

Fundação Faculdade de Filosofia, Ciências e
Letras de Mandaguari
Curso de Ciência da Computação



IoT e integração com Micro-serviços:
Subtítulo

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 Universidade Federal da Paraíba,
como requisito para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Munif Gebara Júnior

Junho 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 Computação intitulado ***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, June 28, 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

O agradecimento é opcional

RESUMO

Um resumo de trabalho de conclusão de curso é do tipo informativo e deve conter somente um parágrafo. A estrutura do resumo deve conter essencialmente os seguintes tópicos: apresentar inicialmente os objetivos do trabalho (o que foi feito?), a justificativa (porquê foi feito) e, finalmente, os resultados alcançados. O resumo deve informar ao leitor todas as informações importantes para o que o leitor possa entender o trabalho desenvolvido, quais foram as finalidades, a metodologia que o autor utilizou e os resultados obtidos. Deve conter frases curtas, porém completas (evitar estilo telegráfico); usar o tempo verbal no passado para os principais resultados e presente para comentários ou para salientar implicações significativas. O resumo em português e inglês são obrigatórios e não devem passar de 200 palavras.

Palavras-chave: <Primeira palavra>, <segunda palavra>, <até 5 palavras>. <Obs.: as palavras-chave devem ser escolhidas com bastante rigor, pois devem representar adequadamente os principais temas abordados pela pesquisa.>

ABSTRACT

<Resumo em Inglês - Write here the abstract of your work>

Key-words:

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

SIGLA – NOME COMPLETO

LUMO – Laboratório de computação Móvel e Ubíqua

UbiComp – Computação Ubíqua

Sumário

1	INTRODUÇÃO	17
2	APRESENTAÇÃO E ANÁLISE DOS RESULTADOS	42
3	CONCLUSÕES E TRABALHOS FUTUROS	43
	REFERÊNCIAS	43
A -	ANEXOS E APÊNDICES 1	45

1 INTRODUÇÃO

IoT e Micro-serviços, dois assuntos distintos, mas que de certa forma pode haver uma conexão entre os mesmos. Os dois assuntos foram explanados durante dois artigos, primeiramente IoT - A Internet das coisas e posteriormente Micro-serviços de Marques (2017). Neste contexto, aplica-se a integração dos mesmos, pois de fato, como são assuntos que vem sendo estudados nestes artigos, obteve-se a idéia de fazer com que os dois trabalhem em conjunto. Micro-serviço é um padrão que segundo Fowler (2016) têm originado muitos projetos, e os resultado têm sido positivos, entretanto, não existe muita informação que descreve o que são e como implementá-los.

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 que como a empresa está crescendo, um dos objetivos é que a mesma tenha uma estabilidade mais palatável. O mesmo também fala que atualmente, ainda grande parte dos sistemas da empresa, funciona de forma monolítica, e estão trabalhando no desacoplamento dos mesmos, para que um não afete os outros, ter 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, e para gerar tal tráfego de streaming de dados, precisa-se de uma arquitetura robusta, para que atenda o mesmo (FELTRIN, 2016).

A Amazon foi uma das primeiras empresas, em que migrou 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).

Segundo Fowler (2014), Micro-serviço é mais um novo termo na área de arquitetura de software. Segundo ele, a terminologia descreve um estilo de sistemas de software, que tem se tornando o estilo padrão para o desenvolvimento de aplicativos corporativos. Algumas características como alta coesão, autonomia, resiliência, observáveis, automatização e centralização no domínio de negócio fazem parte da arquitetura de micro-serviços.

Aproveitando-se deste contexto tecnologico de distribuição de dados, um assunto que está em discussão até mesmo entre os leigos, 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 seria movimentado em média de US\$41 bilhões somente na área de IOT (IDC, 2016).

IoT - A Internet das coisas

Willian Marques Freire e Munif Gebara Junior

Resumo—The abstract goes here.

Index Terms—IEEE, IEEEtran, journal, L^AT_EX, paper, template.

I. INTRODUÇÃO

IOT ou Internet das coisas, algumas vezes referida como a Internet dos objetos, está gerando uma revolução tecnológica em diversas áreas e inclusive na humanidade. Considerando que o IoT representa a próxima evolução da Internet, dando um salto na 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].

Segundo o Artigo Internet of Things volume III, na empresa Dzone (empresa que faz publicações online sobre tecnologia) encontram-se 797 profissionais que são responsáveis pela área de IoT, e uma das maiores preocupações da mesma é este âmbito. Em um inquérito realizado pela mesma, foi perguntado 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 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].

Em 1999 foi fundado um grupo chamado MIT (Massachusetts Institute of Technology), que tem trabalhado no campo de identificação de frequência de rádio, em rede (RFID) e tecnologias de sensores emergentes. Em 2003, haviam aproximadamente 6,3 bilhões de pessoas vivendo no mundo, e aproximadamente 500 milhões de dispositivos conectados à internet. O crescimento mastodôntico de smartphones e tablets, elevou o número de dispositivos conectados a Internet para aproximadamente 12,5 bilhões em 2010 [1, p. 3].

Em janeiro de 2009, uma equipe de pesquisadores chinesa, finalizou um estudo sobre os dados de roteamento da internet em intervalos de seis meses, entre dezembro de 2001 e dezembro de 2006. 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.

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.

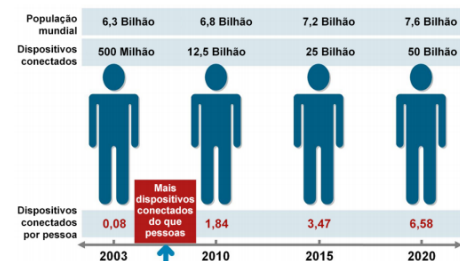


Figura 1. "Nascimento" do IoT entre 2008 e 2009 [1].

A Internet tem 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, compartilharem informações na Internet para 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, Twitter se tornaram famosas e profícuas, ao permitir que pessoas se comuniquem e compartilhem informações [1, p. 6].

IoT Tem mudado o modelo de hardware computacional, que nasceu a aproximadamente 40 anos atrás. Considerando todas as fases diferentes de modelos de hardware que existiram, dentre elas micro-computadores pessoais, e hoje computadores conectados na nuvem expandindo o modelo cliente-servidor para novos paradigmas, comprova que a cada revolução IoT surge a necessidade de modificar este modelo [3, p. 6]. Devido a diversos estudos visando melhor desenvolvimento e aproveitamento do modelo cliente-servidor, surgiram vários protocolos para transmissão de dados. 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.

Aproveitando-se destes protocolos, também surgiu especificações e arquiteturas, para que aplicações possam se 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 restrições arquiteturais aplicadas a componentes, conectores e elementos de dados dentro de um sistema de hipermídia distribuído, entretanto,

é um assunto para o próximo artigo que será falado sobre Micro-serviços.

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.

13 de Maio de 2017

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 entrevistados diversas pessoas, para obter um percentual de preferência entre os dispositivos. Dentre os mesmos, 53% preferem Raspberry 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 Bluetooth 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), 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. Além destes aspectos, complexos sistemas sociotécnicos cercam a humanidade. 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. O ultimo desafio para área de IoT, são as identificações e diagnóstico 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, 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 de processamento, 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 ideia de computação ubíqua, um conceito que surgiu no final dos anos 80. Segundo Mark Weiser, criador do conceito citado, 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, entretanto, o utilizem. 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].

Em um artigo feito pelo professor Michael Wooldridge, chefe do Departamento de Ciência da Computação da Universidade de Oxford, o mesmo definiu um termo utilizado no meio tecnológico, que são os agentes. Sendo 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 resposta 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].

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. 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].

2) *NodeMCU*: Neste trabalho, será utilizado o NodeMCU para desenvolvimento das aplicações IoT, pelo fato do mesmo conter um módulo WiFi, o que facilitará na comunicação via interface de rede com os micro-serviços. NodeMCU é um firmware baseado em eLua - uma 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 Shanghai, 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 Ierusalimsky, 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 não 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 de 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 2003 e é totalmente compatível com o 802.11b, e possui como atrativo a possibilidade de trabalhar com taxas de transmissão 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 transmissão de dados. Resumidamente, Wi-Fi é um conjunto de especificações 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 utilizar tecnologias sem fio como Wi-Fi, assim como todos os sistemas, medidas de segurança devem ser utilizadas para prevenir 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 praticamente 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 número 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 criptografia 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 especialistas, o risco de intrusos em redes domésticas é quase zero. Isto se deve à 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 criptografia que protege os dados que passam pela rede. Devido à complexidade do mesmo, muitos dispositivos, mesmo 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 utilizada uma ferramenta chamada ESPlorer. Ela é utilizada para facilitar o desenvolvimento de aplicações LUA para dispositivos 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.

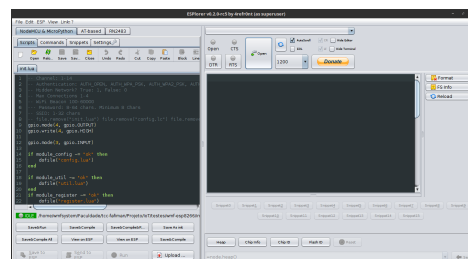


Figura 2. ESPlorer [7]

7) *Protocolos TCP e UDP*: O protocolo TCP ou Protocolo de controle de transmissão, é um dos protocolos 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 feitas verificações 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 criptografia 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 ao meio físico de transmissão 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 Protocol). 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 distribuído 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.

```
1 #define LUA_USE_MODULES\MQTT
2 // #define LUA_USE_MODULES\COAP
3 // #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 do 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 Ahlberg, 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.

Como este projeto utiliza o docker, uma infraestrutura independente de plataforma [5], primeiramente é atualizado o container e posteriormente é executado um comando para

```
1 $ docker pull marcelstoer/nodemcu-build
2 $ sudo docker run --rm -ti -v `pwd`::/opt
3 /nodemcu-firmware
4 marcelstoer/nodemcu-build
```

Figura 4. Compilação do firmware

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.

```
1 $ sudo python esptool.py
2   --port="/dev/ttyUSB1"
3   erase_flash
4
5 $ sudo python ./esptool.py
6   --port /dev/ttyUSB0
7   write_flash
8   0x00000
9   ../nodemcu-firmware/bin/0x00000.bin
10  0x10000
11  ../nodemcu-firmware/bin/0x10000.bin
12
13 $ sudo python ./esptool.py
14   --port="/dev/ttyUSB0"
15   write_flash -fm=dio
16   -fs=16m 0x00000
17   ../nodemcu-firmware/bin
18   /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 endereços 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 necessário 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.


```
1 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.

```
1 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.lua`, 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 configurações como: `ssid` (Nome da conexão), `pwd` (Senha da conexão), `auth` (Tipo de autenticação), `channel` (Canal que será liberado no dispositivo Wi-Fi), `hidden` (indica se é visível), `max` (máximo de conexões simultâneas) e `beacon` (intervalo de tentativas de conexão do dispositivo). Quando é encontrado o arquivo, é feito uma leitura do arquivo, e neste trabalho, está sendo salvo no padrão usuário e senha separados por um espaço em branco, por isto está sendo separado em duas palavras ao ler a primeira linha do arquivo. 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, deve-se fazer conforme a figura 8.

```
1 dofile("config.lua")
2 dofile("util.lua")
3 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 possibilitou manter as configurações salvas no dispositivo, para que se ocorra alguma falha, ou o dispositivo seja desligado, não precise ser reconfigurado. Foi implementado uma regra no dispositivo, que faz a verificação de conexão a internet dentro de 20 segundos, se ocorrer alguma falha na conexão, será deletado o arquivo de configurações e reiniciado o dispositivo, liberando assim, o acesso à configuração de rede.

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

III. CONCLUSÕES E TRABALHOS FUTUROS

Utilizando-se de 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. O padrão de desenvolvimento neste dispositivo é utilizando a linguagem LUA, entretanto, existem alguns projetos que possibilitam o desenvolvimento na linguagem C. Como neste trabalho foi desenvolvido uma estrutura IoT de fácil configuração e com tecnologias atuais 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, pois como a lacuna de uma simples configuração, integrando o dispositivo com a rede de internet de forma fácil, foi resolvida.

APÊNDICE A
CÓDIGO FONTE CONFIGURACIONAL: CONFIG.LUA

```

1 function configureWifi()
2   local ap = {}
3   wifi.sta.getap(
4     function(t)
5       i = 0
6       for ssid, v in pairs(t) do
7         ap[i] = ssid
8         i = i + 1
9       end
10    end)
11  srv:listen(
12    80,
13    function(conn)
14      conn:on(
15        "receive",
16        function(conn, payload)
17          listaAp = "SSID: <select name='ssid'>"
18          print("Request Configuration")
19          for i = 0, tablelength(ap), 1 do
20            if ap[i] ~= nil then
21              listaAp = listaAp
22                .. "<option value='"
23                .. ap[i] .. "'>"
24                .. ap[i] .. "</option>"
25            end
26          end
27          listaAp = listaAp .. "</select><br/>"
28
29          local _GET = getParamsUrl(payload)
30
31          if (_GET ~= nil) then
32            if (_GET.ssid ~= nil
33              and _GET.password ~= nil) then
34              print("Connecting to wifi
35                and creating configuration...")
36              print("SSID: "
37                .. _GET.ssid)
38              print("Password: "
39                .. _GET.password)
40
41              for i = 0, tablelength(ap), 1 do
42                if ap[i] ~= nil then
43                  if (string
44                    .find(ap[i], _GET.ssid)) then
45                    user = ap[i]
46                  end
47                end
48              end
49
50              password = _GET.password
51              wifi.sta.config(user, password)
52              wifi.sta.connect()
53
54              if file.open("config.lc", "w") then
55                file.writeline(user
56                  .. " " .. password)
57                file.close()
58              end
59
60              node.restart()
61              node.chipid()
62            end
63            conn:send([[... HTML Whatever...]])
64            conn:on("sent", function(conn)
65              conn:close()
66              collectgarbage()
67            end)
68          end)
69        end
70      end

```

Figura 9. Módulo Utilitário

APÊNDICE B
CÓDIGO FONTE UTILITÁRIO: UTIL.LUA

```

1 function split(str, pat)
2   local t = {}
3   — NOTE: use {n = 0} in Lua-5.0
4   local fpat = "(.-)" .. pat
5   local last_end = 1
6   local s,
7     e,
8     cap = str:find(fpat, 1)
9   while s do
10     if s ~= 1 or cap ~= "" then
11       table.insert(t, cap)
12     end
13     last_end = e + 1
14     s,
15     e,
16     cap = str:find(fpat, last_end)
17   end
18   if last_end <= #str then
19     cap = str:sub(last_end)
20     table.insert(t, cap)
21   end
22   return t
23 end
24
25 local unescape = function(s)
26   s = string.gsub(s, "+", " ")
27   s = string.gsub(
28     s,
29     "%(%%(%%x%%x))",
30     function(h)
31       return string.char(tonumber(h, 16))
32     end
33   )
34   return s
35 end
36
37 function tablelength(T)
38   local count = 0
39   for _ in pairs(T) do
40     count = count + 1
41   end
42   return count
43 end
44
45 function getParamsUrl(request)
46   local buf = ""
47   local
48     —, —,
49     method,
50     path,
51     vars = string
52       .find(request, "([A-Z]+) (.)?(.) HTTP")
53   if (method == nil) then
54     —, —,
55     method,
56     path = string.find(request, "([A-Z]+) (.)
57     HTTP")
58   end
59   local _GET = {}
60   if (vars ~= nil) then
61     for k, v in string
62       .gmatch(vars, "(%w+)=(%w+)&*" ) do
63       _GET[k] = unescape(v)
64     end
65   end
66   return _GET
67 end

```

Figura 10. Módulo Configuracional

APÊNDICE C

CÓDIGO FONTE INICIAL: INIT.LUA

```

1  srv = net.createServer(net.TCP)
2
3
4  if file.exists("config.lua") == true then
5
6      print("Open configuration...")
7
8      if file.open("config.lua") 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

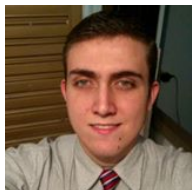
```

Figura 11. Central de Configuração

REFERÊNCIAS

- [1] Dave Evans. *The 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. *Micro-serviços*, vol 1, pp. 1-3, 2017
- [3] John Esposito. *The 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. *Docker 2017* [Online] Disponível: <https://www.docker.com>. [Acesso: 11-Jun-2017]
- [6] Peter Andersson. SPIFFS (SPI Flash File System) 2013 [Online] Disponível: <https://github.com/pellepl/spiffs>. [Acesso: 11-Jun-2017]
- [7] ESPlorer. *ESPlorer Integrated Development Environment (IDE) for ESP8266 developers* 2013 [Online] Disponível: <https://github.com/4refr0nt/ESPlorer>. [Acesso: 11-Jun-2017]
- [8] Felipe Demartini. WEP, WPA, WPA2: o que as siglas significam para o seu Wi-Fi? 2013 [Online] Disponível: <https://www.tecmundo.com.br/wifi/42024-wep-wpa-wpa2-o-que-as-siglas-significam-para-o-seu-wifi.htm>. [Acesso: 03-Jun-2017]
- [9] Mundo Max. *Qual 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-roterador-wireless-e-um-access-point/>. [Acesso: 03-Jun-2017]
- [10] Irving Souza Lima. *NodeMCU (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. *The Computer for the 21st Century*, vol 1, pp. 1-2, 1991
- [12] John B. Kennedy. *WHEN WOMAN IS BOSS*, 1926 [Online] Disponível: <http://www.tfcbooks.com/tesla/1926-01-30.htm>. [Acesso: 10-Mai-2017]
- [13] Michael Wooldridge e Nicholas R. Jennings. *Intelligent agents: theory and practice*, vol 1, pp. 1-3, 1995
- [14] Danilo Sato. *CanaryRelease*. 2014 [Online] Disponível: <https://martinfowler.com/bliki/CanaryRelease.html>. [Acesso: 10-Abr-2017]
- [15] NodeMCU. *NodeMCU 2.0.0*. [Online] Disponível: <https://github.com/nodemcu/nodemcu-firmware>. [Acesso: 10-Dez-2017]
- [16] ELua. *What is eLua*. [Online] Disponível: <http://www.eluaproject.net/overview>. [Acesso: 19-Nov-2017]
- [17] Systems. *Expressif 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. *Authors*. [Online] Disponível: <http://www.lua.org/authors.html>. [Acesso: 5-Fev-2017]
- [20] Roy Thomas Fielding. *Representational State Transfer (REST)*. [Online] Disponível: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. [Acesso: 29-Mai-2017]
- [21] Cate Lawrence. *Internet of Things Applications, Protocols, and Best Practices*, vol 4, pp. 1-10, 2017
- [22] Filipeflop. *Controle 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. *Redes – 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. *RFC 1594*, 1994 [Online] Disponível: <https://tools.ietf.org/html/rfc1594>. [Acesso: 17-Jun-2017]
- [26] NodeMCU. *GPIO Module?* 2014 [Online] Disponível: <https://nodemcu.readthedocs.io/en/master/en/modules/gpio/> [Acesso: 17-Jun-2017]
- [27] NodeMCU. *SJSON Module* 2017 [Online] Disponível: <https://nodemcu.readthedocs.io/en/master/en/modules/sjson/> [Acesso: 17-Jun-2017]
- [28] NodeMCU. *Timer 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—The abstract goes here.

Index Terms—IEEE, IEEEtran, journal, L^AT_EX, paper, template.

I. INTRODUÇÃO

MICRO SERVIÇOS, um novo paradigma que influencia diretamente o modo em que são desenvolvidas, e distribuídas as aplicações. Após diversos estudos realizados nos últimos anos, para descrever o termo "Arquitetura de Micro-serviços", foi definido que, de uma maneira específica, é possível desenvolver softwares como suítes de serviços com deploy (implantação) independente. Embora não exista uma definição precisa deste tipo de arquitetura, devido a sua complexidade, há certas características relacionadas à sua organização, à capacidade de negócio independentes, ao deploy automatizado, à inteligência e controle descentralizado de linguagens e de dados [4].

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. Neste contexto, aplica-se também as vantagens e desvantagens dos sistemas monolíticos.

Um dos principais pontos negativos, é que, se tem um grande ponto de falha, fazendo com que assim este fique fora do ar, isto levará junto o sistema inteiro, 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 projetos improdutivo durante algum tempo, já que a complexidade do código é bem maior [5].

Em uma publicação feita por Cleuton Sampaio [3], o mesmo definiu através de estudos que Micro-serviços são, componentes de alta coesão, baixo acoplamento, autônomos e independentes, que representa um contexto de negócio de uma aplicação. Um fato que ocorreu no ano de 2014, foi que o Docker, uma plataforma Open Source escrito em Go - linguagem de programação de alto desempenho desenvolvida dentro do Google [6], veio como um container portátil padronizado, e está sendo utilizado pela comunidade. Uma razão importante para sua utilização generalizada que Adrian

(Membro e fundados da eBays Research Labs) observa, é sua portabilidade e aumento da velocidade com container, que entregava algo em minutos ou horas e passou a entregar em segundos. Na figura 1 é apresentado sua utilização entre os anos 2012 e 2016.

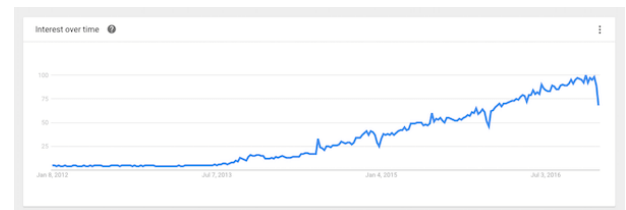


Figura 1. Gráfico de utilização do docker entre os anos 2012 e 2016.

A velocidade de desenvolvimento e implantação de um micro-serviço, permite e incentiva a implementação e estudo dos mesmos. Segundo Adrian micro-serviços possuem características em comum, como: implantação com pouca frequência, novas versões implantadas automaticamente, 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].

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, utilizando comunicação REST através de JSON ou XML, facilitar a ampliação e integração de micro-serviços com serviços terceirizados, através de APIs, organizar o código em função de capacidades de negócio, dando mais visão das ofertas e necessidades dos clientes. Dentre todos os benefícios citados, também é possível fazer o gerenciamento de falhas, o que significa que, se um serviço venha a falhar, os outros continuarão funcionando. Através dos micro-serviços, é possível identificar falhas com mais eficiência, visto que, o particionamento favorece uma visão mais detalhada de cada serviço [8].

Assim como existem os pontos positivos, existem também os negativos, tudo tem seu uso ideal. Em uma publicação feita por Flavio Silveira [2], ele diz que micro-serviços não são uma bala de prata que resolverá todos os problemas de um projeto. A arquitetura de micro-serviços visa o particionamento da aplicação com coesão, entretanto, dependendo da quantidade de partições existentes, torna-se dificultoso a implantação dos mesmos, pelo fato de serem muitos. Além da quantidade de micro-serviços para se implantar, também o cuidado pelas

mesmas será maior, pois ao invés de dar manutenção em uma aplicação, aqui serão várias.

Este trabalho têm por objetivo, preencher a lacuna a respeito do desenvolvimento de um micro-serviço, fazendo com que, utilizando-se de ferramentas existentes, seja fácil a criação de uma estrutura para micro-serviços. No Artigo IoT - Internet das coisas [18], é desenvolvido uma estrutura de IoT, onde tem-se por objetivo, a comunicação dos dispositivos com aplicações Web, que ficarão encarregadas de compartilhar informações dos mesmos. Durante este trabalho serão criados micro-serviços que posteriormente abrirá possibilidades para integração de dispositivos IoT com micro-serviços, fechando assim, uma estrutura flexível para automações residenciais, entretanto, é título para um próximo trabalho.

I wish you the best of success.

mds

13 de Maio de 2017

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. 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.

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 assim que os conceitos sejam amplamente adotados pela indústria e pela academia.

O advento e a difusão da orientação por 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 al. 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 ideia em grande escala. Na era pré-Gamma os padrões para soluções OO já estavam sendo utilizados: 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 eletrônico 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".

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 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 2016 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 peso para os provedores de infraestrutura na América Latina [10].

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 microserviços, se quisermos que um aplicativo seja colocado em esteróides, ele pode ser feito sem afetar outros serviços. Podemos apenas começar a executar este serviço específico em um hardware mais forte. Um microservice único pode ser atualizado nesta arquitetura, sem afetar outros ... a única condição é que o sistema de tempo de execução suporta isso. Cada microservice em uma plataforma pode ser desenvolvido em uma linguagem diferente - Java, C, C++, Python, etc Governança granular é possível para cada microservice porque não tem dependência em outro. Ele pode ser monitorado e governado separadamente. Essa arquitetura descentraliza o gerenciamento de dados, uma vez que cada microserviço pode armazenar seus dados de uma maneira que se adapte a ele. Arquitetura Microservice suporta automação. É possível mover montagens inteiras de microservices de um ambiente de implementação para outro apenas usando as configurações de perfil com um único clique. Eles são muito mais resistentes do que as aplicações tradicionais. Isto é devido ao fato de que uma única aplicação pode ser retirada de um monte de aplicativos microservices, como estes são independentes uns dos outros.

A arquitetura do microservice tem suas vantagens óbvias e aquela é a razão porque assim que muitos negócios e serviços

públicos proeminentes como Netflix, eBay, Amazon, o serviço digital do governo BRIT NICO, realestate.com.au, para diante, Twitter, Paypal, Gilt, Bluemix, Soundcloud, The Guardian, etc, apenas para citar alguns, todos se graduaram de arquitetura monolítica a microservices. Embora este seja o caso, assim como não há um plano perfeito, não há nenhuma arquitetura perfeita. O que funciona sob uma circunstância particular pode se tornar o gargalo em outro.

1) Tecnologias para gerenciamento de Micro-serviços:

Neste trabalho será utilizado no Back-end a tecnologia do netflix Service Discovery (Eureka), e para comunicar com este serviço será utilizado o circuito integrado Nodemcu Esp8266. Existem outras bibliotecas que podem trabalhar em conjunto com o Eureka, 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 uma variedade de diferentes tipos de filtros, que permitem aplicar rapidamente funcionalidades aos serviços de ponta. 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, clientes integrados com balanceadores de carga e configuração de clientes utilizando Archaius. 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 im-

plementação 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.

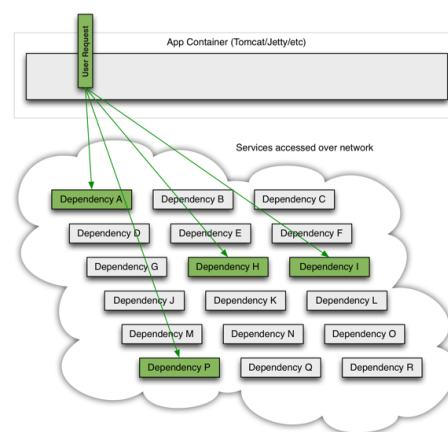


Figura 2. Wiki Hystrix (Internet Overtime) - 2015).

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

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.

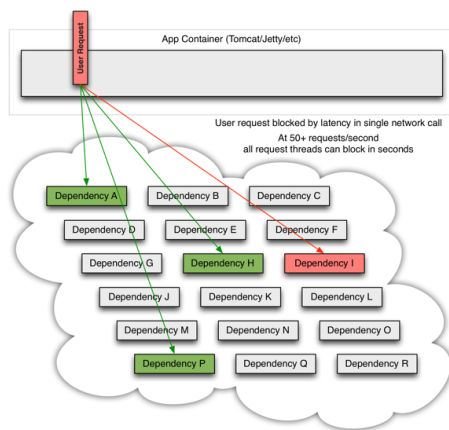


Figura 3. Wiki Hystrix (3 dimensions to scaling) - 2015).

Esses aplicativos também podem resultar em latências entre os serviços, causando ainda mais falhas em cascata em toda a aplicação, conforme a figura 4.

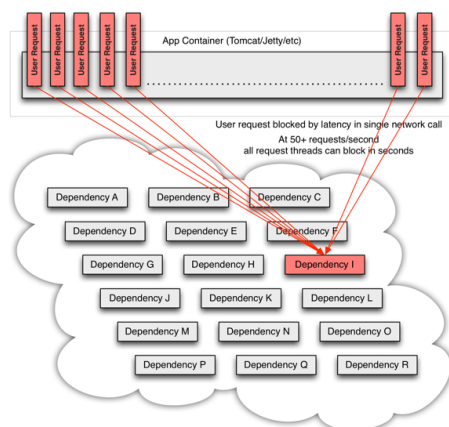


Figura 4. Wiki Hystrix (Container) - 2015).

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 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 auto-configuração, 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.

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.

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.

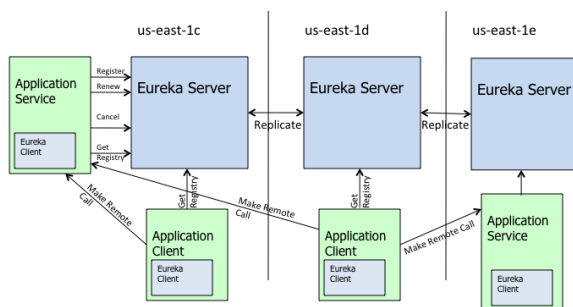


Figura 5. Wiki Eureka - 2015).

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 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.

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.

```

1 @Configuration
2 @ComponentScan
3 @EnableAutoConfiguration
4 @EnableEurekaClient
5 @RestController
6 public class Application {
7
8     @RequestMapping("/")
9     public String home() {
10         return "Hello world";
11     }
12
13     public static void main(String[] args) {
14         new SpringApplicationBuilder
15             (Application.class).web(true)
16             .run(args);
17     }
18 }

```

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.

```

1 Eureka
2 cliente:
3   ServiceUrl:
4     DefaultZone:
5       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 feitas configurações conforme a figura 8.

```

1 eureka:
2   instance:
3     statusPageUrlPath:
4       ${management.context-path}/info
5     healthCheckUrlPath:
6       ${management.context-path}/health
7   client:
8     healthcheck:
9       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 utilizar 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.

```

1 @Autowired
2 private EurekaClient discoveryClient;
3
4 public String serviceUrl() {
5     InstanceInfo instance =
6         discoveryClient
7             .getNextServerFromEureka
8             ("STORES", false);
9     return instance.getHomePageUrl();
10 }

```

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

G. Permanece de registro no Eureka

Registrar um serviço no Eureka pode ser considerado um pouco lento, pelo fato de que, ser uma instância também

```

1 @Autowired
2 private DiscoveryClient discoveryClient;
3
4 public String serviceUrl() {
5     List<ServiceInstance> list =
6         discoveryClient.getInstances
7             ("STORES");
8     if (list != null && list.size() > 0 ) {
9         return list.get(0).getUri();
10    }
11    return null;
12 }

```

Figura 10. Busca de instâncias com DiscoveryClient

envolve um heartbeat periódico para o registro, com duração padrão de 30 segundos. Um serviço não estará disponível para descoberta por clientes, enquanto uma instância 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.

```

1 eureka.instance.metadataMap.zone = zone1
2 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 utilizar 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.

J. Modo Autônomo

A combinação entre o cliente e servidor Eureka, e as pulsões para verificação de disponibilidade entre os mesmos, tornam o servidor Eureka resiliente à falhas, contanto que haja

```

1 @SpringBootApplication
2 @EnableEurekaServer
3 public class Application {
4     public static void main(String[] args) {
5         new SpringApplicationBuilder
6             (Application.class).web(true)
7             .run(args);
8     }
9 }

```

Figura 12. Código inicial Eureka Server

algum tipo de monitoramento para mantê-lo funcionando. No modo autônomo, pode-se preferir desativar o comportamento padrão do lado do cliente, para que ele não continue tentando alcançar seus pares caso haja falha. Para isto será feito diversas configurações como pode ser visto na figura 13.

```

1 server:
2   port: 8761
3
4 eureka:
5   instance:
6     hostname: localhost
7   client:
8     registerWithEureka: false
9     fetchRegistry: false
10    serviceUrl:
11      defaultZone:
12        http://${eureka.instance.hostname}
13        :${server.port}/eureka/

```

Figura 13. Configuração 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

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

```

1 —
2 spring:
3   profiles: peer1
4 eureka:
5   instance:
6     hostname: peer1
7   client:
8     serviceUrl:
9       defaultZone: http://peer2/eureka/
10
11 —
12 spring:
13   profiles: peer2
14 eureka:
15   instance:
16     hostname: peer2
17   client:
18     serviceUrl:
19       defaultZone: http://peer1/eureka/

```

Figura 14. Configuração de perfis

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 implementa o padrão disjuntor que interrompe automaticamente 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

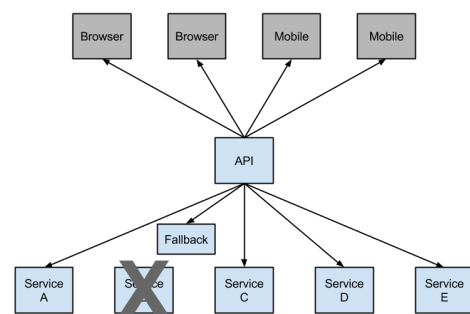


Figura 15. Fallback Hystrix em falhas em cascata.

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

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


```

1 @SpringBootApplication
2 @EnableCircuitBreaker
3 public class Application {
4
5     public static void main(String[] args) {
6         new SpringApplicationBuilder
7             (Application.class).web(true)
8             .run(args);
9     }
10
11 }
12
13 @Component
14 public class StoreIntegration {
15
16     @HystrixCommand
17     (fallbackMethod = "defaultStores")
18     public Object getStores
19     (Map<String, Object> parameters) {
20         //do stuff that might fail
21     }
22
23     public Object defaultStores
24     (Map<String, Object> parameters) {
25         return /* something useful */;
26     }
27 }

```

Figura 16. Hystrix

do mesmo adicionando as dependências conforme a figura 17. O endpoint para acesso ao gerenciador é /hystrix.stream.

```

1 <dependency>
2   <groupId>
3     org.springframework.boot
4   </groupId>
5   <artifactId>
6     spring-boot-starter-actuator
7   </artifactId>
8 </dependency>
9 <dependency>
10  <groupId>
11    org.springframework.cloud
12  </groupId>
13  <artifactId>
14    spring-cloud-starter-hystrix-dashboard
15  </artifactId>
16 </dependency>

```

Figura 17. Dependências do Gerenciador Hystrix

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 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 configuração e será anotado com @RibbonClient, conforme a figura 18.

```

1 @Configuration
2 @RibbonClient(name = "foo",
3   configuration = FooConfiguration.class)
4 public class TestConfiguration {
5 }

```

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.

```

1 @Configuration
2 @ComponentScan
3 @EnableAutoConfiguration
4 @EnableEurekaClient
5 @EnableFeignClients
6 public class Application {
7     public static void main(String[] args) {
8         SpringApplication
9             .run(Application.class, args);
10    }
11 }

```

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.

```

1 @FeignClient("stores")
2 public interface StoreClient {
3     @RequestMapping(method =
4         RequestMethod.GET,
5         value = "/stores")
6     List<Store> getStores();
7
8     @RequestMapping(method =
9         RequestMethod.POST,
10        value = "/stores/{storeId}",
11        consumes = "application/json")
12    Store update(
13        @PathVariable("storeId") Long storeId,
14        Store store);
15 }

```

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 configurações, 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.

```

1 @Configuration
2 public class FooConfiguration {
3     @Bean
4     public Contract feignContract() {
5         return new feign.Contract.Default();
6     }
7
8     @Bean
9     public BasicAuthRequestInterceptor
10    basicAuthRequestInterceptor() {
11        return new BasicAuthRequestInterceptor
12        ("user", "password");
13    }
14 }

```

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 micro-serviços, 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 utilização da mesma.

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

Isto significa que quando ocorrer uma chamada para a rota "/meusprodutos", o mesmo será encaminhado para o serviço

```

1 zuul:
2   ignoredServices: '*'
3   routes:
4     produtos: /meus-produtos/**

```

Figura 22. Zuul

```

1 zuul:
2   routes:
3     produtos:
4       path: /meus-produtos/**
5       serviceId: produtos_service

```

Figura 23. Controle refinado das rotas

produtos_service: Se for desejável especificar uma URL para uma localização física do serviço, pode ser feito conforme a figura 24.

```

1 zuul:
2   routes:
3     produtos:
4       path: /produtos/**
5       url:
6         http://exemplo.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.

```

1 zuul:
2   routes:
3     produtos:
4       path: /meus-produtos/**
5       serviceId: produtos
6
7   ribbon:
8     eureka:
9       enabled: false
10
11   produtos:
12     ribbon:
13       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 substituí-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.

```

1 zuul:
2 routes:
3   primeiro:
4     path: /primeiro/**
5     url: http://primeiro.exemplo.com
6   segundo:
7     path: /segundo/**
8     url: forward:/segundo
9   terceiro:
10    path: /terceiro/**
11    url: forward:/3rd
12  legacy:
13    path: /**
14    url: http://teste.exemplo.com

```

Figura 26. Atualização de Rotas

Ao processar as solicitações de entrada, parâmetros de consulta são decodificados para que os mesmos possam estar disponíveis para possíveis modificações nos filtro Zuul. Estes são recodificados ao 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 `zuul.forceOriginalQueryStringEncoding` definindo-a 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.

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

```

1 @RequestMapping(method =
2   RequestMethod.GET, value = "/multiple")
3 public Single<List<String>> multiple() {
4   return Observable
5     .just("multiple", "values")
6     .toList().toSingle();
7 }
8
9 @RequestMapping(
10   method = RequestMethod.GET,
11   value = "/responseWithObservable")
12 public ResponseEntity<Single<String>>
13   responseWithObservable() {
14
15   Observable<String> observable =
16     Observable.just("single value");
17   HttpHeaders headers = new HttpHeaders();
18   headers
19     .setContentType(APPLICATION_JSON_UTF8);
20   return new ResponseEntity<>(
21     observable.toSingle(),
22     headers, HttpStatus.CREATED);
23 }

```

Figura 27. Compilação do firmware

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 distribuídas. 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 distribuídas.

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 esta estrutura, quando existem muitos micro-serviços envolvidos. Mas utilizando-se de ferramentas para auxílio no desenvolvimento desta estrutura, torna-se possível a criação 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.

APÊNDICE A

PROOF OF THE FIRST ZONKLAR EQUATION

Appendix one text goes here.

APÊNDICE B

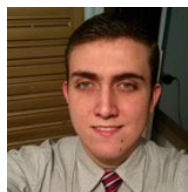
Appendix two text goes here.

ACKNOWLEDGMENT

The authors would like to thank...

REFERÊNCIAS

- [1] Paulo Silveira. Mauricio Balboa Linhares. *Tecnologias na Netflix*. 2017 [Online] Disponível: http://content.blubrry.com/hipsterstech/hipsters_041_netflix.mp3. [Acesso: 25-Abr-2017]
- [2] Flavio Silveira. *Microserviços - O que é?* 2016 [Online] Disponível: <http://flaviosilveira.com/2016/microservicos>. [Acesso: 06-Mai-2017]
- [3] Cleuton Sampaio. *Micro 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). *Microserviços em poucas palavras*. [Online] Disponível: <https://www.thoughtworks.com/pt/insights/blog/microservices-nutshell>. [Acesso: 20-Mar-2017]
- [5] Adriano Almeida. *Arquitetura 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. *Como 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. *Who is using microservices*. 2016 [Online] Disponível: <http://microservices.io/articles/whoususingmicroservices.html>. [Acesso: 20-Mai-2017]
- [10] IDC. *Connecting the IoT: The road to success*. 2016 [Online] Disponível: <http://www.idc.com/infographics/IoT>. [Acesso: 15-Fev-2017]
- [11] Martin Fowler et al. *Microservices*. 2014 [Online] Disponível: <https://www.martinfowler.com/articles/microservices.html>. [Acesso: 20-Mai-2017]
- [12] Netflix. *Ribbon*. 2016 [Online] Disponível: <https://github.com/Netflix/ribbon/wiki>. [Acesso: 06-Mai-2017]
- [13] Netflix. *Zuul*. 2016 [Online] Disponível: <https://github.com/Netflix/zuul/wiki>. [Acesso: 02-Mai-2017]
- [14] Martin Fowler. *CircuitBreaker*. 2014 [Online] Disponível: <https://martinfowler.com/bliki/CircuitBreaker.html>. [Acesso: 02-Mar-2017]
- [15] Danilo Sato. *CanaryRelease*. 2014 [Online] Disponível: <https://martinfowler.com/bliki/CanaryRelease.html>. [Acesso: 10-Abr-2017]
- [16] Netflix. *Spring 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. *Microservices: yesterday, today, and tomorrow*. Technical University of Denmark, vol 1, pp. 1-3, 2016
- [18] Willian Marques Freire. Munif Gebara Junior. *Artigo IoT - Internet das coisas*. vol 1, pp.1-3, 2017
- [19] ReactiveX. *Observable*. 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.

Integração entre IoT e Micro-serviços

Willian Marques Freire e Munif Gebara Junior

Resumo—The abstract goes here.

Index Terms—IEEE, IEEEtran, journal, LATEX, paper, template.

I. INTRODUÇÃO

COM A EVOLUÇÃO da computação distribuída surgiu a necessidade de criação de novos paradigmas, dando assim, origem ao Micro-serviço. 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. 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. Cada micro-serviço é construído em torno de uma única funcionalidade focada; Portanto, é mais eficaz. Desenvolvimento, extensibilidade, escalabilidade e integração são os principais benefícios oferecidos pela Arquitetura de Micro-serviços.

Atualmente tem surgido projetos utilizando este formato nos últimos anos e os resultados têm sido positivos, tanto que para muitos desenvolvedores o mesmo têm-se tornado a forma padrão de desenvolver aplicações. Entretanto, não existe muita informação que descreve o que são micro-serviços e como implementá-los [10]. Utilizando-se da empresa Netflix como referência em micro-serviços, esta provê muitos recursos gratuitos e de código aberto para desenvolvedores como 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 no ano de 2015 [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 seria movimentado em 2016 cerca de US\$ 41 bilhões [9].

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 produtos do mesmo incluem recursos sofisticados para roteamento de mensagens, coreografia, transformação e aplicação de regras de negócios. As aplicações construídas a partir de micro-serviços visam ser independentes e coesas, e estes são coreografados utilizando protocolos RestFul [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 vida útil mais longa 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 rede. Aqui é onde 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, termo que descreve imenso volumes de dado, se tornou um lugar comum e o mundo tecnológico começou a se mover para a economia de API. Este é o ponto onde o clássico SOA começou a mostrar problemas, demonstrando ser muito complicado, com centenas de interfaces e impossível definir granularidade [8].

Os micro-serviços hospedados em nuvem criaram um mo-

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.

delo 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, Eureka, uma Interface de comunicação Java para micro-serviços para a construção dos Micro-serviços e a plataforma de prototipagem eletrônica NodeMcu ESP8266 para desenvolvimento do IoT que se comunicará com os mesmos. 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.

I wish you the best of success.

mds

13 de Maio de 2017

A. Revisão Bibliográfica

1) NodeMCU:

B. Desenvolvimento

1) *IoT*: Neste trabalho, como o objeto é desenvolver a integração entre dispositivos IoT e Micro-serviços, primeiramente, será 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 Marques Freire e Gebara Júnior [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 distribuídas como o REST.

2) *Eureka Server*: Antes de começar o desenvolvimento dos micro-serviços, é necessário ter uma ferramenta que funcionará como gateway entre as aplicações. Para isto será 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 Marques Freire e Gebara Júnior [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 necessidade 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. Será 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 1, é criado uma thread, e nesta thread será instanciado uma classe Java chamada Server, que será um socket conectado ao broadcast.

```

1 @SpringBootApplication
2 @EnableEurekaServer
3 @RestController
4 @RequestMapping("/")
5 public class Application {
6     public static void main(String[] args)
7         throws InterruptedException {
8
9         Thread thread = new Thread(() -> {
10             System.out.println("————> thread");
11             Server server = new Server();
12             server.run();
13         });
14
15         thread.start();
16
17         SpringApplication.run(Application.class,
18             args);
19     }
20     @RequestMapping(value = "send-server",
21         method = RequestMethod.GET)
22     public String sendServer() {
23         return "Server send!";
24     }
25 }

```

Figura 1. Início da thread no Eureka Server

A classe que conterá o código do socket que manterá a conexão via broadcast, pode ser visto na figura 5. Esta classe ficará encarregada de manter a conexão na rede, esperando um dispositivo que busque se registra no Eureka Server.

4) *Registro do dispositivo*: Apartir do momento em que o serviço com 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.

A figura 2 demonstra como ficará a primeira função. Esta função está 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 feita uma verificação dos parâmetros, em busca do parâmetro ip, onde conterá o endereço do servidor. Isto pode ser visualizado na figura 3.

```

1 function waitRegister ()
2 local srvConfigure = net.createServer (net.TCP, 0)
3 srvConfigure:listen (
4   8000,
5   function (conn)
6     conn:on (
7       "receive",
8       function (conn, request)
9         print (request)
10        local buf = "ok";
11        local index =
12          string.find (request, "?ip=")
13        local novo = string
14          .sub (request, index)
15        local ip = string
16          .match (novo, "%d+%.%d+%.%d+%.%d+%.%d+%.%d*")
17
18        if (ip ~= nil) then
19          registered = true
20          registerEureka (wifi.sta.getip (), "http://"
21          .. ip)
22        end
23
24        conn:send (buf);
25        conn:close ();
26        collectgarbage ();
27      end)
28    end)
29 end

```

Figura 3. Espera do Eureka Server.

Após o NodeMCU ESP8266 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 4

```

1 function tryingToRegister ()
2   local registered = false
3   timeout = 0
4   tmr.alarm (2, 1000, 1,
5     function ()
6       if wifi.sta.getip () == nil
7         and wifi.sta.getbroadcast () == nil then
8         print ("IP unavailable, waiting... " ..
9         timeout)
10        timeout = timeout + 1
11        if timeout >= 20 then
12          print ("Deleting configuration...")
13          file.remove ("config.lua")
14          node.restart ()
15        end
16      else
17        if (pcall (function ()
18          ulala = net.createConnection (net.UDP, 0)
19          ulala:send (1234, wifi.sta
20            .getbroadcast (), "Request Address
21            Eureka"))
22          tmr.alarm (3, 1000, 1, function ()
23            if registered == false then
24              print ("Trying to register...")
25              local ulala = net.createConnection (
26                net.UDP)
27              ulala:send (1234, wifi.sta.
28                getbroadcast (),
29                "Request Address Eureka" .. tostring
30                (math.random (1,1000)))
31              ulala:close ()
32              ulala = nil
33            else
34              tmr.stop (3)
35            end
36          end)
37        end)) then
38          print ("Send register to Broadcast...")
39        else
40          print ("Error to send register in
41            Broadcast...")
42        end
43
44        print ("Connected, IP is " .. wifi.sta.
45        getip ())
46        tmr.stop (2)
47      end
48    end
49  )
50 end

```

Figura 4. Registro no Eureka.

A função que contém o protocolo para registra no Eureka Server pode ser visualizada na figura 6, e a função que criará um pequeno servidor para controle dos pinos digitais pode ser visto na figura 7.

C. Conclusão

1) sub2:

APÊNDICE A SOCKET COM BROADCAST

```

1 public class Server {
2
3     public static final int DEFAULT_PORT = 1234;
4     private DatagramSocket socket;
5     private DatagramPacket packet;
6
7     public void run() {
8         try {
9             socket = new DatagramSocket(DEFAULT_PORT
10 );
11         } catch (Exception ex) {
12             System.out.println("Problem creating
13 socket on port: "
14 + DEFAULT_PORT);
15         }
16
17         packet = new DatagramPacket(new byte[1], 1);
18
19         while (true) {
20             try {
21                 socket.receive(packet);
22                 System.out.println("Received from: "
23 + packet.getAddress() + ":" + packet.
24 getPort());
25                 byte[] outBuffer = new java
26 .util.Date().toString().getBytes();
27                 packet.setData(outBuffer);
28                 packet.setLength(outBuffer.length);
29                 socket.setBroadcast(true);
30                 socket.send(packet);
31
32                 Set<InetAddress> localAddress =
33 getLocalAddress();
34
35                 Set<String> ips = localAddress.stream()
36 .map(ad -> ad.getHostAddress())
37 .collect(Collectors.toSet())
38 .stream().sorted()
39 .collect(Collectors.toSet());
40
41                 RestTemplate template = new RestTemplate
42 ();
43
44                 ips.forEach(ip -> {
45                     try {
46                         template.exchange("http://"
47 + packet.getAddress().
48 getHostAddress()
49 .concat(":8000?ip={ip}"),
50 HttpMethod.GET,
51 HttpEntity.EMPTY,
52 Void.class,
53 ip.concat(":8000"));
54                     } catch (Exception e) {
55                         e.printStackTrace();
56                     }
57                 });
58
59                 System.out.println("Message -> " +
60 packet
61 .getAddress().getHostAddress());
62             } catch (IOException ie) {
63                 ie.printStackTrace();
64             }
65         }
66     }
67 }

```

Figura 5. Socket com Broadcast

APÊNDICE B PROTOCOLO PARA REGISTRO NO EUREKA SERVER

```

1 function registerEureka(ip, addressEureka)
2     print("Registering on Address Eureka "..
3 addressEureka.." the ip "..ip)
4     http.post(
5 addressEureka.."/eureka/apps/appID",
6 "Content-Type: application/json\r\n",
7 [[
8     {
9         "instance": {
10             "hostname": "]]..ip..[[",
11             "app": "]]..NodeMcuEsp8266"
12             ..node.chipid()..[[",
13             "ipAddr": "http://]]..ip..[[",
14             "status": "UP",
15             "port": {
16                 "@enabled": "true",
17                 "$": "8080"
18             },
19             "securePort": {
20                 "@enabled": "false",
21                 "$": "443"
22             },
23             "dataCenterInfo": {
24                 "@class": "com.netflix.appinfo
25 .InstanceInfo$DefaultDataCenterInfo",
26                 "name": "MyOwn"
27             },
28             "leaseInfo": {
29                 "renewalIntervalInSecs": 30,
30                 "durationInSecs": 90,
31                 "registrationTimestamp": 1492644843509,
32                 "lastRenewalTimestamp": 1492649644434,
33                 "evictionTimestamp": 0,
34                 "serviceUpTimestamp": 1492644813469
35             },
36             "homePageUrl": "http://]]..ip..[:8080/",
37             "statusPageUrl": "http://]]..ip..[:8080/info"
38         },
39         "healthCheckUrl": "http://]]..ip..[:8080/
40 health",
41         "vipAddress": "]]..NodeMcuEsp8266"..node.
42 chipid()
43         ..[[ " ]]]],
44         function(code, data)
45             if (code < 0) then
46                 print("HTTP request failed")
47             else
48                 gpio.mode(4, gpio.OUTPUT)
49                 gpio.write(4, gpio.LOW)
50                 print(code, data)
51             end
52         end
53     )
54 end

```

Figura 6. JSON Eureka

APÊNDICE C

SERVIDOR PARA CONTROLE GPIO

```

1 function registerServer()
2   srv:listen(
3     8080,
4     function(conn)
5       conn:on("receive", function(client, request)
6         local buf = [[
7           <!DOCTYPE html>
8           <html lang="pt-br">
9             <head>
10              <meta charset="UTF-8">
11              <title>NodeMcu ESP8266</title>
12            </head>
13            <body>
14              ]]
15            local _, _, method,
16            path, vars = string
17            .find(request, "([A-Z]+) (.+) ?(.+) HTTP");
18            if (method == nil) then
19              _, _, method, path =
20                string.find(request, "([A-Z]+) (.+) HTTP
21            ");
22            end
23            local _GET = {}
24            if (vars ~= nil) then
25              for k, v in string
26                .gmatch(vars, "(%w+)=(%w+)&*" ) do
27                _GET[k] = v
28              end
29            end
30            buf = buf.."<h1> ESP8266 Web Server</h1>";
31            for i=1,8,1
32            do
33              gpio.mode(i, gpio.OUTPUT)
34              buf = buf.."<p>DIGITAL " .. i
35              .. " <a href=\"?pin=" .. i .. "&stat=on\">
36                <button>ON</button>
37                </a>&nbsp;
38                <a href=\"?pin=" .. i .. "&stat=off\">
39                <button>OFF</button></a></p>"
40            end
41            if (_GET.stat ~= nil and _GET.stat ~= nil)
42            then
43              if (_GET.stat == "on") then
44                gpio.write(_GET.pin, gpio.HIGH);
45              else
46                gpio.write(_GET.pin, gpio.LOW);
47              end
48            end
49            buf = buf..[[
50              </body>
51              </html>
52            ]]
53            print(buf)
54            client:send(buf);
55          end)
56        conn:on(
57          "sent",
58          function(conn)
59            conn:close();
60            collectgarbage();
61          end
62        )
63      end
64    end
65  end
66 end
67 end
68 end
\begin{lstlisting}

```

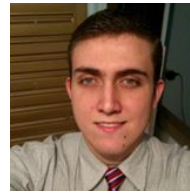
Figura 7. Servidor GPIO

ACKNOWLEDGMENT

The authors would like to thank...

REFERÊNCIAS

- [1] Willian Marques Freire. Munif Gebara Júnior. *IoT - A Internet das coisas*, vol. 1, 2017
- [2] Willian Marques Freire. Munif Gebara Júnior. *IoT - 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. Eureka at a glance. 2014 [Online] Disponível: <https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance> [Acesso: 26-Jun-2017]
- [5] Finep. Kevin 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. Netflix 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. NETFLIX 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. IoT and Microservices Architecture. [Online] Disponível: <http://www.happiestminds.com/blogs/iot-and-microservices-architecture>. 2016 [Acesso: 25-Abr-2017]
- [9] IDC. Connecting the IoT: The road to success. 2016 [Online] Disponível: <http://www.idc.com/infographics/IoT>. [Acesso: 15-Fev-2017]
- [10] Martin Fowler et al. Microservices. 2014 [Online] Disponível: <https://www.martinfowler.com/articles/microservices.html>. [Acesso: 20-Mai-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.

2 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

Toda pesquisa deve apresentar uma análise sobre a investigação que foi realizada através da metodologia que foi aplicada. Nesta sessão é interessante inserir tabelas, gráficos, imagens que mostrem os resultados, análise de dados coletados, etc.

É interessante que nessa sessão o autor compare os seus resultados com os resultados de outros trabalhos existentes. Essa comparação aumenta a qualidade do trabalho e demonstra a relevância do mesmo.

Nesta sessão o autor pode/deve incluir as contribuições científicas desenvolvidas tais como artigos, patentes, livros e outras contibuições que foram publicadas ou estão em fase de publicação e que são parte do trabalho.

3 CONCLUSÕES E TRABALHOS FUTUROS

A conclusão deve conter os principais aspectos e contribuições de forma a finalizar o trabalho apresentado. Deve-se apresentar o que era esperado do trabalho através dos objetivos inseridos inicialmente e mostrar o que foi conseguido.

Não deve-se inserir um novo assunto na conclusão. Aqui o autor apresentará as próprias impressões sobre o trabalho efetuado.

É importante também que sejam identificadas limitações e problemas que surgiram durante o desenvolvimento do trabalho e quais as consequências do mesmo.

Os trabalhos futuros devem conter oportunidades de expansão do trabalho apresentado, bem como, novos projetos que puderam ser vislumbrados a partir do desenvolvimento do trabalho

REFERÊNCIAS

A – ANEXOS E APÊNDICES 1

Anexos e apêndices são materiais adicionais, utilizados para complementar o texto, acrescentados ao final do trabalho, com a finalidade de esclarecimento ou de comprovação.

Apêndices são elaborados pelo autor e visam complementar uma argumentação. Os Anexos não são elaborados diretamente pelo autor e servem de fundamentação teórica, comprovação e ilustração (ex. mapas, leis, estatutos entre outros). Os apêndices devem aparecer antes dos anexos.