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 [7]. 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 [3]. 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 [4].

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 [1]. Segundo Ashton (Primeiro especialista a utilizar o termo "Internet das Coisas") [2] 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 [6].

Todas as evoluções tecnológicas na área de micro-serviços e IoT tem gerado grande interesse por parte dos desenvolvedores. Com base nas informações apresentadas, observa-se que são duas áreas distintas que crescem exponencialmente em razão do surgimento de novas tecnologias e têm-se necessidade de verificar relações que podem ser feitas entre as mesmas. Ao construir estruturas de comunicação entre diferentes processos, é visto que, muitos produtos e abordagens enfatizam a inserção de inteligências significativas no próprio mecanismo de comunicação. Um exemplo do que foi citado é o Enterprise Service Bus (ESB), onde os os produtos do mesmo incluem recursos sofisticados para roteamento de mensagens, coreografia, transformação e aplicação de regras de negócios. As aplicações construídas a partir de micro-serviços visam ser independentes e coesas, e estes são coreografados utilizando protocolos RestFul [7].

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 [5].

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 [5].

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 [**?**] foi desenvolvido uma estrutura IoT que permite a comunicação entre dispositivos utilizando o módulos Wi-Fi. É factível o desenvolvimento da comunicação entre estes dispositivos como os micro-serviços através de protocolos como HTTP, utilizando de tecnologias de comunicação entre aplicações distribuidas como o REST.

2) Eureka Server: Antes de começar o desenvolvimento dos micro-serviços, 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 [**?**]. No artigo Micro-serviços de Marques Freire e Gebara Júnior [**?**, p. 6], é explicado detalhadamente o processo de uma configuração simples do Eureka Server para descoberta de serviços.

3) Broadcast no Eureka Server: Um dos objetivos deste trabalho é a fácil configuração de registro dos dispositivos nos micro-serviços. Um dos problemas encontrados, é que o dispositivo não sabe onde está o Eureka Server, e devido a este fato, torna complicado o registro dos dispositivos. Foi analisado algumas estratégias para que o dispositivo descubra onde está o Eureka Server, e nota-se que neste trabalho não há a necessariedade dos dispositivos estar em uma rede diferente do Eureka Server. A solução proposta então, para que os dispositivos consigam se registrar no Eureka sem conhecer o mesmo, é utilizando o Broadcast. 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 **??**, 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,
     args);
18     }
19
20     @RequestMapping(value = "send-server",
21       method = RequestMethod.GET)
22     public String sendServer() {
23         return "Server send!";
24     }
25 }
```

Figura 1. Inicio da thread no Eureka Server

A classe que conterá o código do socket que manterá a conexão via broadcast, pode ser visto na figura **??**. 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 [**?**], 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 **??** demonstra como ficará a primeira função. Este função esta 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.

```
function startEureka()
  registerServer()
  waitRegister()
  tryingToRegister()
end
```

Figura 2. Função de tentativa de registro.

Assim que o dispositivo receber a requisição do Eureka Server, será feito uma verificação dos parâmetros, em busca do parâmetro ip, onde conterá o endereço do servidor. Isto pode ser visualizado na figura **??**.

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

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

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

Figura 3. Espera do Eureka Server.

Após o NodeMCU ESP82666 saber onde se encontra o gateway de serviços, será feito o registro no mesmo, e finalizado a conexão do dispositivo no endereço broadcast, conforme a figura **??**

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

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

Figura 4. Registro no Eureka.

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

C. Conclusão

1) sub2:

APÊNDICE A
SOCKET COM BROADCAST

```java
public class Server {

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

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

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

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

              Set<InetAddress> localAddress =
  getLocalAddress();

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

              RestTemplate template = new
  RestTemplate();

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

              System.out.println("Message ----> "
  + packet.getAddress().getHostAddress());

          } catch (IOException ie) {
              ie.printStackTrace();
          }
      }
  }
}
```

Figura 5. SOcket com Broadcast

APÊNDICE B
PROTOCOLO PARA REGISTRO NO EUREKA SERVER

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

Figura 6. JSON Eureka

APÊNDICE C
SERVIDOR PARA CONTROLE GPIO

```
function registerServer()
  srv:listen(
      8080,
      function(conn)

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

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

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

              print(buf)

              client:send(buf);
          end)

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

ACKNOWLEDGMENT

The authors would like to thank...

REFERÊNCIAS

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

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

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

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

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

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

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

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

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

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

PLACE PHOTO HERE

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.