client:
    serviceUrl:
      defaultZone: http://peer2/eureka/

---
spring:
  profiles: peer2
eureka:
  instance:
    hostname: peer2
  client:
    serviceUrl:
      defaultZone: http://peer1/eureka/
```

Figura 14. Configuração de perfis

Neste exemplo, o serviço pode ser utilizado para executar o mesmo servidor em 2 hosts (peer1 e peer2), executando-o em diferentes perfis Spring. É possível utilizar esta configuração para testar a descoberta dos pares em um único host, manipulando neste caso em servidor Linux, o arquivo hosts para resolver os nomes de host que pode ser encontrado dentro do diretório */etc/hosts*.

Em alguns casos, é preferível que o Eureka utilize os endereços IP dos serviços, ao invés do nome do host. Para isto deve ser definido a configuração *eureka.instance.preferIpAddress* como true, e quando a aplicação se registrar com o Eureka, o mesmo utilizará seu endereço IP ao invés do nome de host.

K. Clientes Hystrix

Segundo Martin Flower Et. Al [11], é comum que os sistemas de software façam chamadas remotas para software em execução em diferentes processos, provavelmente em máquinas diferentes em uma rede. Uma das grandes diferenças entre chamadas em memória e chamadas remotas é que, chamadas remotas podem falhar ou travar sem uma resposta, até que algum limite de tempo limite seja atingido. Devido a diversos problemas que podem ocorrer na arquitetura de micro-serviços, a Netflix criou a biblioteca chamada Hystrix, que impelementa o padrão disjuntor que interromple automáticamente o serviço quando ocorre falhas. Uma falha de serviço no nível inferior de serviços pode causar falha em cascata em todo o caminho até o usuário. No Hystrix o padrão de limite de falhas são 20 em 5 segundos, e quando ocorre o circuito é aberto e a chamada não é feita, isto pode ser visto na figura 15.

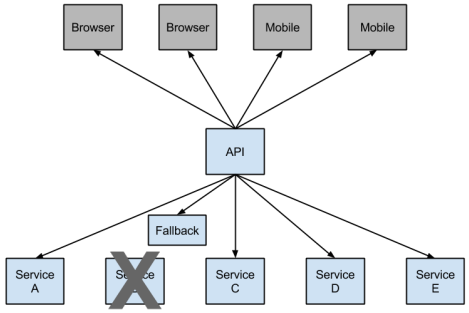


Figura 15. Fallback Hystrix em falhas em cascata.

L. Primeiros passos com Hystrix

Para utilizar o Hystrix, precisa ser incluso a dependência com o groupId org.springframework.cloud e o artifactId *spring-cloud-starter-Hystrix*, e a anotação *@EnableCircuitBreaker* na classe principal, conforme a figura 16.

```
@SpringBootApplication
@EnableCircuitBreaker
public class Application {

    public static void main(String[] args) {
        new SpringApplicationBuilder
          (Application.class).web(true)
          .run(args);
    }

}

@Component
public class StoreIntegration {

    @HystrixCommand
      (fallbackMethod = "defaultStores")
    public Object getStores
      (Map<String, Object> parameters) {
        //do stuff that might fail
    }

    public Object defaultStores
      (Map<String, Object> parameters) {
        return /* something useful */;
    }
}
```

Figura 16. Hystrix

No exemplo de código da figura 16, foi implementando uma classe que se contém um método que buscará os serviços, e para isto foi utilizado a anotação *@HystrixCommand*. É possível ativar as métricas Hystrix e a central de gerenciamento do mesmo adicionando as dependências conforme a figura 17. O endpoint para acesso ao gerenciador é */hystrix.stream*.

M. Ribbon

Ribbon é um balanceador de carga do lado do cliente, que fornece controles sobre o comportamento dos clientes HTTP e TCP. Uma observação importante é que a anotação *@FeignClient* já utiliza Ribbon, fazendo com que assim seja desnecessário a utilização do Ribbon, entretanto, quando for

```
<dependency>
    <groupId>
      org.springframework.boot
    </groupId>
    <artifactId>
      spring-boot-starter-actuator
    </artifactId>
</dependency>
<dependency>
    <groupId>
      org.springframework.cloud
    </groupId>
    <artifactId>
    spring-cloud-starter-hystrix-dashboard
    </artifactId>
</dependency>
```

Figura 17. Dependencias do Gerenciados Hystrix

preciso um controle mais versátil sobre a tecnologia é optável a utilização do mesmo.

Para incluir o ribbon no projeto, será utilizado o mesmo padrão de configurações feito anteriormente. O que será alterado é o artifactId da dependência maven, que agora será utilizado *spring-cloud-starter-ribbon*, e para configurar o cliente Ribbon criado uma classe de configuraçao e será anotado com *@RibbonClient*, conforme a figura 18.

```
@Configuration
@RibbonClient(name = "foo",
  configuration = FooConfiguration.class)
public class TestConfiguration {
}
```

Figura 18. Ribbon

N. FeignClient

FeignClient é uma biblioteca que faz com que clientes de serviços web sejam escritos de forma mais fácil. Para utilizá-lo é preciso instalar a dependência *spring-cloud-starter-feign*, e anotar a classe principal com *@EnableFeignClients* conforme a figura 19. O mesmo provê suporte para anotações Spring MVC, e por utilizar o mesmo conversor de mensagens HTTP que o Spring Web, é integrada com Hystrix para fornecer um cliente com balanceamento de carga.

```
@Configuration
@ComponentScan
@EnableAutoConfiguration
@EnableEurekaClient
@EnableFeignClients
public class Application {
  public static void main(String[] args) {
      SpringApplication
        .run(Application.class, args);
  }
}
```

Figura 19. FeignClient

Ao anotar uma interface com *@FeignClient* conforme a figura 20, pode ser mapeado os métodos para que consiga acesso a endpoints da biblioteca. O nome do método será qualificado e aplicado ao contexto da aplicação, fazendo com que assim não seja implementado corpo ao método pois será apenas repassados chamadas REST.

```
@FeignClient("stores")
public interface StoreClient {
  @RequestMapping(method =
    RequestMethod.GET,
    value = "/stores")
    List<Store> getStores();

    @RequestMapping(method =
      RequestMethod.POST,
      value = "/stores/{storeId}",
      consumes = "application/json")
    Store update(
    @PathVariable("storeId") Long storeId,
    Store store);
}
```

Figura 20. Mapeamento FeignClient

Cada Cliente Feign faz parte de um conjunto de componentes que trabalham juntos, para comunicar-se via HTTP. O Spring Cloud permite com que se tenha controle total sobre clientes *Feign*, declarando uma classe de configuração que implemente determinados métodos do FeignClient conforme a figura 21. Duas das possíveis configuração, são: modificar o padrão de Contrato que o FeignClient utiliza para que assim seja personalizado o padrão de comunicação REST que o mesmo utiliza e modificar o método de autenticação do FeignClient.

```
@Configuration
public class FooConfiguration {
  @Bean
  public Contract feignContract() {
      return new feign.Contract.Default();
  }

  @Bean
  public BasicAuthRequestInterceptor
    basicAuthRequestInterceptor() {
    return new BasicAuthRequestInterceptor
        ("user", "password");
  }
}
```

Figura 21. Autenticação com FeignClient

O. Zuul

Até neste capítulo, foi falado de rotas e endpoints, mas ainda não foi explorado a essencialidade de utilizar rotas. Roteamento é uma parte fundamental de uma arquitetura de microserviços, pelo fato de que, uma uri (identificador uniforme de recursos) pode ser mapeado de acordo com sua utilidade. Como exemplo, uma uri apiusuario é mapeado para o serviço de usuário, apivendas pode ser mapeado para o serviço de vendas de uma loja. Zuul é um roteador com balanceamento de carga básico. A empresa Netflix atualmente utiliza o Zuul para os seguintes desígnios: autenticação, insights teste de

stress da api, *Canary test* que segundo Danilo Sato [15] é uma técnica para reduzir o risco de introduzir uma nova versão de software na produção lentamente lançando a mudança para um pequeno subconjunto de usuário antes de lança-la em toda a infra-estrutura e torná-la disponível para todos, roteamento dinâmico, serviço de migração, divisão de carga, segurança, manipulação de respostas e gestão de tráfego de dados.

Para incluir o Zuul em um projeto utilizando os padrões aplicados até neste capítulo, é necessário utilizar o artifactId *spring-cloud-starter-zuul*, e para habilitá-lo, a classe principal deve ser anotada com *@EnableZuulProxy*, e este encaminhará chamadas locais para o serviço adequado. Para que de acordo com a rota seja chamado o serviço desejado deve ser configurado no arquivo de configurações principais *application.yml*, e para ignorar demais serviços utiliza-se a propriedade zuul.ignored-services. Na figura 22 está um exemplo da utilizaçao da mesma.

```
zuul:
  ignoredServices: '*'
  routes:
  produtos: /meus-produtos/**
```

Figura 22. Zuul

Para obter um controle mais refinado sobre determinadas rotas, pode-se especificar o caminho e o serviceId, conforme a figura 23.

```
zuul:
  routes:
    produtos:
      path: /meus-produtos/**
      serviceId: produtos_service
```

Figura 23. Controle refinado das rotas

Isto significa que quando ocorrer uma chamado para a rota ¨myprodutos¨, o mesmo será encaminhado para o serviço ¨produtos_service¨. Se for desejável especificar uma URL para uma localização física do serviço, pode ser feito conforme a figura 24.

```
zuul:
  routes:
    produtos:
      path: /produtos/**
      url:
      http://exemple.com/produtos_service
```

Figura 24. URL física do serviço

As configurações de rotas feitas até o momento, não são executadas como um HystrixCommand ou balancedas com Ribbon. Para conseguir isto, deve-se especificar um serviço de rota e criar um cliente Ribbon para o serviceId. Na figura 25 contém um exemplo de configuração para o mesmo.

```
zuul:
  routes:
    produtos:
      path: /meus-produtos/**
      serviceId: produtos

ribbon:
  eureka:
    enabled: false

produtos:
  ribbon:
    listOfServers: exemplo.com,faype.com
```

Figura 25. Cliente Ribbon

P. Migração de aplicações

Um padrão comum ao migrar um serviço Web existente, é remover endpoints antigo, e lentamente substitui-los por novas implementações. O proxy Zuul é uma ferramenta extremamente útil para isto, pelo fato de que, pode-se utilizá-lo para lidar com todo o tráfego de clientes antigos, mas lidar com solicitações para novos. Na figura 26 contém um exemplo de configuração.

```
zuul:
routes:
  primeiro:
    path: /primeiro/**
    url: http://primeiro.exemplo.com
  segundo:
    path: /segundo/**
    url: forward:/segundo
  terceiro:
    path: /terceiro/**
    url: forward:/3rd
  legacy:
    path: /**
    url: http://teste.exemplo.com
```

Figura 26. Atualização de Rotas

Ao processar as solicitações de entrada, parâmetros de consulta são decoficados para que os mesmos possam estar disponíveis para possíveis modificações nos filtro Zuul. Estes são recodificados aos reconstruir o pedido backend nos filtros de rota. O resultado pode ser diferente da entrada original, principalmente se o mesmo foi codificando utilizando por exemplo na linguagem Javascript a função *encodeURIComponent()* . Isto não causa problemas na maiorias dos casos, entretanto, algumas aplicações web podem exigir codificação de url para consultas complexas. Para forçar a codificação original da url, é possível utilizar a configuração *zull.forceOriginalQueryStringEncoding* definindo-à como true.

Q. RxJava com Spring MVC

RxJava é uma implementação Java VM de extensões reativas : uma biblioteca para compor programas assíncronos e baseados em eventos utilizando sequências observáveis [19].

Spring Cloud fornece suporte para *observables* que em RxJava é um objeto que implementa a interface Observable

que em seguida, este assinante reage a qualquer item ou sequência de itens que o objeto *Observable* emite. Esse padrão facilita operações simultâneas porque não precisa bloquear enquanto espera que o Observable emita objetos, mas ao invés disto, cria uma sentinela na forma de assinante que está pronto para reagir apropriadamente em qualquer tempo futuro que o Observable gere [19]. Utilizando o Spring Cloud pode-se retornar objetos *rx.Single*, *rx.Observale* e SSE (Eventos enviados pelo servidor) que é uma tecnologia pelo qual um navegador recebe atualizações de um servidor via HTTP. Na figura 27 está dois exemplos, segundo a Netflix [16] de como utilizá-los.

```
@RequestMapping(method =
  RequestMethod.GET, value = "/multiple")
  public Single<List<String>> multiple() {
    return Observable
      .just("multiple", "values")
      .toList().toSingle();
}

@RequestMapping(
    method = RequestMethod.GET,
    value = "/responseWithObservable")
public ResponseEntity<Single<String>>
    responseWithObservable() {

Observable<String> observable =
Observable.just("single value");
HttpHeaders headers = new HttpHeaders();
headers
  .setContentType(APPLICATION_JSON_UTF8);
return new ResponseEntity<>(
      observable.toSingle(),
    headers, HttpStatus.CREATED);
}
```

Figura 27. Compilação do firmware

R. Motivação de uso das tecnologias apresentadas

Até o prezado momento, foram apresentados diversas tecnologias, entretanto, fica a questão sobre o por que de tantas tecnologias. e a resposta é clara e objetiva. Para que se tenham micro-serviços, é necessário a utilização de um recurso para a orquestração dos mesmos, neste caso, entra o Eureka. Quando se tem micro-serviços, é necessário a centralização dos mesmos para que não fique espalhados e perdidos em hosts ou portas diferentes, surge a necessidade de um gateway, neste caso, o Zuul. A partir do momento que são integrados muitos micro-serviços, torna-se difícil o gerenciamento da configuração dos mesmos, surgindo a necessidade de um mecanismo para fácil configuração e integração de aplicações, por isto, foi apresentado o Spring Config. Após tudo isto, surge ainda um problema, o balanceamento de carga, um problema abrangente quando se trabalha com aplicações distribuidas. Surge então, a motivação de utilizar o Ribbon para distribuição de carga. Com tudo isto configurado, ainda não se tem garantia precisa de disponibilidade da aplicação, fazendo com que, se caso ocorrer um problema em um micro-serviço, necessite de uma tecnologia que resolva esta questão, fazendo assim necessário a utilização do Hystrix. A motivação de apresentar tecnologias Spring Cloud e Netflix, como citado na introdução deste trabalho, é o fato da experiência e sucesso por parte dos mesmos em aplicações distribuidas.

II. Resultados

Desenvolver uma estrutura utilizando micro-serviços pode ser trabalhoso, quando se tem muitos serviços para gerenciar se torna um processo complexo. Entretanto, existem diversas ferramentas que auxiliam neste processo. Durante este trabalho, foram feitos diversos testes, utilizando ferramentas da empresa Netflix para auxílio no processo de desenvolvimento de micro-serviços. Foi utilizado ferramentas para registro de micro-serviços, balanceamento de carga e tolerância a falhas. Em todos foi obtido resultados e têm funcionado da forma esperada.

III. Conclusões e trabalhos futuros

Conclui-se portanto, que é possível criar uma estrutura utilizando micro-serviços, e que existem dificuldades ao gerenciar este estrutura, quando existem muitos micro-serviços envolvidos. Mas utilizando-se de ferramentas para auxílio no desenvolvimento desta estrutura, torna-se possível a criaçao da mesma. Quando é utilizado esta estrutura, os sistemas são mais tolerantes a falhas pelo fato de que, se um serviço parar, os demais continuam. Um ponto a ser levantado, é que as aplicações possuem alta coesão e baixo acoplamento, o que facilita no gerenciamento das mesmas.

Referências

[1] Paulo Silveira. Mauricio Balboa Linhares. *T*ecnologias na Netflix. 2017 [Online] Disponível: http://content.blubrry.com/hipsterstech/hipsters_041_netflix.mp3. [Acesso: 25-Abr-2017]

[2] Flavio Silveira. *M*icroserviços - O que é? 2016 [Online] Disponível: http://flaviosilveira.com/2016/microservicos. [Acesso: 06-Mai-2017]

[3] Cleuton Sampaio. *M*icro serviços: O que são e para que servem. 2015 [Online] Disponível: http://www.obomprogramador.com/2015/03/micro-servicos-o-que-sao-e-para-que.html. [Acesso: 28-Fev-2017]

[4] James Lewis. (2016). *M*icrosserviços em poucas palavras. [Online] Disponível: https://www.thoughtworks.com/pt/insights/blog/microservices-nutshell. [Acesso: 20-Mar-2017]

[5] Adriano Almeida. *A*rquitetura de microserviços ou monolítica. 2015 [Online] Disponível: http://blog.caelum.com.br/arquitetura-de-microservicos-ou-monolitica. [Acesso: 22-Mar-2017]

[6] Cristiano Diedrich. *O* que é Docker. 2015 [Online] Disponível: http://www.mundodocker.com.br/o-que-e-docker. [Acesso: 01-Dez-2016]

[7] Jan Stenberg. *O* estado da arte em micro serviços. 2015 [Online] Disponível: https://www.infoq.com/br/news/2015/04/microservices-current-state. [Acesso: 08-Fev-2017]

[8] Ricardo Peloi. *C*omo implantar uma verdadeira Arquitetura de Microserviços na sua empresa. 2016 [Online] Disponível: http://sensedia.com/blog/soa/implantar-arquitetura-de-microservicos. [Acesso: 08-Mar-2017]

[9] Chris Richardson. *W*hois using microservices. 2016 [Online] Disponível: http://microservices.io/articles/whoisusingmicroservices.html. [Acesso: 20-Mai-2017]

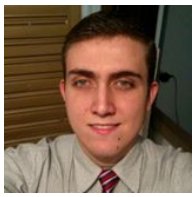
[10] IDC. *C*onnecting the IotT: The road to success. 2016 [Online] Disponível: http://www.idc.com/infographics/IoT. [Acesso: 15-Fev-2017]

[11] Martin Fowler et al. *M*icroservices. 2014 [Online] Disponível: https://www.martinfowler.com/articles/microservices.html. [Acesso: 20-Mai-2017]

[12] Netflix. *R*ibbon. 2016 [Online] Disponível: https://github.com/Netflix/ribbon/wiki. [Acesso: 06-Mai-2017]

[13] Netflix. *Z*uul. 2016 [Online] Disponível: https://github.com/Netflix/zuul/wiki. [Acesso: 02-Mai-2017]

[14] Martin Fowler. *C*ircuitBreaker. 2014 [Online] Disponível: https://martinfowler.com/bliki/CircuitBreaker.html. [Acesso: 02-Mar-2017]
[15] Danilo Sato. *C*anaryRelease. 2014 [Online] Disponível: https://martinfowler.com/bliki/CanaryRelease.html. [Acesso: 10-Abr-2017]
[16] Netflix. *S*pring Cloud Netflix. 2016 [Online] Disponível: http://cloud.spring.io/spring-cloud-netflix/spring-cloud-netflix.html. [Acesso: 18-Dez-2016]
[17] Nicola Dragoni et al. *M*icroservices: yesterday, today, and tomorrow. Technical University of Denmark, vol 1, pp. 1-3, 2016
[18] Willian Marques Freire. Munif Gebara Junior. *A*rtigo IoT - Internet das coisas. vol 1, pp.1-3, 2017
[19] ReactiveX. *O*bservable. 2015 [Online] Disponível: http://reactivex.io/documentation/observable.html. [Acesso: 25-Nov-2016]



Willian Marques Freire Possui Ensino Médio completo pelo Colégio Estadual Rosa Delúcia Calsavara (2013). Atualmente é Desenvolvedor de Software da Gumga Tecnologia da Informação S/A. Tem experiência na área de Ciência da Computação.

Munif Gebara Júnior Possui graduação em Ciência da Computação pela Universidade Estadual de Maringá (1997) e mestrado em Engenharia Elétrica e Informática Industrial pela Universidade Tecnológica Federal do Paraná (2001). Atualmente é professor da Fundação Faculdade de Filosofia Ciências e Letras de Mandaguari e professor de ensino superior da Faculdade de Tecnologia e Ciências do Norte do Paraná Ltda e desenvolvedor.

IoT e Integração com Micro-serviços

Willian Marques Freire e Munif Gebara Junior

***Resumo*—Este trabalho têm por objetivo o desenvolvimento e prova de conceito de uma estrutura que integra micro-serviços e IoT *(Internet of Things)*. Durante dois artigos anteriores a este, foram apresentados estes conceitos, e atualmente neste artigo, será melhorados as duas estrutura que já foram desenvolvidas, para integração dos mesmos. Dentre as justificativas para o desenvolvimento deste trabalho, está o fato de que, os dois termos visam o compartilhamento de informações na rede mundial de internet. Aplicações coesas e independentes são viáveis pelo fato da disponibilidade das mesmas, pois é possível controlar pontos de falhas sem afetas as demais. Sendo assim, obteve-se a idéia de utilizar o micro-serviço e seu conceito para organizar os dispositivos, e integrar-se aos mesmos de forma simples. Este assunto é tratado detalhadamente durante este trabalho, e ao final é comprovado a possibilidade de desenvolvimento do mesmo.**

***Palavras-chave*—IoT, Internet, Coisas, Micro, Serviço, Micro-service, Micro-serviço, Integração.**

I. Introdução

COM A EVOLUÇÃO da computação distribuída surgiu a necessidade da criação de novos paradigmas, dando assim, origem aos Micro-serviços. O termo "Arquitetura de Micro-serviços" surgiu nos últimos anos para descrever uma maneira específica de desenvolver suítes de serviços com implantação *(deploy)* independente. Esta arquitetura tem várias características-chave que reduzem a complexidade de um software. Cada micro-serviço funciona como um processo separado. Consiste em interfaces impulsionadas por dados que normalmente têm menos de quatro entradas e saídas. Cada micro-serviço é auto-suficiente para ser implantado em qualquer lugar em uma rede, pois contém tudo o que é necessário para que ele funcione - bibliotecas, instalações de acesso a banco de dados e arquivos específicos do sistema operacional. Os mesmos são construídos em torno de uma única funcionalidade focada. Portanto, é mais eficaz, sendo que, desenvolvimento, extensibilidade, escalabilidade e integração são os principais benefícios oferecidos pela Arquitetura de Micro-serviços.

Atualmente tem surgido projetos utilizando este formato e os resultados têm sido positivos, tanto que para muitos desenvolvedores o mesmo têm-se tornado a forma padrão de desenvolver aplicações. Utilizando-se da empresa Netflix como referência em micro-serviços, esta provê recursos gratuitos e de código aberto para desenvolvedores. Dentre eles, estão ferramentas como o Eureka, Hystrix, Ribbon entre outros. Estimativas apontam que a mesma faturou algo em torno de R$ 1,1 bilhões somente no Brasil no ano de 2015, e fontes do mercado registraram que o canal de streaming faturou cerca de R$ 260 milhões a mais do que a previsão mais otimista de faturamento do SBT (Sistema Brasileiro de Televisão) neste ano [6]. A empresa Netflix é uma das pioneiras em micro-serviços, e este termo nem sequer existia quando o serviço por streaming da empresa começou a caminhar. Atualmente a plataforma da mesma é sustentada por um *Gateway* (Ponte de Ligação) de APIs que lida com cerca de dois bilhões de requisições todo o dia. No total, as requisições citadas são tratadas por mais de 600 APIs [7].

Atualmente, um assunto também em discussão, que tem chamado a atenção desde pessoas com pouco conhecimento em tecnologia, até pessoas que trabalham na área, é o IoT *(Internet of Things)* ou "Internet das Coisas", que se refere a uma revolução tecnológica que tem como objetivo conectar itens utilizados no dia a dia, à rede mundial de computadores. Cada dia surgem mais eletrodomésticos, meios de transporte, e até mesmo acessórios vestíveis conectados à Internet e a outros dispositivos, como computadores e smartphones [3]. Segundo Ashton (Primeiro especialista a utilizar o termo "Internet das Coisas") [5] a limitação de tempo e da rotina fará com que as pessoas se conectem à Internet de outras maneiras, sendo para tarefas pessoais ou trabalho, permitindo o compartilhamento de informações e experiências existentes na sociedade. Segundo uma pesquisa realizada em 2015 pelo IDC (Corporação Internacional de dados), no mercado de IoT foi movimentado em 2016 cerca de US$ 41 bilhões [9], o que indica o grande investimento por parte dos interessados nesta área.

Ademais, todas as evoluções tecnológicas na área de micro-serviços e IoT tem gerado grande interesse por parte dos desenvolvedores. Com base nas informações apresentadas, observa-se que são duas áreas distintas que crescem exponencialmente em razão do surgimento de novas tecnologias, e têm-se necessidade de verificar relações que podem ser feitas entre as mesmas. Ao construir estruturas de comunicação entre diferentes processos, é visto que, muitos produtos e abordagens enfatizam a inserção de inteligências significativas no próprio mecanismo de comunicação. Um exemplo do que foi citado é o *Enterprise Service Bus* (ESB), onde os os produtos do mesmo incluem recursos sofisticados para roteamento de mensagens, coreografia, transformação e aplicação de regras de negócios [10].

No ano de 1990 estava em alta uso a Arquitetura Orientada a Serviços (SOA). Foi um padrão que incluiu serviço como uma funcionalidade individual. O SOA trouxe muitas vantagens como velocidade, melhores fluxos de trabalho e melhor vida útil das aplicações. Desta vez, foi do ponto de vista da criação de aplicativos desenvolvidos em torno de componentes de domínio de negócios, e que poderiam ser desenvolvidos, manipulados e decompostos em serviços que se comunicassem por meio de APIs, e protocolos de mensagens baseados em

Faculdade de Filosofia Ciências e Letras de Mandaguari é uma fundação situada em Mandaguari no Paraná região sul brasileira, na rua Rene Taccola, 152 - Centro Site: (see http://www.fafiman.br/index.html).

Artigo realizado em 2017.

rede. Este é o ponto a Arquitetura de micro-serviços nasceu. A mesma adiciona agilidade, velocidade e eficiência quando se trata de implantação e modificação de sistemas. Como a tecnologia evolui, especificamente com IoT ganhando tanta tração, as expectativas das plataformas baseadas em nuvem mudaram. *Big Data* é um termo que descreve imenso volumes de dados, e se tornou um lugar comum, e o mundo tecnológico começou a se mover para a economia de APIs. Nesse interim, é onde o clássico SOA começou a mostrar problemas, demonstrando ser muito complicado, com centenas de interfaces e impossível definir granularidade [8].

Sendo assim, os micro-serviços hospedados em nuvem criaram um modelo de coleção de serviços, representando uma função específica. Os mesmos oferecem uma maneira de dimensionar a infra-estrutura tanto horizontal quanto verticalmente, proporcionando benefícios de longo prazo para as implantações de aplicações. Cada um dos serviços pode escalar com base nas necessidades. Dando o dinamismo das expectativas de implantação e escalabilidade que vem com o Micro-serviço, os mesmos precisam se tornar uma parte importante da estratégia IoT [8].

Neste trabalho, tem-se por objetivo o desenvolvimento de uma interação entre as tecnologias citadas, através de uma interface de comunicação simples, onde cada sistema embarcado se comunicará com algum micro-serviço genérico, permitindo assim, a escalabilidade, sustentabilidade e independência dos serviços propostos. Será utilizado tecnologias como Spring Boot (uma plataforma Java criado por Rod Johnson baseado nos padrões de projeto inversão de controle (IoC) e injeção de dependência), o Eureka (uma Interface de comunicação Java para micro-serviços), e a plataforma de prototipagem eletrônica NodeMcu ESP8266, para desenvolvimento do dispositivo IoT que se comunicará com os micro-serviços. Para exemplo de aplicação, pode ser citado um conjunto de dispositivos que iriam coletar informações de sensores e controladores, e torná-los visíveis na forma de dados. Os micro-serviços poderiam apenas processar esses dados e aplicar algumas regras a esses dados. Outros serviços também poderiam buscar dados de sistemas empresariais de terceiros, como sistemas CRM / ERP.

A. Revisão Bibliográfica

1) Hipermídia e Hipertexto: Os conceitos hipermídia e hipertexto foram criados na década de 1960 pelo filósofo e sociólogo estadunidense Ted Nelson. Surgiram utilizando-se da idéia de explorar a metáfora do labirinto, onde o mito e a matemática do labirinto poderiam auxiliar o compreender da realidade multidimensional que os sistemas de hipermídia oferecem. Hipertexto é composto por blocos de informação e por vínculos eletrônicos *(Links)*. Estes blocos são denominados lexias, que é o ponto onde se está antes de seguir um link, que pode ser formados por textos, imagens, vídeos, entre outros. Hipermídia é a reuniao de várias mídias digitais em um ambiente computacional, suportada por sistemas eletrônicos de comunicação, e uma forma basteante comum de hipermídia é o hipertexto, no qual a informação é apresentado na forma de texto interativo [11].

2) HTTP: O HTTP *(Hypertext Transfer Protocol)* ou Protocolo de transferência de hipertexto é um protocolo de comunicação na camada de aplicação segundo o Modelo OSI *(Open System Interconnection)*, utilizado para sistemas de informação de hipermídia, distribuídos e colaborativos. Normalmente, este protocolo utiliza a porta 80 e é bastante utilizado para comunicação de sites [12].

3) Rest: REST *(Representational State Transfer)* ou Transferência de estado representacional, é uma abstração da arquitetura da *World Wide Web* (Web). É um estilo arquitetural que consiste de um conjunto coordenado de restriços arquiteturais aplicadas a componentes, conectores e elementos de dados dentro de um sistema de hipermídia distribuido. Este termo foi apresentado no ano de 2000 por Roy Fielding, um dos principais autores da especificação do protocolo HTTP. [13].

4) Socket - Soquete de rede: Um soquete de rede (em inglês: *network socket*) é um ponto final de um fluxo de comunicação entre processos através de uma rede de computadores. Hoje em dia, a maioria da comunicação entre computadores é baseada no Protocolo de Internet, portanto a maioria dos soquetes de rede são soquetes de Internet. Uma API de soquetes (API *sockets*) é uma interface de programação de aplicativos (API), normalmente fornecida pelo sistema operacional, que permite que os programas de aplicação controlem e usem soquetes de rede. APIs de soquete de Internet geralmente são baseados no padrão Berkeley *sockets*. Um endereço de soquete (socket address) é a combinação de um endereço de IP e um número da porta, muito parecido com o final de uma conexão telefônica que é a combinação de um número de telefone e uma determinada extensão. Com base nesse endereço, soquetes de *internet* entregam pacotes de dados de entrada para o processo ou thread de aplicação apropriado [14].

5) Thread: Thread é uma forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas concorrencialmente. O suporte à *thread* é fornecida pelo sistema operacional ou implementada através de uma biblioteca de uma determinada linguagem. Como exemplo, uma *thread* permite que o usuário de um programa utilize a funcionalidade do ambiente enquanto outras linhas de execução realizem outros cáculos e operações [15].

6) JSON: JSON *(JavaScript Object Notation)* é um formato de padrão aberto que utiliza texto legível a humanos para trasmitir objetos de dados consistema de atributos e valores. Atualmente, é o formato de dados mais utilizado para comunicação assíncrona entre navegadores e sevidores, substituindo o XML *(Extensible Markup Language)*, sendo utilizado pelo AJAX (*Asynchronous Javascript* and XML). JSON é em formato de texto e independente de linguagem de programação, pois utiliza convenções que são familiares a divesas linguagens [16].

7) GPIO: GPIO *(General Purpose Input Output)* ou entrada/saída de uso geral é um pino genérico em um circuito integrado ou placa de computador cujo objetivo é entrar ou sair pulsos elétricos. São utilizados em circuitos integrados com poucos pinos, microfones, gerentes de energia, placas de vídeos, entre outros. Estes pinos podem ser configurados para serem entrada ou saída, e os valores de entrada e saída são legíveis, tipicamente alto ou baixo [17, p. 3].

B. Desenvolvimento

1) IoT: Neste trabalho, como o objeto é desenvolver a integração entre dispositivos IoT e Micro-serviços, primeiramente, foi criado uma estrutura IoT que tenha como objetivo principal a comunicação entre os mesmos e servidores contendo os micro-serviços. No primeiro Artigo escrito por Freire e Gebara [1] foi desenvolvido uma estrutura IoT que permite a comunicação entre dispositivos utilizando o módulos Wi-Fi. É factível o desenvolvimento da comunicação entre estes dispositivos como os micro-serviços através de protocolos como HTTP, utilizando de tecnologias de comunicação entre aplicações distribuidas como o REST.

2) Eureka Server: Antes de começar o desenvolvimento dos micro-serviços, foi necessário ter uma ferramenta que funcionará como gateway entre as aplicações. Para isto foi utilizado o Eureka Server, uma ferramenta REST desenvolvida pela Netflix com principal objetivo de localizar serviços e fornecer balanceamento de carga intermediário [4]. No artigo Micro-serviços de Freire e Gebara [2, p. 6], é explicado detalhadamente o processo de uma configuração simples do Eureka Server para descoberta de serviços.

3) Broadcast no Eureka Server: Um dos objetivos deste trabalho é a fácil configuração de registro dos dispositivos nos micro-serviços. Um dos problemas encontrados, é que o dispositivo não sabe onde está o Eureka Server, e devido a este fato, torna complicado o registro dos dispositivos. Foi analisado algumas estratégias para que o dispositivo descubra onde está o Eureka Server, e nota-se que neste trabalho não há a necessariedade dos dispositivos estar em uma rede diferente do Eureka Server. A solução proposta então, para que os dispositivos consigam se registrar no Eureka sem conhecer o mesmo, é utilizando o Broadcast. É enviado uma mensagem contendo informações como localização do dispositivo via broadcast, e a aplicação Eureka conterá um socket conectado ao mesmo, que quando visualizar uma mensagem, verificará sua origem, e enviará uma requisição a uma determinada rota neste endereço IP (no caso, o dispositivo), e quando o dispositivo receber esta requisição, saberá assim, onde se encontra o Eureka Server. Assim que o dispositivo conter a localização do Eureka Server, ele fará uma requisição a alguma rota do Eureka Server, enviando todas informações necessárias para registro do dispositivo, fazendo com que assim, o dispositivo seja visível pelos demais.

Assim que iniciar a aplicação Eureka Server, conforme a figura 3, é criado uma thread, e nesta thread será instanciado uma classe Java chamada Server, que será um socket conectado ao broadcast. A classe que conterá o código do socket que manterá a conexão via broadcast, pode ser visto na figura 6. Esta classe ficará encarregada de manter a conexão na rede, esperando um dispositivo que busque se registra no Eureka Server.

4) Registro do dispositivo: Conforme detalhado na figura **??**, a partir do momento em que o serviço Eureka Server visualizar uma mensagem no broadcast, o mesmo enviará um requisição para o dispositivo IoT em determinada rota. Com isto, o dispositivo IoT (neste caso o NodeMCU ESP8266), terá o endereço IP do Eureka Server, e com isto, será feito uma requisição para o mesmo, fazendo o registro do dispositivo. Para isto, como o micro-serviço utiliza comunicação REST via HTTP utilizando JSON, o dispositivo também deve utilizar o mesmo padrão de comunicação. No artigo IoT - A Internet das coisas [1], já foi desenvolvido uma estrutura, entretanto, para integração do dispositivo com um micro-serviço, serão feitos algumas implementações adicionais. Será implementado um módulo chamado register.lua, onde conterá 3 funções. A primeira será para tentar o registro no Eureka, a segunda para enviar a requisição de registro com as informações necessárias, e a terceira, conterá um simples servidor, que exibirá as entradas e saídas digitais, podendo ativá-las ou desativá-las.

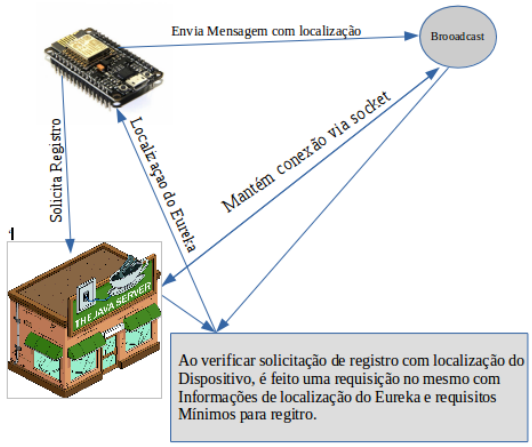


Figura 1. Comunicação entre o dispositivo e o EurekaServer [18].

A figura 2 demonstra a primeira função. Este função foi dividida em três funções menores. Uma que registrará o servidor da aplicação, outra que aguardará uma requisição do Eureka Server, e outra que fará a tentativa de registro.

```
1 function startEureka()
2   registerServer()
3   waitRegister()
4   tryingToRegister()
5 end
```

Figura 2. Função de tentativa de registro.

Assim que o dispositivo receber a requisição do Eureka Server, será feito uma verificação dos parâmetros, em busca do parâmetro ip, onde conterá o endereço do servidor. Isto pode ser visualizado na figura 4. Após o NodeMCU ESP82666 saber onde se encontra o gateway de serviços, será feito o registro no mesmo, e finalizado a conexão do dispositivo no endereço broadcast, conforme a figura 5. A função que contém o protocolo para registra no Eureka Server pode ser visualizada na figura 7, e a função que criará um pequeno servidor para controle dos pinos digitais pode ser visto na figura 8.

II. Resultados

Neste trabalho foi feito a integração entre um dispositivo IoT com módulo WiFi e um micro-serviço. Foi utilizado *Broadcast* e *Socket*, para que o EurekaServer observasse a

rede, para que quando recebesse uma mensagem de algum dispositivo, fosse feito a requisição para o mesmo, para que o dispositivo tivesse as informações necessárias para registro da aplicação.

III. Conclusões e trabalhos futuros

Considerando a arquitetura que foi desenvolvida, foi possível ser feita esta integração, e o dispositivo conseguiu se registrar no EurekaServer. Fácil configuração dos dispositivos e dos micro-serviços é algo plausível para preenchimento da lacuna deste trabalho. Sendo assim, foram abertas possibidades para desenvolvimento de outros serviços que se integrem com os dispositivos de forma recíproca. Para trabalhos futuros, é viável o desenvolvimento de dispositivos que tenham funções coesas, seguindo o padrão de arquitetura dos micro-serviços. De tal forma, se os micro-serviços e os dispositivos trabalhem de forma independente, torna-se fácil a manutenção de cada serviço ou dispositivo sem afetar os demais.

Apêndice A
Início da thread

```
@SpringBootApplication
@EnableEurekaServer
@RestController
@RequestMapping("/")
public class Application {
    public static void main(String[] args)
      throws InterruptedException {

        Thread thread = new Thread(() -> {
            System.out.println("------>  thread");
            Server server = new Server();
            server.run();
        });

        thread.start();

        SpringApplication.run(Application.class,
    args);
    }

    @RequestMapping(value = "send-server",
      method = RequestMethod.GET)
    public String sendServer() {
        return "Server send!";
    }
}
```

Figura 3. Início da thread no Eureka Server

Apêndice B
Espera por registro

```
function waitRegister()
local srvConfigure = net.createServer(net.TCP, 0)
  srvConfigure:listen(
  8000,
  function(conn)
    conn:on(
    "receive",
    function(conn, request)
      print(request)
      local buf = "ok";
      local index =
          string.find(request, "?ip=")
      local novo = string
          .sub(request, index)
      local ip = string
          .match(novo, "%d+%.%d+%.%d+%.%d+%:*%d*")

      if (ip ~= nil) then
        registered = true
        registerEureka(wifi.sta.getip(), "http://"
    ..ip)
      end

      conn:send(buf);
      conn:close();
      collectgarbage();
    end)
    end)
end
```

Figura 4. Espera do Eureka Server.

APÊNDICE C
REGISTRO NO EUREKA

```
function tryingToRegister()
  local registered = false
  timeout = 0
  tmr.alarm(2, 1000, 1,
    function()
        if wifi.sta.getip() == nil
          and wifi.sta.getbroadcast() == nil then
          print("IP unavaiable, waiting... " ..
timeout)
          timeout = timeout + 1
          if timeout >= 20 then
            print("Deleting configuration...")
            file.remove("config.lc")
            node.restart()
          end
        else
        if (pcall(function()
          ulala = net.createConnection(net.UDP, 0)
          ulala:send(1234, wifi.sta
            .getbroadcast(), "Request Address
Eureka")
          tmr.alarm(3, 1000, 1, function()
            if registered == false then
              print("Trying to register...")
              local ulala = net.createConnection(
net.UDP)
              ulala:send(1234, wifi.sta.
getbroadcast(),
                "Request Address Eureka"..tostring
(math.random(1,1000)))
              ulala:close()
              ulala = nil
            else
              tmr.stop(3)
            end
          end)
        end)) then
          print("Send register to Broadcast...")
        else
          print("Error to send register in
Broadcast...")
        end

        print("Connected, IP is " .. wifi.sta.
getip())
        tmr.stop(2)
        end
      end
  )
end
```

Figura 5. Registro no Eureka.

APÊNDICE D
SOCKET COM BROADCAST

```java
public class Server {

  public static final int DEFAULT_PORT = 1234;
  private DatagramSocket socket;
  private DatagramPacket packet;

  public void run() {
      try {
          socket = new DatagramSocket(DEFAULT_PORT
);
      } catch (Exception ex) {
          System.out.println("Problem creating
socket on port: "
            + DEFAULT_PORT);
      }

      packet = new DatagramPacket(new byte[1], 1);

      while (true) {
        try {
          socket.receive(packet);
          System.out.println("Received from: "
            + packet.getAddress() + ":" + packet.
getPort());
          byte[] outBuffer = new java
            .util.Date().toString().getBytes();
          packet.setData(outBuffer);
          packet.setLength(outBuffer.length);
          socket.setBroadcast(true);
          socket.send(packet);

          Set<InetAddress> localAddress =
getLocalAddress();

          Set<String> ips = localAddress.stream()
                  .map(ad -> ad.getHostAddress())
                  .collect(Collectors.toSet())
                  .stream().sorted()
                  .collect(Collectors.toSet());

          RestTemplate template = new RestTemplate
();

          ips.forEach(ip -> {
              try {
                  template.exchange("http://"
                  + packet.getAddress().
getHostAddress()
                  .concat(":8000?ip={ip}"),
                          HttpMethod.GET,
                          HttpEntity.EMPTY,
                          Void.class,
                          ip.concat(":8000"));
              } catch (Exception e) {
                  e.printStackTrace();
              }
          });

          System.out.println("Message ----> " +
packet
            .getAddress().getHostAddress());
        } catch (IOException ie) {
          ie.printStackTrace();
        }
      }
  }
}
```

Figura 6. Socket com Broadcast

APÊNDICE E
PROTOCOLO PARA REGISTRO NO EUREKA SERVER

```
function registerEureka(ip, addressEureka)
  print("Registering on Address Eureka ..
  addressEureka.." the ip "..ip)
  http.post(
  addressEureka.."/eureka/apps/appID",
  "Content-Type: application/json\r\n",
  [[{
  "instance": {
    "hostName": "]]..ip..[[",
    "app": "]].."NodeMcuEsp8266"
      ..node.chipid()..[[",
    "ipAddr": "http://]]..ip..[[",
    "status": "UP",
    "port": {
      "@enabled": "true",
      "$": "8080"
    },
    "securePort": {
      "@enabled": "false",
      "$": "443"
    },
    "dataCenterInfo": {
      "@class": "com.netflix.appinfo
        .InstanceInfo$DefaultDataCenterInfo",
      "name": "MyOwn"
    },
    "leaseInfo": {
      "renewalIntervalInSecs": 30,
      "durationInSecs": 90,
      "registrationTimestamp": 1492644843509,
      "lastRenewalTimestamp": 1492649644434,
      "evictionTimestamp": 0,
      "serviceUpTimestamp": 1492644813469
    },
    "homePageUrl": "http://]]..ip..[[:8080/",
    "statusPageUrl": "http://]]..ip..[[:8080/info"
,
    "healthCheckUrl": "http://]]..ip..[[:8080/
  health",
    "vipAddress": "]].."NodeMcuEsp8266"..node.
  chipid()
    ..[["}}]],
  function(code, data)
      if (code < 0) then
          print("HTTP request failed")
      else
          gpio.mode(4, gpio.OUTPUT)
          gpio.write(4, gpio.LOW)
          print(code, data)
      end
  end)
end
```

Figura 7. JSON Eureka

APÊNDICE F
SERVIDOR PARA CONTROLE GPIO

```
function registerServer()
  srv:listen(
    8080,
    function(conn)
      conn:on("receive", function(client,request)
        local buf = [[
        <!DOCTYPE html>
        <html lang="pt-br">
        <head>
        <meta charset="UTF-8">
        <title>NodeMcu ESP8266</title>
        </head>
        <body>
        ]];
        local _, _, method,
        path, vars = string
        .find(request, "([A-Z]+) (.+)?(.+) HTTP");
        if(method == nil)then
          _, _, method, path =
          string.find(request, "([A-Z]+) (.+) HTTP
  ");
        end
        local _GET = {}
        if (vars ~= nil)then
          for k, v in string
            .gmatch(vars, "(%w+)=(%w+)&*") do
              _GET[k] = v
          end
        end
        buf = buf.."<h1> ESP8266 Web Server</h1>";
        for i=1,8,1
        do
            gpio.mode(i, gpio.OUTPUT)
            buf = buf.."<p>DIGITAL "..i
            .." <a href=\"?pin="..i.."&stat=on\">
                  <button>ON</button>
                </a> 
                <a href=\"?pin="..i.."&stat=off\">
                <button>OFF</button></a></p>"
        end

        if (_GET.stat ~= nil and _GET.stat ~= nil)
  then
            if (_GET.stat == "on") then
                gpio.write(_GET.pin, gpio.HIGH);
            else
                gpio.write(_GET.pin, gpio.LOW);
            end
        end

        buf = buf..[[
            </body>
            </html>
        ]]

        print(buf)

        client:send(buf);
      end)

      conn:on(
          "sent",
          function(conn)
              conn:close();
              collectgarbage();
          end)
    end)
end
```

Figura 8. Servidor GPIO

REFERÊNCIAS

[1] Willian Marques Freire. Munif Gebara Júnior. *I*oT - A Internet das coisas, vol. 1, 2017

[2] Willian Marques Freire. Munif Gebara Júnior. *I*oT - Micro-serviços, vol. 1, 2017

[3] Pedro Zambarda. 'Internet das Coisas': entenda o conceito e o que muda com a tecnologia. 2014 [Online] Disponível: http://www.techtudo.com.br/noticias/noticia/2014/08/internet-das-coisas-entenda-o-conceito-e-o-que-muda-com-tecnologia.html [Acesso: 11-Mai-2017]

[4] Netflix. *E*ureka at a glance. 2014 [Online] Disponível: https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance [Acesso: 26-Jun-2017]

[5] Finep. *K*evin Ashton – entrevista exclusiva com o criador do termo "Internet das Coisas", 2015 [Online] Disponível: http://finep.gov.br/noticias/todas-noticias/4446-kevin-ashton-entrevista-exclusiva-com-o-criador-do-termo-internet-das-coisas. [Acesso: 13-Jan-2017]

[6] Ricardo Feltrin. *N*etflix fatura R$ 1,1 bi no Brasil e ultrapassa o SBT, 2016 [Online] Disponível: https://tvefamosos.uol.com.br/noticias/ooops/2016/01/11/netflix-fatura-r-11-bi-no-brasil-e-ultrapassa-o-sbt.htm. [Acesso: 01-Mai-2017]

[7] SmartBear. *N*ETFLIX E O SEU SUCESSO COM APIS, 2016 [Online] Disponível: http://mundoapi.com.br/materias/netflix-e-o-seu-sucesso-com-apis. [Acesso: 20-Fev-2017]

[8] Manu Tayal. *I*oT and Microservices Architecture. [Online] Disponível: http://www.happiestminds.com/blogs/iot-and-microservices-architecture. 2016 [Acesso: 25-Abr-2017]

[9] IDC. *C*onnecting the IotT: The road to success. 2016 [Online] Disponível: http://www.idc.com/infographics/IoT. [Acesso: 15-Fev-2017]

[10] Martin Fowler et al. *M*icroservices. 2014 [Online] Disponível: https://www.martinfowler.com/articles/microservices.html. [Acesso: 20-Mai-2017]

[11] Lúcia Leão. *O* labirinto da hipermídia: arquitetura e navegação no ciberespaço. 1997 [Online] Disponível: http://www.alvarestech.com/lillian/Pos-graduacao/Hipermidia.pdf. [Acesso: 01-Jul-2017]

[12] Marcos Elias. *O* que é FTP? E HTTP? 2010 [Online] Disponível: http://www.explorando.com.br/ftp-http. [Acesso: 01-Jul-2017]

[13] Roy Thomas Fielding. *R*epresentational State Transfer (REST) 2000 [Online] Disponível: http://www.ics.uci.edu/fielding/pubs/dissertation/rest_arch_style.htm. [Acesso: 01-Jul-2017]

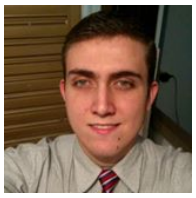
[14] Pedro Pinto. *R*edes – Sabe o que são sockets de comunicação? (Parte I) 2012 [Online] Disponível: https://pplware.sapo.pt/tutoriais/networking/redes-sabe-o-que-sao-sockets-de-comunicacao-parte-i.

[15] Fábio Jordao. *O* que são threads em um processador? 2011 [Online] Disponível: https://www.tecmundo.com.br/9669-o-que-sao-threads-em-um-processador-.htm

[16] Ecma international. *T*he JSON Data. 2013 [Online] Disponível: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[17] Sasang Balachandran. *G*eneral Purpose Input/Output (GPIO), vol. 1, 2009

[18] Simpsons Wikia. *T*he Java Server 2017. [Online] Disponível: http://simpsons.wikia.com/wiki/The_Java_Server



Willian Marques Freire Possui Ensino Médio completo pelo Colégio Estadual Rosa Delúcia Calsavara (2013). Atualmente é Desenvolvedor de Software da Gumga Tecnologia da Informação S/A. Tem experiência na área de Ciência da Computação.

Munif Gebara Júnior Possui graduação em Ciência da Computação pela Universidade Estadual de Maringá (1997) e mestrado em Engenharia Elétrica e Informática Industrial pela Universidade Tecnológica Federal do Paraná (2001). Atualmente é professor da Fundação Faculdade de Filosofia Ciências e Letras de Mandaguari e professor de ensino superior da Faculdade de Tecnologia e Ciências do Norte do Paraná Ltda e desenvolvedor.

3 CONCLUSÕES E TRABALHOS FUTUROS

Durante este trabalho foram desenvolvidos três artigos, nos quais contêm diversas informações sobre IoT e Micro-serviços. Dentres estes, foram desenvolvidos testes e exemplos práticos dos mesmos, incluindo a integração deles. Como é necessário conexão com internet para compartilhamento de informações, foi utilizado o dispositivo NodeMCU ESP8266, pois este contém módulo WiFi integrado, o que possibilita a comunicação com outros serviços e dispositivos, além de um microcontrolador para pequeno processamento. Também foram utilizadas ferramentas prontas para desenvolvimento dos serviços, o que facilita e agiliza o desenvolvimento da aplicação. Considerando a arquitetura que foi estabelecida, conclui-se que é possível desenvolver esta integração, sendo assim, o objetivo inicial do trabalho é concluido. A mesma abre possibilidades para a integração entre novas tecnologias e serviços, pois visa a independência e alta coesão de cada serviço e dispositivo. Automação residencial e industrial, são apenas duas das áreas que podem ser beneficidadas, devido a simples configuração que é empregada a esta estrutura.

REFERÊNCIAS

FELTRIN. Ricardo *N*etflix fatura R$ 1,1 bi no Brasil e ultrapassa o SBT. Disponível em: <https://tvefamosos.uol.com.br/noticias/ooops/2016/01/11/netflix-fatura-r-11-bi-no-brasil-e-ultrapassa-o-sbt.htm>. Acesso em: 15 Jan. 2017.

MARQUES, Willian; GEBARA, Munif. *I*oT - A Internet das coisas, vol. 1, 2017

MARQUES, Willian; GEBARA, Munif. *I*oT - Micro-serviços, vol. 1, 2017

MARQUES, Willian; GEBARA, Munif. *I*ntegração entre IoT e Micro-serviços, vol. 1, 2017

Martin Fowler et al. *M*icroservices. Disponível em: <https://www.martinfowler.com/articles/microservices.html>. Acesso em: 25 Jul. 2017.

WORLD. *O*ptar por micro-serviços em vez de software monolítico. Disponível em: <https://www.computerworld.com.pt/2016/12/05/optar-por-micro-servicos-em-vez-de-software-monolitico/>. Acesso em: 16 Jan. 2017.