XIII ERIC – (ISSN 2526-4230)

Eixo Temático – Administração e Tecnologias Virtuais – sala nº 27 (ARTIGO)

XIII ERIC – (ISSN 2526-4230)

Micro-serviços

Willian Marques Freire
Munif Gebara Junior (Orientador)

Artigo apresentado como trabalho de conclusão do curso de Bacharel em Ciência da Computação pela Fundação Faculdade de Filosofia, Ciências e Letras de Mandaguari – FAFIMAN.

RESUMO

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

1 INTRODUÇÃO

Micro-serviço é um novo paradigma que influencia diretamente o modo em que são desenvolvidas, e distribuídas as aplicações. Há certas características relacionadas à sua organização, à capacidade de negócio independentes, ao *deploy* automatizado, à inteligência e controle descentralizado de linguagens e de dados (LEWIS, 2016). Em uma publicação feita por Sampaio (2015), 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.

Para exemplificação sobre a motivação do uso de micro-serviços, pode-se citar os sistemas ERP (*Enterprise Resource Planning* ou sistemas para Planejamentos de Recursos Empresariais), que são desenvolvidos para cuidar de setores empresariais, desde o financeiro, recursos humanos, produção, estoque, dentre outros. Em um sistema para Planejamento de Recursos Empresariais, todas

as funcionalidades do mesmo são agrupadas dentro deste grande sistema, fazendo com que seja uma aplicação monolítica, ou seja, uma aplicação feita em somente uma unidade.

Por consequência, um dos principais pontos negativos, é que se houver um grande ponto de falha em algum módulo do sistema, isto poderá afetar a aplicação inteira, incluindo funcionalidades não relacionadas com o mesmo. Outro ponto negativo, é a base de código fonte, que se torna exponencialmente extensa de acordo com o tempo de desenvolvimento, tornando assim, novos membros do projeto improdutivo durante algum tempo, já que a complexidade do código é bem maior (ALMEIDA, 2015).

Um fato que ocorreu no ano de 2014, foi que o Docker (*container* portátil padronizado) começou a ser amplamente utilizado pela comunidade de desenvolvedores. Uma razão importante para sua utilização generalizada que Adrian (Membro e fundador da eBays Research Labs) observa, é sua portabilidade e aumento na velocidade com *container* de aplicações, que entregava algo em minutos ou horas e passou a entregar em segundos (DIEDRICH, 2015). Na figura 1 é apresentado sua utilização entre os anos 2012 e 2016.

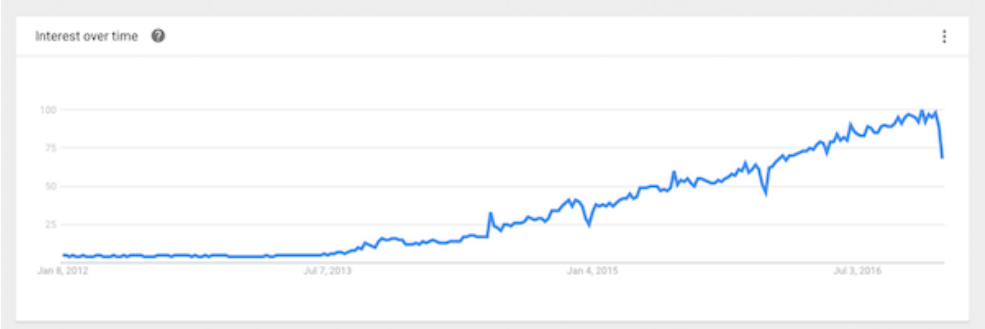


Figura 1 - Gráfico de utilização do Docker entre os anos 2012 e 2016.
Fonte: O que é Docker (DIEDRICH, 2015)

2 REVISÃO BIBLIOGRÁFICA

Segundo dados de Richardson (2016), diversas empresas estão utilizando micro-serviços, dentre as citadas estão: Comcast Cable, Uber, Netflix, Amazon, Ebay, SoundCloud, Karma, Groupon, Hailo, Gilt, Zalando, Lending Club, AutoScout24.

Os problemas associados ao desenvolvimento de software em larga escala ocorreram em torno da década de 1960. Posteriormente, na década de 1970 viu-se um enorme aumento de interesse da comunidade de pesquisa, para o design de software em suas aplicações e no processo de desenvolvimento. Nesta década, o design foi muitas vezes considerado como uma atividade não associada com a implementação em si, e portanto requerendo um conjunto especial de notações e ferramentas. Por volta da década de 1980, a integração do design nos processos de desenvolvimento contribuiu para uma fusão parcial dessas duas atividades, tornando assim mais difícil fazer distinções puras.

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

O advento e a difusão da orientação a objetos, a partir dos anos 80 e, em particular, a década de 1990, trouxe sua própria contribuição para o campo da Arquitetura de Software. O clássico por Gamma et ai. abrange a concepção de software orientado a objetos e como traduzi-lo em código que apresenta uma coleção de soluções recorrentes, chamados padrões. Esta ideia não é nova nem exclusiva à Engenharia de Software, mas o livro é o primeiro compêndio a popularizar a idéia em grande escala. Na era pré-Gamma, os padrões para soluções orientada a objetos já estavam sendo utilizado. Um exemplo típico de um padrão de projeto arquitetônico em programação orientada a objetos, é o *Model-View-Controller* (MVC), que tem sido um dos insights seminais no desenvolvimento precoce de interfaces gráficas de usuário (DRAGONI ET AL, 2016).

Cerca de sete anos atrás a empresa Netflix (provedora global de filmes e séries de televisão via streaming - distribuição de dados, geralmente de multimídia em uma rede através de pacotes) começou a migrar suas aplicações legadas para uma arquitetura baseada em APIs, (Interface de programação de aplicativos)

hospedadas na nuvem (local para armazenamento de dados online) da Amazon (empresa transnacional de comércio electrónico dos Estados Unidos com sede em Seattle), influenciando assim, o crescimento de uma ideologia na área de desenvolvimento de softwares que foi batizada pelo nome de “micro-serviço”.

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

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

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

ser retirada de um monte de aplicações, considerando que são independentes uns dos outros.

2.1. TECNOLOGIA PARA GERENCIAMENTO DE MICRO-SERVIÇOS

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

2.2. ZUUL

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

Em suma, esses filtros ajudam a executar as seguintes funções: Autenticação e segurança, identificação de requisitos de autenticação para cada recurso, negação de solicitações indesejadas, insights e monitoramento, rastreamento de dados significativos e estatísticas, a fim de dar uma visão precisa da produção, roteamento dinâmico, encaminhamento dinâmico solicitações para diferentes clusters de backend conforme necessário, *Stress Testing*, aumento gradual de tráfego para um *cluster*, a fim de avaliar o desempenho, *load Shedding*, alocação de capacidade para cada tipo de solicitação e liberação de pedidos que excedem o limite, manipulação de resposta estática e construção de respostas diretamente na ponta ao invés de encaminhá-las para um cluster interno.

Dentre os vários componentes que integram a biblioteca do Zuul, estão: *Zuul-core* que contém funcionalidades a fim de compilar e executar filtros, *Zuul-*

simple que demonstra como construir um aplicativo com *zuul-core* e *Zuul-netflix* que adiciona componentes Netflix utilizando Ribbon para solicitações de roteamento (NETFLIX, 2016).

2.3. RIBBON

Ribbon oferece suporte à comunicação entre processos na nuvem, e inclui balanceadores de carga desenvolvidos pela netflix. A tecnologia citada fornece os seguintes recursos: regras de balanceamento de carga múltiplas e conectáveis, integração com a descoberta de serviços, resiliência de falhas incorporada e clientes integrados com balanceadores de carga. O Ribbon é composto pelos seguintes projetos: *Ribbon-core* que inclui definições de interface e balanceamento de carga e cliente, implementações de balanceador de carga comuns, integração de cliente com balanceadores de carga e fábrica de clientes. *Ribbon-eureka* que inclui implementações do balanceador de carga com base no *Eureka-client* (biblioteca para registro e descoberta de serviços). *Ribbon-httpclient* que inclui a implementação de balanceamento de carga baseada em JSR-311 (NETFLIX, 2016).

2.4. 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.

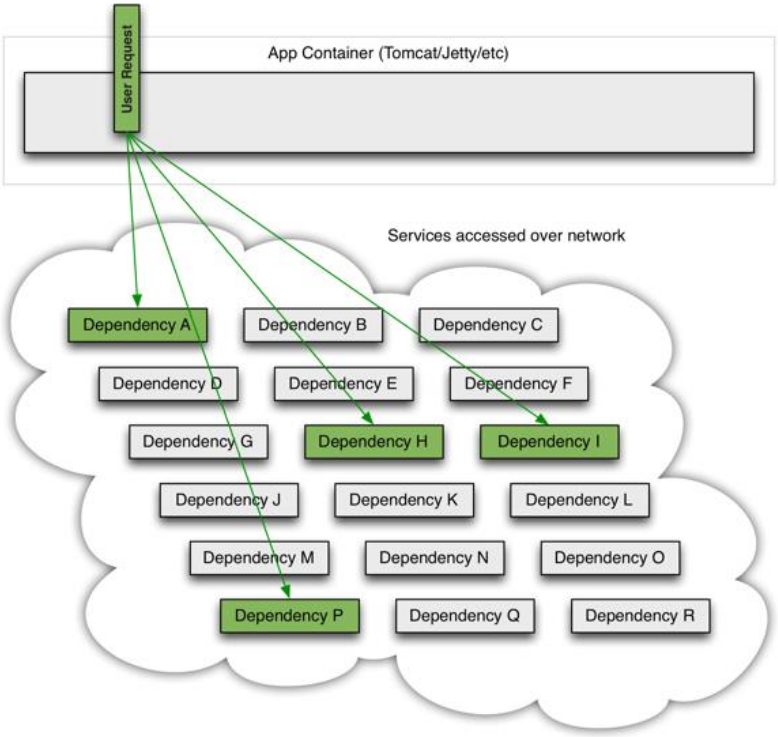


Figura 2 - *Wiki Hystrix (Internet Overtime)* - 2015.
Fonte: Github Hystrix

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

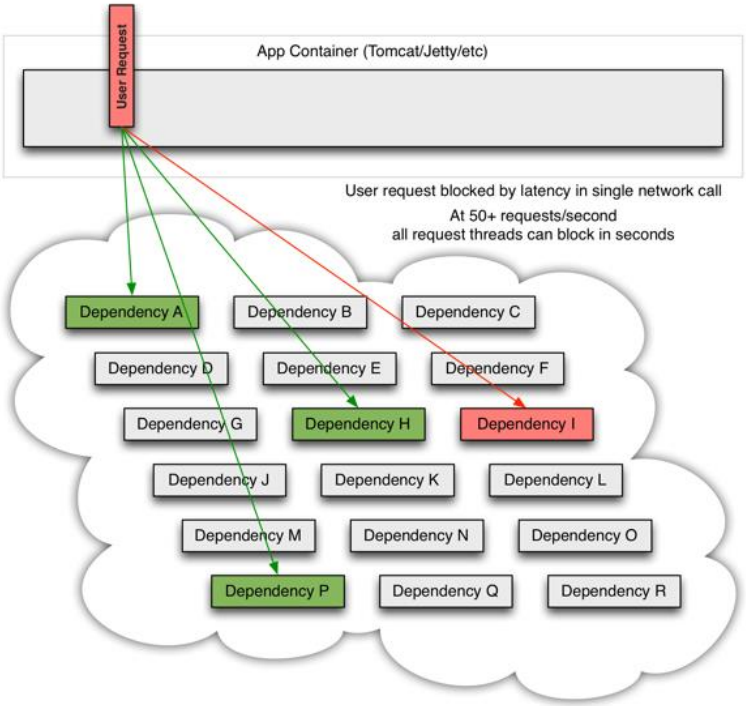


Figura 3 - *Wiki Hystrix (3 dimensions to Scaling)* - 2015.
Fonte: Github Hystrix

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

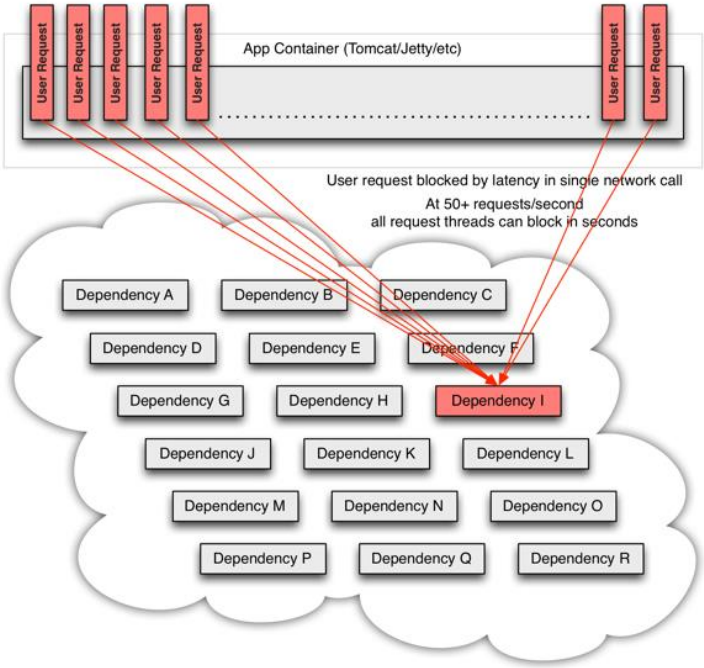


Figura 4 - *Wiki Hystrix (Container)* - 2015.
Fonte: Github Hystrix

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)

2.5. 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 encontram-se, os clientes *Eureka ou Service Discovery*, que é um dos princípios fundamentais de uma arquitetura baseada em micro-serviços. Configurar um micro-serviço é trabalhoso, pois envolve diversas técnicas de descoberta e registro de serviços, e com o Service *Discovery* da Netflix, torna-se eficiente este trabalho pois com poucas anotações Java consegue-se criar uma aplicação simples Eureka.

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

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

Na Netflix, além de desempenhar um papel crítico no balanceamento de carga de nível médio, o Eureka é utilizado para os seguintes fins: implementações

com Netflix Asgard, um serviço para fazer atualizações de serviços de forma rápida e segura, registro e exclusão de instâncias e transporte de metadados específicos de aplicativos adicionais sobre serviços. Dentre os motivos para utilizar o Eureka está o fato de que, o mesmo provê uma solução para balanceamento de carga round-robin simples, e quando não pode-se expor o tráfego das aplicações externamente com o AWS ELB, o Eureka resolve este problema.

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

2.6. 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.

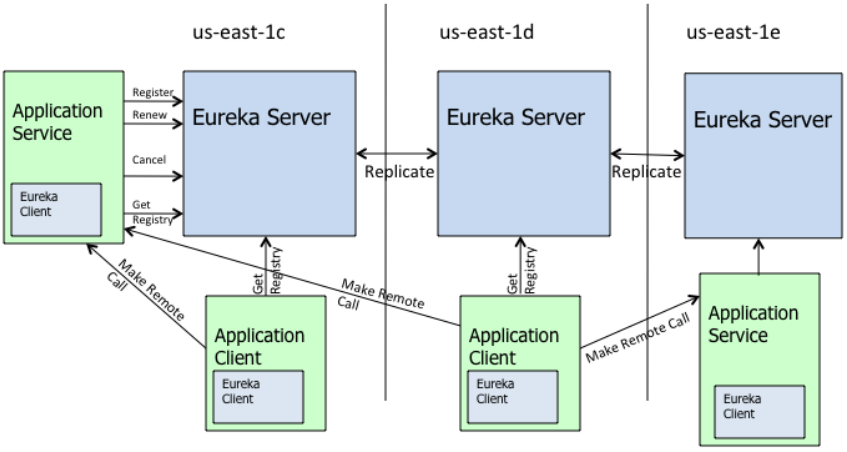


Figura 5 - *Wiki Eureka*.
Fonte: Github Eureka

3 DESENVOLVIMENTO

Neste capítulo serão descritas as etapas de desenvolvimento da estrutura IoT, no qual foi citado na introdução deste artigo.

3.1. INICIANDO COM EUREKA CLIENT

Este trabalho tem por objetivo 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.

Como resultado, quando um cliente se registra com o Eureka, ele fornece metadados sobre si, indicador de estado ou saúde, página inicial, dentre outros. O Eureka recebe mensagens *heartbeat* (disponibilidade) de cada instância pertencente a um serviço, e se algum heartbeat falhar, a instância é removido do registro.

Para inicializar um projeto com *Eureka Client*, será utilizado algumas anotações Java fornecidas pelo Eureka descritas a seguir: *@Configuration*, para utilizar recursos do projeto *Spring Config*, para facilitar configurações de projetos Spring baseado em Java, *@ComponentScan* para buscar componentes em pacotes java, *@EnableAutoConfiguration* para ativar as configurações automáticas internas, *@EnableEurekaClient* para ativar a descoberta de serviços do Eureka, *@RestController* para criar um controlador Rest *(Representational State Transfer)*, *@RequestMapping* para mapear as rotas da aplicação. As mesmas podem ser visualizadas na figura 6.

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

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

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

Figura 6 - Início de utilização do Eureka.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

3.2. INÍCIO DE UTILIZAÇÃO DO EUREKA

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

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

Figura 7 - Configuração do Eureka.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

Neste arquivo de configuração, encontra-se uma peculiaridade. O *DefaultZone* é a URL *(Uniform Resource Locator)* 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.

3.3. 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 funcione corretamente, deve ser feito 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.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

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.*

3.4. 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.*

3.5. BUSCA DE INSTÂNCIAS COM *EUREKA CLIENT*

O próximo passo para utilizar o Eureka Server para coreografar os micro-serviços, é utilizar o Eureka Client para descobrir instâncias do mesmo. Para fazer isto utilizando o framework, primeiramente é necessário incluir a dependência do *Eureka Client*, e criar um método que busque as instâncias registradas no Eureka conforme a figura 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.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

3.6. BUSCA DE INSTÂNCIAS COM DISCOVERY CLIENT

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

```
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 *Discovery Client*.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

3.7. PERFORMANCE DE REGISTRO NO EUREKA

Registrar um serviço no Eureka pode ser considerado um pouco lento, pelo fato de que, ser uma instância também envolve um heartbeat periódico para o registro, com duração padrão de 30 segundos. Um serviço não estará disponível para descoberta por clientes, enquanto uma 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.

3.8. 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 - Zonas .
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

3.9. PRIMEIRO 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 visualizado na figura 12.

```
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.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

3.10. MODO AUTÔNOMO

A combinação entre o cliente e servidor Eureka, e as pulsações para verificação de disponibilidade entre os mesmos, tornam o servidor Eureka resiliente à falhas, contanto que haja algum tipo de monitoramento para mantê-lo funcionando. No modo autônomo, pode-se 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.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

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 indica 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.

```
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.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

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.

3.11. CLIENTES HYSTRIX

Segundo Fowler Et. Al (2014), é comum que os sistemas de *software* façam chamadas remotas para *software* em execução em diferentes processos, provavelmente em máquinas diferentes em uma rede. Uma das grandes diferenças entre chamadas em memória e chamadas remotas é que, chamadas remotas podem falhar ou travar sem uma resposta, até que algum limite de tempo limite seja atingido. Devido a diversos problemas que podem ocorrer na arquitetura de micro-serviços, a Netflix criou a biblioteca chamada Hystrix, que implementa o padrão disjuntor que interrompe automaticamente o serviço quando ocorrem 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.

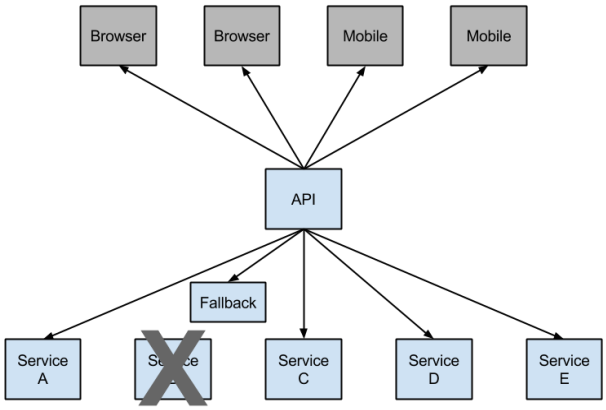


Figura 15 - *Fallback Hystrix* em falhas em cascata.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

3.12. PRIMEIROS PASSOS COM HYSTRIX

Para utilizar o Hystrix, é necessário a inclusão da 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.

```java
@SpringBootApplication
@EnableCircuitBreaker
public class Application {

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

}

@Component
public class StoreIntegration {

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

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

Figura 16 - Hystrix.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

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

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

Figura 17 - Dependências do Gerenciador Hystrix.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

3.13. 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 plausível 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.

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

Figura 18 - Ribbon.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

3.14. *FEIGN CLIENT*

Feign Client 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.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

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

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

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

Figura 20 - Mapeamento *FeignClient*.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

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*.

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

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

Figura 21 - Autenticação com *FeignClient*.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

3.15. 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 *"/api/usuario"* é mapeado para o serviço de usuário, *"/api/vendas"* 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 Sato (2014) 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 lançá-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.

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

Figura 22 - Zuul.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

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

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

Figura 23 - Controle refinado das r.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

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

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

Figura 24 - URL física do serviço.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

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

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

ribbon:
  eureka:
    enabled: false

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

Figura 25 - Cliente Ribbon.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

3.16. 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.

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

Figura 26 - Atualização de Rotas.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

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 aos reconstruir o pedido backend nos filtros de rota. O resultado pode ser diferente da entrada original, principalmente se o mesmo foi codificado 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.

3.17. RX JAVA

RxJava é uma implementação Java VM de extensões reativas : uma biblioteca para compor programas assíncronos e baseados em eventos utilizando sequências observáveis (REACTIVEX, 2015).

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 disso, cria uma sentinela na forma de assinante que está pronto para reagir apropriadamente em qualquer tempo futuro que o *Observable* gere (REACTIVEX, 2015). 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 estão dois exemplos, segundo a Netflix (2016), de como utilizá-los.

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

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

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

Figura 27 - RX JAVA.
Fonte: Spring Cloud Netflix (NETFLIX, 2016)

3.18. MOTIVAÇÃO DE USO DAS TECNOLOGIAS APRESENTADAS

Até o prezado momento, foram apresentados diversas tecnologias, entretanto, fica a questão sobre o porquê de tantas tecnologias. e a resposta é clara e objetiva. Para que se tenham micro-serviços, é necessário a utilização de um recurso para a orquestração dos mesmos, neste caso, entra o Eureka. Quando se tem micro-serviços, é necessário a centralização dos mesmos para que não fique espalhados e perdidos em hosts ou portas diferentes, surge a necessidade de um gateway, neste caso, o Zuul. A partir do momento que são integrados muitos micro-serviços, torna-se difícil o gerenciamento da configuração dos mesmos, surgindo a necessidade de um mecanismo para fácil configuração e integração de aplicações, por isto, foi apresentado o Spring Config. Após tudo isto, surge ainda um problema, o balanceamento de carga, um problema abrangente quando se trabalha com aplicações distribuídas. Surge então, a motivação de utilizar o Ribbon para distribuição de carga. Com tudo isso 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.

3.19. 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.

4 CONCLUSÕES E TRABALHOS FUTUROS

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

REFERÊNCIAS

SILVEIRA E LINHARES, Paulo. Mauricio B. Tecnologias na Netflix. 2017. Disponível em: <http://content.blubrry.com/hipsterstech/hipsters_041_netflix.mp3>. Acesso em 25 Abr. 2017

SILVEIRA, Flavio. Microserviços - O que é? 2016. Disponível em: <http://flaviosilveira.com/2016/microservicos>. Acesso em 06 Mai. 2017

SAMPAIO, Cleuton. Micro serviços: O que são e para que servem. 2015. Disponível em: <http://www.obomprogramador.com/2015/03/micro-servicos-o-que-sao-e-para-que.html>. Acesso em 28 Fev. 2017

LEWIS, James. Microsserviços em poucas palavras. 2016. Disponível em: <https://www.thoughtworks.com/pt/insights/blog/microservices-nutshell>. Acesso em 20 Mar. 2017

ALMEIDA, Adriano. Arquitetura de microserviços ou monolítica. 2015. Disponível em: <http://blog.caelum.com.br/arquitetura-de-microservicos-ou-monolitica>. Acesso em 22 Mar. 2017

DIEDRICH, Cristiano. O que é Docker. 2015. Disponível em: <http://www.mundodocker.com.br/o-que-e-docker>. Acesso em: 01 Dez. 2016

STENBERG, Jan. O estado da arte em micro serviços. 2015. Disponível em: <https://www.infoq.com/br/news/2015/04/microservices-current-state>. Acesso em 08 Fev. 2017

PELOI, Ricardo. Como implantar uma verdadeira Arquitetura de Microserviços na sua empresa. 2016. Disponível em: <http://sensedia.com/blog/soa/implantar-arquitetura-de-microservicos>. Acesso em 08 Mar. 2017

RICHARDSON, Chris. Whois using microservices. 2016. Disponível em: <http://microservices.io/articles/whoisusingmicroservices.html>. Acesso em 20 Mai. 2017

IDC, Idc. Connecting the IotT: The road to success. 2016. Disponível em: <http://www.idc.com/infographics/IoT>. Acesso em 15 Fev. 2017

FOWLER ET AL, Martin. Microservices. 2014. Disponível em: <https://www.martinfowler.com/articles/microservices.html>. Acesso em 20 Mai. 2017

NETFLIX, Netflix. Ribbon. 2016. Disponível em: <https://github.com/Netflix/ribbon/wiki>. Acesso em 06 Mai. 2017

NETFLIX, Netflix. Zuul. 2016. Disponível em: <https://github.com/Netflix/zuul/wiki>. Acesso em 02 Mai. 2017

FOWLER, Martin. CircuitBreaker. 2014. Disponível em: <https://martinfowler.com/bliki/CircuitBreaker.html>. Acesso em 02 Mar. 2017

SATO, Danilo. CanaryRelease. 2014. Disponível em: <https://martinfowler.com/bliki/CanaryRelease.html>. Acesso em 10 Abr. 2017

NETFLIX. Netflix. Spring Cloud Netflix. 2016. Disponível em: <http://cloud.spring.io/spring-cloud-netflix/spring-cloud-netflix.html>. Acesso em 18 Dez. 2016

DRAGONI ET AL, Nicola. Microservices: yesterday, today, and tomorrow. Technical University of Denmark, vol 1, pp. 1-3, 2016

FREIRE E JUNIOR, Willian M. Munif G. Artigo IoT - Internet das coisas. vol 1, pp.1-3, 2017

REACTIVEX, ReactiveX. Observable. 2015. Disponível em: <http://reactivex.io/documentation/observable.html>. Acesso em 25 Nov. 2016

XIII ERIC – (ISSN 2526-4230)

IoT – A Internet das coisas

Willian Marques Freire
Munif Gebara Junior (Orientador)

Artigo apresentado como trabalho de conclusão do curso de Bacharel em Ciência da Computação pela Fundação Faculdade de Filosofia, Ciências e Letras de Mandaguari – FAFIMAN.

RESUMO

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

Palavras-chave: IoT, Internet, Coisas.

1. INTRODUÇÃO

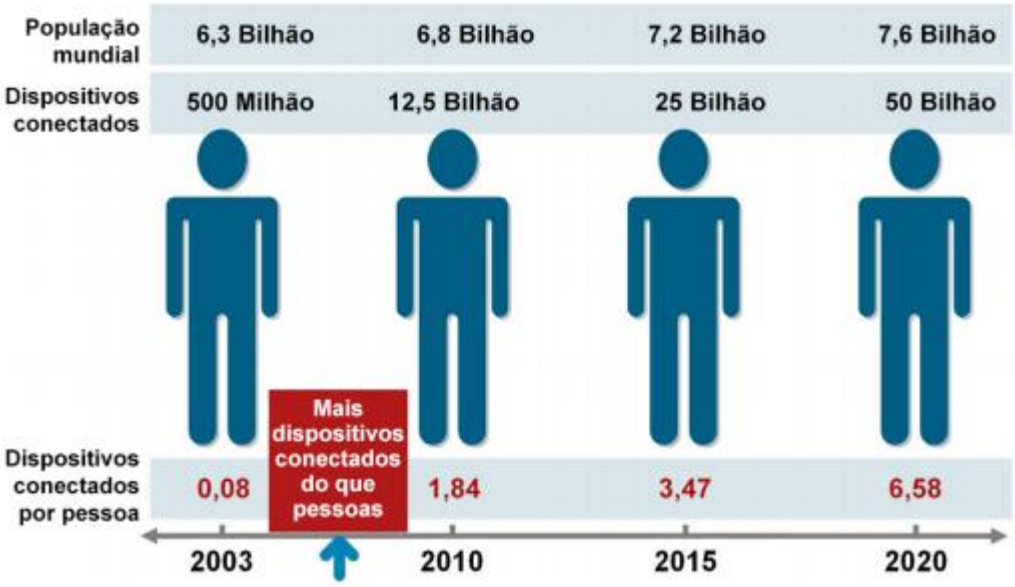
A Internet têm transcorrido por diversas etapas evolucionárias distintas. A primeira fase existente, foi quando a Web foi chamada de ARPANET *(Advanced Research Projects Agency Network)*. A segunda fase da Web, foi caracterizada pela concentração de todas as empresas, para compartilharem informações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

Facebook e Twitter se tornaram famosas e profícuas, ao permitir que pessoas se comuniquem e compartilhem informações (EVANS, 2011).

Considerando estas evoluções tecnológicas, IoT ou Internet das coisas, algumas vezes referida como a Internet dos objetos, também têm revolucionado, mudando o modelo de *hardware* computacional, que nasceu a aproximadamente 40 anos atrás. Dentre todas as fases diferentes de modelos de *hardware* que existiram, comprova-se que a cada revolução IoT, surge a necessidade de modificar este modelo (ESPOSITO, 2016). Devido a diversos estudos visando o melhor desenvolvimento e aproveitamento do modelo cliente-servidor, surgiram vários protocolos para transmissão de dados, e o protocolo principal utilizado atualmente é o HTTP *(Hypertext Transfer Protocol)*, encontrando-se por padrão em praticamente quase todos os navegadores atuais. Existem ainda outros protocolos como FTP *(File Transfer Protocol)* para transferência de arquivos, IMAP *(Internet Message Access Protocol)* para envio de Mensagens, entre outros.

Aproveitando-se destes protocolos, também surgiram especificações e arquiteturas, para que aplicações possam ser disponibilizadas como serviços. Dentre elas encontra-se o REST *(Representational State Transfer)* ou Transferência de Estado representacional, que segundo Fielding (2017), é uma abstração da arquitetura World Wide Web, sendo um estilo arquitetural, que consiste em um conjunto coordenado de restrições aplicadas a componentes, conectores e elementos de dados dentro de um sistema de hipermídia distribuído, todavia, é um assunto para um próximo trabalho.

Considerando que o IoT representa a próxima evolução da Internet, melhorando a capacidade de coletar, analisar e distribuir dados, faz-se com que os mesmos possam ser transformados em informação. Atualmente existem projetos IoT em desenvolvimento, que prometem fechar a lacuna entre ricos e pobres, melhorando a distribuição dos recursos mundiais para aqueles que precisam deles, e ajudando a entender a sociedade atual para que possa ser mais proativa e menos reativa (EVANS, 2011). Em 2003, haviam aproximadamente 6.3 bilhões de pessoas vivendo no mundo, e aproximadamente 500 milhões de dispositivos conectados à internet. O crescimento de smartphones e tablets, elevou o número de dispositivos conectados a Internet para aproximadamente 12,5 bilhões em 2010 (EVANS, 2011).



Anteriormente, em janeiro de 2009, uma equipe de pesquisadores chinesa, finalizou um estudo sobre os dados de roteamento da internet em intervalos de seis meses, e foi descoberto pelos mesmos, que a Internet dobra de tamanho a cada 5,32 anos (EVERS, 2003). Na figura 1, é possível observar o refinamento desta pesquisa. Devido a este crescimento no número de dispositivos conectados a internet, no artigo Internet of Things volume III, publicado pela empresa Dzone (empresa que faz publicações online sobre tecnologia) no ano de 2017, foi feito uma

Figura 1. "Nascimento"do IoT entre 2008 e 2009
Fonte: The Internet of Things How the Next Evolution of the Everything (EVANS, 2011) Internet Is Changing Everything,.

pesquisa interna na área de IoT. Na mesma, encontram-se 797 profissionais que são responsáveis por esta área, e foi questionado sobre as maiores preocupações quando se trata de IoT. O centro de preocupação estava pela segurança e privacidade, pois os mesmos, estavam interessados em IoT para o contexto empresarial de suas empresas. A falta de padrões, era um terço na coluna "muito preocupada" chegando a 34% dos envolvidos. Dentre eles, 25% dos entrevistados também desconfiam da conectividade e do baixo custo de energia, sendo que 24% se preocupam com a manutenção de hardware e *software*, e 14% estão apreensivos com o desenvolvimento imaturo e paradigmas de redes (EVANS, 2011).

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 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 pela Internet. Este artigo está organizado da seguinte forma: primeiramente será feito uma revisão bibliográfica sobre os assuntos que serão tratados, posteriormente será desenvolvido uma estrutura IoT que vise a fácil configuração da mesma, será feito também a apresentação de resultados e por fim a conclusão deste artigo.

2. REVISÃO BIBLIOGRÁFICA

Neste capítulo serão descritas as tecnologias que serão utilizadas para o desenvolvimento desta estrutura IoT.

a. IOT (INTERNET OF THINGS)

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 dispositivos. Dentre os mesmos, 53% preferem Rapberry Pi, 28% preferem o Arduino, e 19% não apresentam nenhuma preferência. Nesta pesquisa, foram relatados os 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 (2016), é um protocolo de mensagens leve, criado para comunicação M2M *(Machine to Machine)*, que obteve 33% de preferência de utilização em produção (LAWRENCE, 2016).

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 podem existir, a

vulnerabilidade de um dispositivo, pode ser alvo para ataques e invasões. Uma das maneiras de gerenciar a testabilidade do software no dispositivo, é testar isoladamente. Reproduzir o ecossistema em torno do dispositivo e como os usuários interagem garantem uma boa testabilidade. Estudos avançados reconhecem que não se podem identificar, e muito menos testar cada possível cenário de falha. Por este fato, estes estudos concentram cada vez mais na confiabilidade e segurança IoT. O último desafio para esta área, são as identificações e correções de falhas. Tomar ações rapidamente para resolução de problemas, é um aspecto fundamental no desenvolvimento de dispositivos IoT. Técnicas para integração contínua são utilizados para resolver estes problemas, e uma delas é a *Canary Release*, que Segundo Sato (2014), é uma técnica para reduzir o risco de introduzir uma nova versão de software em produção, lançando mudanças lentamente para um pequeno subconjunto de usuários, antes de lançá-la em toda a infra-estrutura e torná-la disponível para todos (LAWRENCE, 2016).

Nas últimas duas décadas, houveram avanços tecnológicos na área computacional, surgindo processadores, armazenamentos, memória e dispositivos de rede com baixo custo e mais capacidade tecnológica. Atualmente, dispositivos físicos estão sendo desenvolvidos com mais capacidade computacional, e interligados através da internet de maneira efetiva. A adoção generalizada das tecnologias IoT, enriquecem a idéia de computação ubíqua, um conceito que surgiu no final dos anos 80. Mark Weiser, criador deste conceito, escreveu em seu artigo The Computer for the 21st Century, que o computador se integra a vida das pessoas de modo que elas não o percebam, todavia, o utilizam. Motivado por esta convicção, Weiser percebeu que em sua época não haviam tecnologias necessárias para que a Computação Ubíqua se tornasse realidade, fazendo com que assim, dedicasse esforços para desenvolver estes meios (WEISER, 1991).

Anteriormente, 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 (KENNEDY, 1926).

Além disto, em um artigo feito pelo professor Michael Wooldridge (chefe do Departamento de Ciência da Computação da Universidade de Oxford), definiu um termo utilizado no meio tecnológico, que são os agentes. Segundo ele, um agente é um computador que está situado em algum ambiente, capaz de realizar ações autônomas sobre este, para cumprir seus objetivos delegados. O mesmo torna-se inteligente com as seguintes propriedades: Reatividade (a percepção de seu ambiente, e a capacidade de resposta em tempo hábil sobre mudanças que ocorrem), proatividade (antecipação de problemas futuros) e habilidade social (capacidade de interação entre agentes para satisfazer seus objetivos de design) (WOOLDRIGE e JENNINGS, 1995).

b. 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 outros dispositivos. NodeMCU é um firmware baseado em eLua (implementação completa utilizando programação Lua para sistemas embarcados) (ELUA, 2017). O mesmo foi projetado para ser integrado com o *Chip Wi-Fi ESP8266* desenvolvido pela empresa Espressif, situada em Shangai, especializada no ramo de IoT (SYSTEMS, 2016). 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 (STEWART, 2016) e o demais existente em código escrito na linguagem Lua - criada em 1993 por Roberto Ierusalimschy, Luiz Henrique de Figueiredo e Waldemar Celes (LUA, 2017).

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 3,3 V. Comparado ao Arduino UNO, o NodeMCU se destaca por ter um processador Tensilica LX106, que pode atingir até 160 MHZ, e possui uma memória RAM de 20 KB e uma memória flash de 4MB. Já o Arduino UNO possui um microcontrolador de 16 MHZ, possui uma memória RAM de 2 KB e uma memória flash de 32 KB. 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 (LIMA, 2016).

c. MÓDULO WI-FI

Devido ao elevado número de cabos necessários para interconectar computadores e dispositivos, surgiu o WiFi. Atualmente, ainda são 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 comum 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 o estabelecimento de conexões nas seguintes velocidades de transmissão: 1 Mb/s, 2Mb/s, 5,5 Mb/s e 11 Mb/s. O padrão 802.11g que surgiu em 2003 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 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 por fim, o 802.11ac, também chamado 5G WiFi, sua principal vantagem está em sua velocidade, estimada em até 433 Mbps no modo mais simples, fazendo com que, seja 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"* (ALECRIM, 2013).

d. SEGURANÇA WI-FI

Apesar de todas as facilidades que se encontram ao utilizar tecnologias sem fio como Wi-Fi, assim como todo sistema, medidas de segurança devem ser utilizadas para prevenir ataques e invasões. Existem protocolos de segurança que auxiliam na proteção de conexões Wi-Fi, dentre eles se encontram: WEP, WPA, e o WPA2. O *Wired Equivalent Privacy* (WEP) é um algoritmo de segurança que foi criado em 1999 e é compatível com praticamente todos os dispositivos Wi-Fi disponíveis 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 ataque. *Wi-Fi Protected Access* ou WPA, surgiu quando o WEP saiu de circulação, e entrou como protocolo-padrão industrial. Adotado em 2003, trazia como novidade a encriptação 256 bits e como segurança adicional, fazia análise de pacotes, para verificar alterações e inovações. 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 a utilização de duas tecnologias que são utilizadas neste algoritmo, o AES *(Advanced Encryption Standard),* que é um novo padrão para segurança das informações, e o CCMP *(Counter Cipher Mode)*, um mecanismo de encriptação que protege os dados que passam pela rede. Devido a complexidade do mesmo, muitos dispositivos, mesmos recentes, não são compatíveis com ele (DEMARTINI, 2013).

e. SISTEMA DE ARQUIVOS

O NodeMCU utiliza o sistema de arquivos SPIFFS *(SPI Flash File System)*, um sistema de arquivos destinado a dispositivos *flash* embarcados. SPIFFS é projetado para projetos que são pequenos e memória sem pilha (ANDERSON, 2013). O projeto é aberto e pode ser clonado de seu repositório no github.

f. FERRAMENTA PARA DESENVOLVIMENTO

Neste projeto será utilizado 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 (ESPLORER, 2013). Na figura 2 é possível observar estas características.

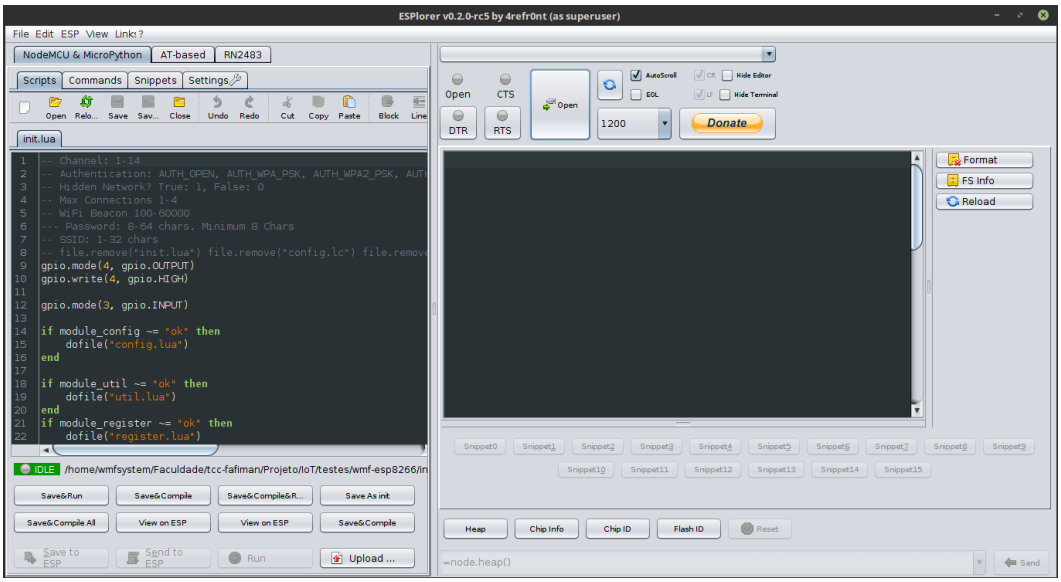


Figura 2 - ESPlorer
Fonte: Autoria Própria.

g. PROTOCOLOS TCP E UDP

O protocolo TCP ou Protocolo de controle de transmissão, é um dos 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 - fornece serviços aos clientes, apresentação que fornece encriptação e compressão de dados, além de garantir a compatibilidade entre camadas de aplicação de sistemas diferentes, sessão - controla as sessões entre aplicações, transporte - controla o fluxo de informação e controle de erros, rede - encaminha pacotes e possui esquema de endereçamentos, dados - 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 (PINTO, 2017) (CERF E KAHN, 1974).

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 (MARINE ET AL, 2017) é uma entidade de dados completa e independente que contém informações suficientes para ser roteada dá 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 seja enviado pela aplicação um datagrama encapsulado em um Pacote IPv4 (ou IPv6), e então enviado ao destino. Ao contrário do TCP, o UDP não possui qualquer tipo de garantia que o pacote será entregue.

3. DESENVOLVIMENTO

Neste capítulo serão descritas as etapas de desenvolvimento da estrutura IoT, no qual foi citado na introdução deste artigo.

a. CONFIGURAÇÕES INICIAIS

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 no github. No caso deste projeto, o *build* do mesmo será feito manualmente. Sendo assim, 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 serão utilizadas.

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

Figura 3 – Primeira etapa da compilação do *firmware*
Fonte: Autoria Própria.

Duas possibilidades disponíveis 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, deve-se 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 onde é definido a constante *LUA_NUMBER_INTEGRAL* dentro do arquivo *user_config.h*. .

b. GERANDO E GRAVANDO O FIRMWARE

Após ajustar as devidas configurações iniciais, 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. A mesma foi iniciada por Fredrik Ahlbeg, um dos envolvidos no projeto, e atualmente é

Figura 4 - Compilação do firmware
Fonte: Autoria Própria.

mantido por Angus Gratton e pela comunidade. Posteriormente, para compilar os códigos fontes, deve-se primeiramente executar os seguintes comandos dentro do diretório do projeto, conforme a figura 4.

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

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

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

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

Figura 5 - Compilação do firmware
Fonte: Autoria Própria.

Como este projeto utiliza o docker, uma infraestrutura independente de plataforma (DOCKER, 2017), primeiramente é atualizado o container e posteriormente é executado um comando para compilar o projeto.

Após compilado, deve ser executado os comandos conforme a figura 5, utilizando a ferramenta *esptool*. Estes comandos serão utilizados para gravar o *firmware* no ESP8266.

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

c. 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 da mesma, onde será selecionado 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 (MAX, 2011). 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*
Fonte: Autoria Própria.

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
Fonte: Autoria Própria.

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

Para configurar um Access Point, é necessário algumas considerações como: *ssid* (Nome da conexão), *pwd* (Senha da 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 mesmo, e neste trabalho, está sendo salvo no padrão usuário e senha separados por um espaço em branco. Os demais códigos desenvolvidos, podem ser encontrados no apêndice deste trabalho. No arquivo *init.lua* deverá ser feito a importação dos mesmos, pois serão divididos da seguinte forma: *init.lua* (arquivo inicial), *config.lua* (arquivo que terá o código de configuração) e *util.lua* (arquivo que terá funções utilitárias). Para importar os demais módulos, foi feito conforme a figura 8.

```
1 dofile("config.lua")
2 dofile("util.lua")
3 dofile("register.lua")
```

Figura 8 – Importação dos módulos
Fonte: Autoria Própria.

d. RESULTADOS

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

Como este dispositivo tem suporte a protocolos TCP e UDP, é aberto possibilidades para integração com outros dispositivos ou sistemas que utilizam este protocolo. Foram realizados diversos testes com o mesmo, dentre eles utilizando o GPIO *(General Purpose Input/Output)* para utilização de pinos digitais, SJSON para serialização de Objetos utilizados textos no padrão JSON (NODEMCU, 2017) e o Timer para executar funções em determinado tempo (NODEMCU, 2014).

4. CONCLUSÕES E TRABALHOS FUTUROS

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

ser resolvidos acoplando estas tecnologias, integrando o dispositivo com a rede de internet de forma fácil, foi resolvida.Apêndices

APÊNDICE A - Código fonte configuracional: config.lua

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

          local _GET = getParamsUrl(payload)

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

              for i = 0, tablelength(ap), 1 do
                if ap[i] ~= nil then
                  if (string
```

```
                        .find(ap[i], _GET.ssid)) then
                        user = ap[i]
                    end
                  end
                end

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

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

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

APÊNDICE B - Código fonte utilitário: util.lua

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

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

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

```
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]+) (.+)
    HTTP")
57     end
58     local _GET = {}
59     if (vars ~= nil) then
60         for k, v in string
61           .gmatch(vars, "(%w+)=(%w+)&*") do
62             _GET[k] = unescape(v)
63         end
64     end
65     return _GET
66 end
```

A

PÊNDICE C - Código fonte inicial: init.lu

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

REFERÊNCIAS

ALECRIM, Emerson. O que é Wi-Fi (IEEE 802.11)? 2013. Disponível em: <https://www.infowester.com/wifi.php#80211>. Acesso em 20 Mai. 2017

ANDERSON, Peterson. SPIFFS (SPI Flash File System). 2013. Disponível em: <https://github.com/pellepl/spiffs>. Acesso em 11 Jun. 2017

CERF E KAHN, Vinton G. e Robert E. A Protocol for Packet Network Intercommunication, IEEE Transactions on Communications, vol. 22, pp. 637-648, 1974

DEMARTINI, Felipe. WEP, WPA, WPA2: o que as siglas significam para o seu WiFi? 2013. Disponível em: <https://www.tecmundo.com.br/wi-fi/42024-wep-wpa-wpa2-o-que-as-siglas-significam-para-o-seu-wifi-.htm>. Acesso em 03 Jun. 2017

DOCKER, Docker. Disponível: <https://www.docker.com>. Acesso em 11 Jun. 2017

ELUA. What is eLua. Disponível em: <http://www.eluaproject.net/overview>. Acesso em 19 Nov. 2017

ESPOSITO, John. The Dzone Guide to Internet of Things, vol 3, pp. 1-6, 2016

EVANS, Dave. The Internet of Things How the Next Evolution of the Internet Is Changing Everything, vol 1, pp. 2-4, 2011

ESPLORER, ESPlorer. Integrated Development Environment (IDE) for ESP8266 developers . 2013. Disponível em: <https://github.com/4refr0nt/ESPlorer>. Acesso em 11 Jun. 2017

FIELDING, Roy Thomas. Representational State Transfer (REST). Disponível em: <http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>. Acesso em 29 Mai. 2017

FILIPEFLOP. Controle e monitoramento IoT com NodeMCU e MQTT. 2016. Disponível em: <http://blog.filipeflop.com/wireless/controle-monitoramento-iot-nodemcu-e-mqtt.html>. Acesso em 10 Abr. 2017

KENNEDY, John B. WHEN WOMAN IS BOSS, 1926 Disponível em: <http://www.tfcbooks.com/tesla/1926-01-30.htm>. Acesso em 10 Mai. 2017

LAWRENCE, Cate. Internet of Things Applications, Protocols, and Best Practices, vol 4, pp. 1-10, 2017

LIMA, Irving Souza. NodeMCU (ESP8266) o módulo que desbanca o Arduino e facilitará a Internet das Coisas. 2016. Disponível em: <http://irving.com.br/esp8266/nodemcu-esp8266-o-modulo-que-desbanca-o-arduino-e-facilitara-a-internet-das-coisas> Acesso em 05 Jun. 2017

LUA, Authors. Disponível: http://www.lua.org/authors.html. Acesso em 5 Fev. 2017

MAX, Mundo. Qual a diferença entre um Roteador Wireless e um Access Point? 2011. Disponível em: <http://www.mundomax.com.br/blog/informatica/qual-a-diferenca-entre-um-roteador-wireless-e-um-access-point>. Acesso em 03 Jun. 2017

MARINE ET AL. April N. RFC 1594, 1994. Disponível em: <https://tools.ietf.org/html/rfc1594>. Acesso em 17 Jun. 2017

NODEMCU. NodeMCU. GPIO Module? 2014. Disponível em: <https://nodemcu.readthedocs.io/en/master/en/modules/gpio> Acesso em 17 Jun. 2017

NODEMCU. NodeMCU. SJSON Module. 2017. Disponível em: <https://nodemcu.readthedocs.io/en/master/en/modules/sjson>. Acesso em 17 Jun. 2017

NODEMCU. NodeMCU. Timer Module. 2014. Disponível em: <https://nodemcu.readthedocs.io/en/master/en/modules/tmr>. Acesso em 17 Jun. 2017

NODEMCU. NodeMCU. NodeMCU 2.0.0. Disponível em: <https://github.com/nodemcu/nodemcu-firmware>. Acesso em 10 Dez. 2017

PINTO. Pedro. Redes – Sabe o que é o modelo OSI? 2010. Disponível em: <https://pplware.sapo.pt/tutoriais/networking/redes-sabe-o-que-e-o-modelo-osi>. Acesso em 17 Jun. 2017

SATO, Danilo. CanaryRelease. 2014. Disponível em: <https://martinfowler.com/bliki/CanaryRelease.html>. Acesso em 10 Abr. 2017

STEWART, William. C Programming Language History. Disponível em: <http://www.livinginternet.com/i/iw_unix_c.htm>. Acesso em 15 Mai. 2017

SYSTEMS, Expressif. Disponível em: <http://espressif.com/company/contact/pre-sale-questions>. Acesso em 11 Nov. 2016

WEISER, Mark. The Computer for the 21st Century, vol 1, pp. 1-2, 1991

WOOLDRIGE e JENNINGS, Michael e Nicholas R. Intelligent agents: theory and practice, vol 1, pp. 1-3, 1995

XIII ERIC – (ISSN 2526-4230)

ADMINISTRAÇÃO FINANCEIRA E ORÇAMENTÁRIA

VERIDIANA DOS SANTOS FRANÇA BARROS
LUCIANE OLIVEIRA DA SILVA BEM

INTRODUÇÃO

1.1 IMPORTÂNCIA E RELEVÂNCIA DO TEMA

.

> A Administração Financeira diz respeito às atribuições dos administradores financeiros nas empresas. Os administradores financeiros são responsáveis pela gestão dos negócios financeiros de organizações de todos os tipos – financeiras ou não, abertas ou fechadas, grandes ou pequenas, com ou sem fins lucrativos. Eles realizam as mais diversas tarefas financeiras, tais como planejamento, concessão de crédito a clientes, avaliação de propostas que envolvam grandes desembolsos e captação de fundos para financiar as operações da empresa. (GITMAN, 2013, p. 4).

Para que os administradores possam por em prática todas as suas atribuições ele necessita de parâmetros de orçamento, para que sua gestão possa ser a mais lucrativa e também atender com maior eficiência as necessidades do mercado.

> O orçamento empresarial é a tradução das prováveis operações necessárias para empresas atingir os objetivos esperados e está intimamente ligado às estratégias e planos da diretoria administrativa, ou seja, é importante termos o conhecimento da evolução histórica do pensamento estratégico. (BERTI, 2010, P.20)

Para que as tomadas de decisões sejam mais corretas possíveis o administrador também precisa de um planejamento, para que não seja surpreendido em determinadas situações. E sem um planejamento prévio as empresas estão fadadas ao fracasso ou encaminhadas a ele.

> Uma das áreas mais importantes da administração é a estratégia. Pensar sobre estratégia empresarial sem pensar ao mesmo tempo sobre estratégia financeira é receita exceleme para o desastre.
> Consequentemente os estrategistas precisam ter clara compreensão das implicações financeiras de seus planos estratégicos. Em termos gerais esperam-se que todos os tipos de administradores tenham grande entendimento de como seus negócios afetam a lucratividade e que sejam

> capazes de aumentar a lucratividade de suas áreas. Isso é precisamente o que o estudo de finanças lhe ensina. (ROSS, p. 37 E 38)

Muitas empresas encontram dificuldades em identificar e analisar sua real situação no cenário econômico, dificultando qualquer tipo de tomada de decisão, seja ela a curto, médio ou longo prazo. É extremamente necessário a capacitação de pessoal para que interpretem as informações fornecidas por todos os setores da empresa, analisem e identifiquem os possíveis problemas e assim dar parâmetros para tomada de decisões imediatas ou futuras.

Na atual situação econômica social, muitas organizações encontram dificuldades em gerenciar, analisar, tomar as decisões corretas em relação aos empreendimentos, esse tipo de situação é decorrente de problemas internos, pouco conhecimento em relação à concorrência, por isso é necessário um treinamento de gerente financeiro, auxiliares, e até de diretores financeiros para que a tomada de decisão seja baseada em informações filtradas e selecionadas, tornando a decisão a mais correta possível.

Finanças é dividida em áreas e fornece oportunidades, sendo elas: serviços financeiros e administração financeira.

O serviço financeiro trata-se de uma concepção e oferta de assessoria e produtos financeiros a pessoas físicas, fornece varias oportunidades de carreira.

Administração financeira é exercida por um administrador financeiro e está frequentemente ligada a um alto executivo de uma empresa geralmente denominado diretor financeiro ou vice- presidente de finanças.

1.2 JUSTIFICATIVA DA ESCOLHA DO TEMA

O tema abordado é de extrema importância para a rotina de qualquer empresa, trata-se de um campo flexível e ajustável às mudanças econômicas. Seu conhecimento é essencial para que administradores realizem seu trabalho com eficiência e adequando suas diretrizes com a constante globalização.

Para escolha do tema foram considerados processos essenciais na organização por fazer o controle dos setores de contas a pagar, receber, movimentação bancaria e auxiliando no processo de controle e tomada de decisão e

assim contribuir para uma melhor administração e conseguir alcançar melhores resultados.

A maximização do lucro é um objetivo comum entre os proprietários das organizações, e por isso os administradores financeiros têm essa incumbência e para que isso ocorra o administrador financeiro escolhera aquela que trará maior rentabilidade.

Não apenas as organizações e seus administradores, os princípios básicos da administração financeira também são aplicáveis na vida pessoal, em transações bancarias, de compra e venda, na obtenção de empréstimos e investimentos. Seu campo é amplo e dinâmico afetando assim a vida de todas as pessoas, e o conhecimento mesmo básico dos princípios da administração financeira auxiliam na vida de pessoas físicas.

1.3 PROBLEMÁTICA

Como se encontra estruturada e organizada a área/função financeira dentro de uma organização?

1.4 OBJETIVOS

1.4.1 Objetivo Geral

Compreender como a área/função financeira, encontra-se estruturada e organizada na empresa Metais Globo indústria e Comercio Ltda.

1.4.2 Objetivos Específicos

a) Identificar a existência da área/função financeira na empresa em estudo, bem como sua nomenclatura;

b) Apontar as atividades realizadas dentro da área/função financeira, destacando os tipos de controles administrativos e financeiros utilizados;

c) Verificar os profissionais que cuidam das atividades da área/função financeira, apresentado a nomenclatura do cargo e o perfil do ocupante;

d) Levantar a necessidade de melhoria ou mudança em diferentes áreas da organização, a fim de apresentar projetos que atendam tais necessidades;

e) Realizar um estudo de viabilidade financeira dos projetos apresentados, destacando o mais viável.

1.5 METODOLOGIA

Pesquisa de campo, exploratória, descritiva bibliográfica.
A pesquisa exploratória, designada por alguns autores como pesquisa quase científica ou não científica, é normalmente o passo inicial no processo de pesquisa pela experiência e um auxílio que traz a formulação de hipóteses significativas para posteriores pesquisas.

A pesquisa descritiva busca descrever as características de determinada populações ou fenômenos. Uma de suas peculiaridades está na utilização de técnicas padronizadas de coleta de dados, tais como o questionário e a observação sistemática. Ex.: pesquisa referente à idade, sexo, procedência, eleição etc.

A pesquisa bibliográfica apresenta uma importância fundamental dentro das atividades acadêmicas, por ser o tipo mais frequente, representa uma dimensão extremamente importante. Ela procura explicar e responder problemas dentro de uma dimensão teórica.

Pesquisa Explicativa busca identificar os fatores que determinam ou que contribuem para a ocorrência dos fenômenos. É o tipo que mais aprofunda o conhecimento da realidade, porque explica a razão, o porquê das coisas. Por isso, é o tipo mais complexo e delicado.

Pesquisa de Campo procura o aprofundamento de uma realidade específica. É basicamente realizada por meio da observação direta das atividades do grupo

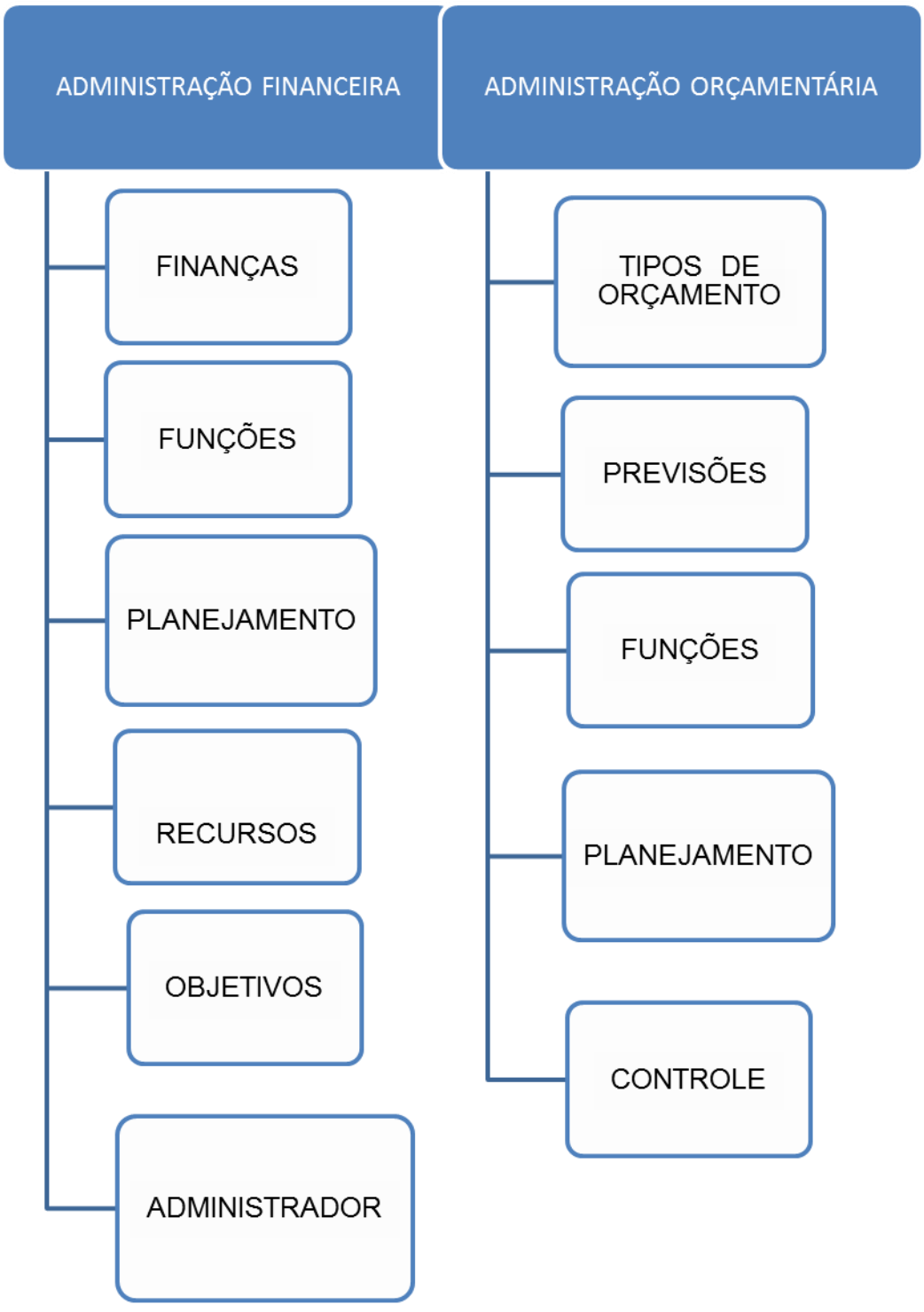
estudado e de entrevistas com informantes para captar as explicações e interpretações do ocorrem naquela

2 FUNDAMENTAÇÃO TEÓRICA

Na figura 1 será apresentado o mapeamento do conteúdo

FIGURA 1 – MAPEAMENTO DO CONTEÚDO

Na figura 1 será tratado dos principais tópicos para esclarecer o que é e como funciona a Administração financeira e orçamentária, e mostrar sua importância e a necessidade de ter informações confiáveis e administradores capazes de tomar decisões com menor risco possível para a organização afim de maximizar o lucro e minimizar as perdas.



FONTE: AUTORAS (2017)

Finanças:

É uma série de princípios econômicos para maximizar a riqueza ela se dedica a avaliar como são obtidos e geridos ou seja finanças tratam da gestão do dinheiro. Um assunto que é pertinente a todos e não só a empresas ou instituições financeiras.

Funções:

Dependem do tamanho da empresa, em pequenas empresas a função financeira é exercida pelo departamento de contabilidade, em medida que a empresa cresce a importância da função financeira conduz em geral a criação de um departamento próprio.

Planejamento:

O planejamento financeiro é um processo continuo que pode ser avaliado na medida em que a organização se desenvolve, sendo que a cada crescimento organizacional, se identifiquem novas necessidades e oportunidades de desenvolvimento, tanto na forma de agir como na forma de enfrentar os obstáculos financeiros existentes.

Recursos:

Os recursos financeiros compreendem os recursos na forma de capital, fluxo de caixa, investimentos, aplicações, empréstimos, financiamento etc.

Objetivos:

Dividem-se em primários e secundários, os primários são o lucro da empresa e riqueza de seus proprietários, os secundários são os meios que levam a alcançar os objetivos primários.

Administrador:

O administrador financeiro, diante da complexidade do mundo empresarial, precisa de uma visão holística da empresa e de seu relacionamento com o ambiente externo. Pois, o conhecimento de técnicas e métricas financeiras isoladas se mostra insuficiente, sendo necessária uma abertura para valores e informações estratégicas.

ADMINISTRAÇÃO ORÇAMENTÁRIA

Tipos de Orçamento:

Orçamento Estático, orçamento flexível, orçamento Rolling ou contínuo, orçamento Beyond Budgeting, orçamento ajustado, orçamento base zero (OBZ), controle matricial e peças orçamentárias.

Previsões:

Ela é tão necessária na empresa como ter uma planilha de fluxo de caixa, ou seja, é uma ferramenta que todo gestor necessita ter em mãos.

Planejamento:

Planejamento financeiro nas organizações é importante porque trata-se de uma ferramenta necessária ao crescimento, fortalecimento e existência. Ao estipular os objetivos da empresa, o gestor traça metas que deverão ser seguidas para que não faltem recursos para a realização das operações. O planejamento define as linhas de investimento e financiamento da empresa.

Controle:

Nem sempre é possível seguir o rumo traçado, as condições mudam e a empresa tem que se ajustar a elas se não puder controlá-las. De qualquer modo, o administrador necessitará acompanhar os desvios, analisá-los e tentar corrigi-los, e para isso necessita de um controle.

2.1 FINANÇAS

> Podemos dizer que finanças é a arte e a ciência de gerenciar recursos. Portanto, os mercados financeiros, as instituições financeiras e toda a estrutura de funcionamento desses sistemas, em nosso pais, bem como, no mercado internacional formam um campo de estudos muito importante conhecido como finanças. (CHENÇO, 2009, p. 11)

> O principal objetivo da Administração Financeira para as empresas é o aumento de seu lucro/rentabilidade para com seus proprietários. Todas as atividades empresariais envolvem recursos financeiros e orientam-se para a obtenção de lucros (Braga, 1995).

> Os proprietários investem em suas entidades e doravante pretendem ter um retorno compatível com o risco assumido, através de geração de resultados econômico-financeiros (lucro/caixa) adequados por um tempo longo, ou seja, durante a perpetuidade da organização.
> Uma geração adequada de lucro e caixa faz com que a empresa contribua de forma ativa e moderna em funções sociais, ou seja, pagamentos de salários e encargos, capacitação dos funcionários, investimentos em novas Tecnologias de Informação (TI), etc. (JUNIOR, 2010, p. 03)

> A administração financeira ou finanças, como quase toda ciência, traz em seu escopo as mudanças do mundo contemporâneo. Suas técnicas, métodos quantitativos e estrutura conceitual vêm sendo ampliada, o que aumenta sua relevância para as organizações. Por outro lado, o administrador financeiro passou a ser mais exigido, o que ocasionou a necessidade de especialização e atualização perene. (SILVA, 2013) ARTIGO

Talvez a razão mais importante para conhecer finanças é a de que você terá que tomar decisões financeiras que serão muito importantes em termos pessoais. (ROSS, p. 38).

> O conhecimento pode auxiliar as pessoas, tanto em suas atividades profissionais como pessoais, a atingir seus objetivos ou objetivos que lhe são impostos. Para alcança-los, é necessário que o indivíduo destine, da melhor maneira possível, os seus recursos disponíveis, sejam eles financeiros, materiais ou conhecimento, pois, dessa forma, sua tomada de decisão será embasada em argumentos reais e conscientes, que permitirão que as transações financeiras tenham o resultado maximizado. (SELEME, 2009, p. 13)

> Finanças é a ciência de gerenciar fundos de investimento; virtualmente todos os indivíduos ou organizações ganham com a captação de recursos oriundos de dinheiro ou outro bem da mesma espécie. (MARQUES, 2013, p. 5)

> O crescimento econômico depende da escolha e do financiamento de bons projetos de investimento. O estudo de Finanças é relevante para se entender como os empreendimentos são avaliados e financiados, seus recursos financeiros e riscos são geridos e os mercados e instituições financeiras atuam ao intermediar as transações que os viabilizam. Brealey, Myers e Allen (2006, p. 5) afirmam que Finanças também considera o comportamento das pessoas, além de dinheiro
> e mercados. (ARTIGO PRODUÇÃO Editores Científicos: Carlos Osmar Bertero, Flavio Carvalho de
> Vasconcelos, Marcelo Pereira Binder, Thomaz Wood JO CIENTÍFICA BRASILEIRA EM FINANÇAS NO PERÍODO 2000-2010)

A administração financeira é uma ferramenta muito importante para as empresas, com a sua ajuda as companhias podem ter condições de contabilizar seus resultados, ficando a par de onde estão sendo aplicados seus recursos e se está obtendo o retorno esperado. A administração financeira se utiliza de dados contábeis para formular suas decisões, para que a empresa tenha um desempenho elevado, trazendo recursos ao caixa para a realização de novos investimentos e a valorização. (BISPO, 2015 ARTIGO)

BIBLIOGRAFIAS

Gitiman, Lawrence J. Princípios da administração financeira

http://www.administradores.com.br/artigos/economia-e-financas/objetivos-da-administracao-financeira/69169/

XIII ERIC – (ISSN 2526-4230)

CONCESSÃO DE CRÉDITO A PESSOA JURÍDICA E SUA RELAÇÃO COM A COBRAÇA E O CONTAS A RECEBER ENQUANTO FERRAMENTAS DA ADMINISTRAÇÃO

Veridiana dos Santos França Barros
Kátia Tóffolo Simino (Orientadora)

1 INTRODUÇÃO

2.2 IMPORTÂNCIA E RELEVÂNCIA DO TEMA

> As contas a receber são geradas pelas vendas a prazo, que são feitas após a concessão de crédito. As vendas a prazo geram riscos de inadimplência e despesas com análise de crédito, cobrança e recebimento, mas alavancam as vendas, isto é, aumentam o volume de vendas e, consequentemente o lucro. As vendas a prazo são condições necessárias para aumentar o nível de operações e o giro dos estoques e, assim, ganhar a escala e maximizar a rentabilidade. As compras e vendas a prazo são operações comerciais e normais entre as empresas, e são chamadas crédito mercantis. Mesmo as empresas de prestação de serviços, primeiro prestam serviços e depois recebem. A concessão de crédito para consumidores finais pessoas físicas é chamada crédito ao consumidor. (HOJI, 2002, p 129)

O fato de uma empresa conceder crédito ao cliente significa que a empresa espera receber este valor em seu determinado prazo, e o mesmo acontece quando a empresa recebe algum crédito de instituições bancárias por exemplo, a concessão do crédito gera uma relação de credibilidade entre as partes, porem essa relação de credibilidade não exclui uma análise minuciosa do cadastro e das referências comerciais.

Assaf Neto e Silva (1997, p.97) afirmam que “crédito diz respeito à troca de bens presentes por bens futuros” e para Silva (200, p.63), “num sentido restrito e específico, Crédito consiste na entrega de um valor presente mediante uma promessa de pagamento”.

A análise de crédito não se apresenta apenas nas primeiras compras, mas mesmo aqueles clientes tradicionais e mais antigos necessitam de uma analise constante para manter as informações sobre o cliente atualizadas, como a pontualidade, capacidade de pagamento e a situação financeira no momento, para isso são usados meios como Serviço de Proteção ao Crédito (SPC) e Centralização de Serviços Bancários S.A (SERASA).

> A correta e legitima concessão de crédito é imprescindível para a sobrevivência da organização, uma vez que o não recebimento é considerado perda total de um ativo certo e detrimento de ganho em outras operações (LEONI,1998, p.123)

Por definição, o processo de análise subjetivo envolve decisões individuais quanto à concessão ou recusa de crédito. Neste processo, a decisão baseia-se na experiência adquirida, disponibilidade de informações e sensibilidade de cada analista quanto ao risco do negócio Gitman (2001) acredita que um dos insumos básicos à decisão final de crédito é o julgamento subjetivo que o analista financeiro faz para determinar se é válido ou não assumir riscos. Segundo o autor, a experiência adquirida do analista e a disponibilidade de informações (internas e externas) sobre o caráter do cliente são requisitos fundamentais para análise subjetiva do risco de crédito. As informações que são necessárias para análise subjetiva da capacidade financeira dos clientes são tradicionalmente conhecidas como C´s do Crédito: Caráter, capacidade, capital, colateral e condições, conforme evidenciado por Gitman (1997):

> **1** Caráter: O histórico do solicitante quando ao cumprimento de suas obrigações financeiras, contratuais e morais.
> **2** Capacidade: O potencial do cliente para quitar o crédito solicitado.
>
> **3** Capital: A solidez financeira do solicitante, conforme indicada pelo patrimônio liquido da empresa.
> **4** Colateral: O montante de ativos colocados à disposição pelo solicitante para garantir o crédito.
> **5** Condições: As condições econômicas e empresariais vigentes, bem como circunstâncias particulares que possam afetar qualquer das partes envolvidas. (GITMAN, 1999, p.697).

1.3 PROBLEMÁTICA

Quanto à concessão, análise e liberação de crédito à pessoa jurídica qual a importância de manter uma relação com a cobrança e o contas a receber?

1.4 OBJETIVO

1.4.1 OBJETIVO GERAL

1.4.2 Objetivos Específicos

a) Analisar todo o processo de análise e concessão de crédito a clientes.

b) Demonstrar os diferentes métodos e parâmetros utilizados na liberação de crédito a clientes.

c) Identificar em quais circunstâncias os métodos adotados pela empresa poderão ser alterados ou sofrer exceções.

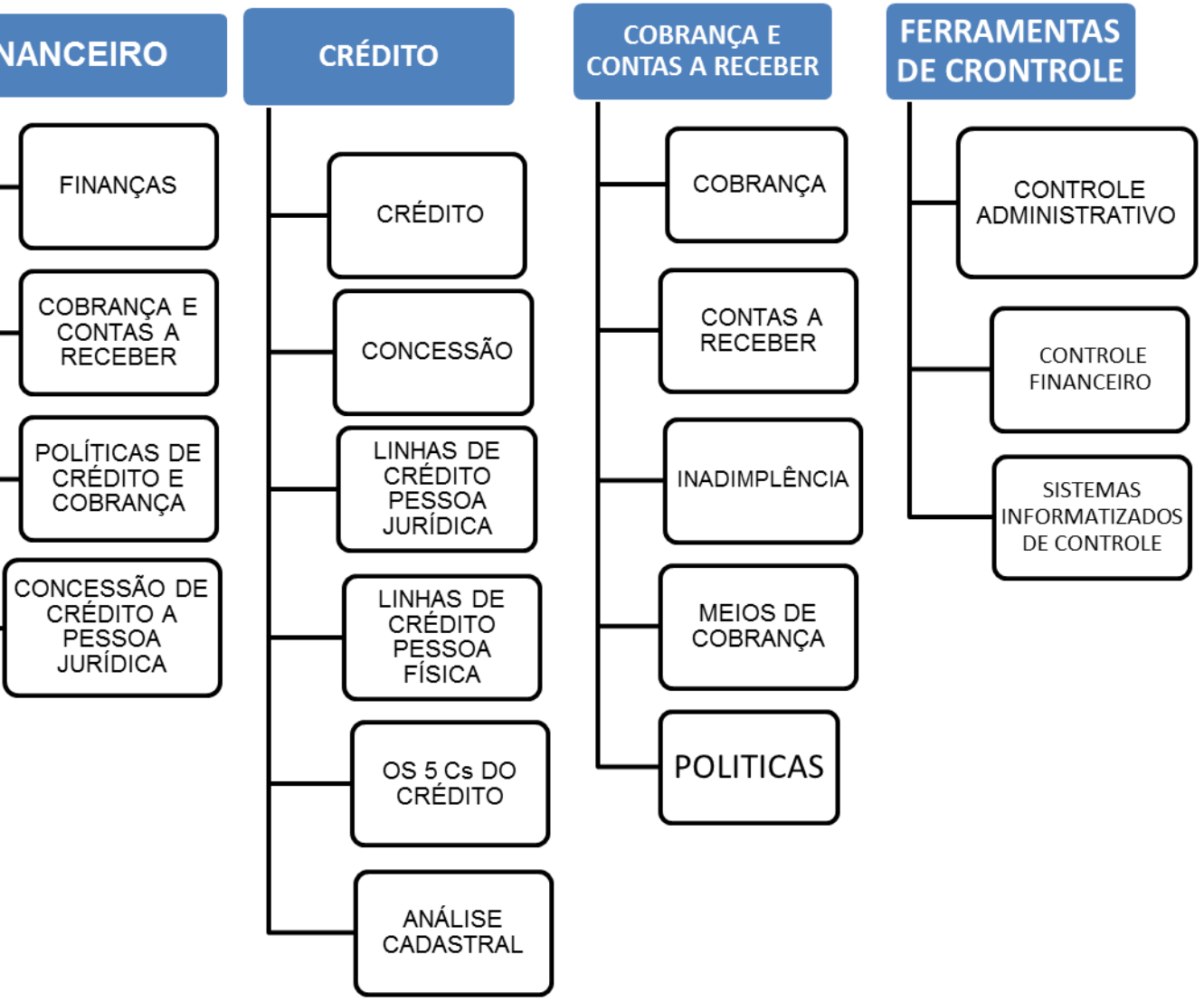
1.5 METODOLOGIA
Bibliográfica, descritiva, explicativa, exploratória...

A pesquisa exploratória, designada por alguns autores como pesquisa quase científica ou não científica, é normalmente o passo inicial no processo de pesquisa pela experiência e um auxílio que traz a formulação de hipóteses significativas para posteriores pesquisas.
A pesquisa descritiva busca descrever as características de determinada populações ou fenômenos. Uma de suas peculiaridades está na utilização de técnicas padronizadas de coleta de dados, tais como o questionário e a observação sistemática. Ex.: pesquisa referente à idade, sexo, procedência, eleição etc.
A pesquisa bibliográfica apresenta uma importância fundamental dentro das atividades acadêmicas, por ser o tipo mais frequente, representa uma dimensão extremamente importante. Ela procura explicar e responder problemas dentro de uma dimensão teórica.
Pesquisa Explicativa busca identificar os fatores que determinam ou que contribuem para a ocorrência dos fenômenos. É o tipo que mais aprofunda o conhecimento da realidade, porque explica a razão, o porquê das coisas. Por isso, é o tipo mais complexo e delicado.

3 FUNDAMENTAÇÃO TEÓRICA

Na figura 1 será apresentado o mapeamento do conteúdo
FIGURA 1 – MAPEAMENTO DO CONTEÚDO

Na figura 1 será apresentada a importância dos tópicos para que a administração financeira e orçamentária seja mais precisa e garanta a maximização do lucro e a minimização dos gastos.



FONTE: AUTORAS (2017)

FINANÇAS:

Finanças pode ser definida como "a arte e a ciência de administrar fundos. Praticamente, todos os indivíduos e organizações obtêm receitas ou levantam fundos, gastam ou investem. Finanças ocupa-se do processo, instituições, mercados e instrumentos envolvidos na transferência de fundos entre pessoas, empresas e governo”

COBRANÇA/CONTAS A RECEBER:

A administração das contas a receber, representa uma concessão de crédito aos clientes, gerando o volume de vendas à vista e a prazo.

POLÍTICAS DE CRÉDITO:

É a política de crédito que disciplina o conceito de crédito da organização normatiza os padrões a serem seguidos para a concessão, controle e acompanhamento do crédito determina o público alvo apresenta as modalidades de credito ofertados; bem como atribui a condições e critérios para a concessão e cobrança do crédito bem como atribui a responsabilidade para cumprir as normas estabelecidas por esta política e as penalidades ao descumprimento delas.

CONCESSÃO DE CRÉDITO A PESSOA JURÍDICA:

O crédito tem se tornado uma das principais atividades financeiras das empresas, por isso mesmo, a necessidade de que o credor analise cuidadosamente a capacidade de pagamento de cada cliente antes da concessão do crédito. Desta forma, o estabelecimento de procedimentos, visa tornar a concessão de crédito menos arriscada.

CRÉDITO:

O Crédito é parte da estrutura financeira das organizações. Ao conceder crédito, a empresa disponibiliza parte de seus recursos para financiar seus clientes.

CONCESSÃO:

Concessão de crédito é o fornecimento de crédito para um indivíduo. Este capital possibilita acesso a bens e serviços que, de outra forma, não seriam adquiridos ou demorariam a ser.

LINHAS DE CRÉDITO PARA PESSOA JURÍDICA:

As linhas de crédito, como o próprio nome diz, são limites de crédito/ empréstimos que são concedidos às pessoas físicas e jurídicas que tem como objetivo o crescimento e, para isso, precisam de investimento. São oferecidas por instituições bancárias e/ ou financeiras. O limite para as linhas de crédito são estipulados pela análise de renda ou pelo faturamento da empresa. Existem muitas linhas de crédito disponíveis.

OS 5 C’s DO CRÉDITO:

Os 5 C’s do crédito ou da cobrança vem sendo muito utilizada na gestão das empresas, talvez não na sua totalidade, mas em partes contribuiu para o bom andamento das políticas de crédito e cobrança.

O mais comum é o caráter, que entende o cenário financeiro da empresa, como está classificada entre os seus credores, se existem pendências financeiras com outras empresas.
Já com o termo capacidade, a avaliação se concentra na análise do fluxo de caixa e endividamento da empresa que solicita a compra. O estudo realizado é se a empresa dispõe de fluxo de caixa suficiente para saldar as dívidas assumidas e ainda assim garantir caixa suficiente para saldar com os novos compromissos adquiridos.
O Capital é quase um subitem da capacidade, já que tem como definição pura e simples o total de dinheiro ou patrimônio investido no negócio.
A definição de colateral está baseada basicamente nas garantias de retorno, caso existam problemas com a quitação dos créditos adquiridos.
Por fim, o fato condições, determina qual é a situação real da empresa como um todo. O mercado que está inserida, a avaliação da empresa dentro deste mercado, se está em queda ou apresenta resultado de crescimento e expansão.

ANÁLISE CADASTRAL:
A análise cadastral consiste no levantamento e análise de informações relacionadas à idoneidade do cliente com credor e mercado de crédito.

COBRANÇA:
A atividade da cobrança é importantíssima, possui a capacidade de antecipação dos valores em atraso e a responsabilidade para com os resultados positivos junto aos clientes. A cobrança é uma atividade dependente da qualidade dos controles, desde o acionamento do devedor até o devido pagamento de todos os débitos, que assegura a transformação das vendas em recebimentos.

CONTAS A RECEBER:
Contas a receber é o nome da conta onde se registram os aumentos (entradas) e as reduções (baixas) relacionados com a venda de conceitos diferentes de produtos ou serviços. Esta conta é composta por letras de câmbio, títulos de crédito e promissórias a favor da empresa.

INADIMPLÊNCIA:
O não pagamento até a data de vencimento de um compromisso financeiro, quando feita negociação de prazos entre as partes, para aquisição de bem durável ou não-durável, ou prestação de serviços, devidamente executados.

MEIOS DE COBRANÇA:
Para evitar que a inadimplência prejudique os resultados e a saúde financeira da empresa, o ideal é começar a cobrar os clientes assim que o atraso é identificado. O que pode ser feito por meio de, telefone, mensagem, email, correspondência, cobrança recorrente, lembrete da data de vencimento, incentivos para pagamento em dia, renegociação da dívida entre outros.

POLÍTICAS:
Toda empresa deve estabelecer regras claras e objetivas para efetuar a cobrança dos clientes que atrasam ou não efetuam o pagamento de uma prestação. Se, de um lado, a política de concessão de crédito busca minimizar o risco de que um determinado cliente se torne inadimplente, por outro a política de cobrança procura aumentar as chances de o cliente efetuar o pagamento de pelo menos parte da quantia que deve à empresa.

CONTROLE ADMINISTRATIVO:
Definido como qualquer processo que direciona as atividades das pessoas para alcançar as metas organizacionais. Aplica-se a toda organização, sendo que todos os aspectos do desempenho de uma empresa devem ser monitorados e avaliados, considerando objetivos e critérios diferentes em cada um dos níveis hierárquicos de controle.

CONTROLE FINANCEIRO:
Controle financeiro baseia-se na coordenação das atividades e avaliação da condição financeira da empresa. Por meio de relatórios financeiros elaborados a partir dos dados patrimoniais e da situação do fluxo de caixa.

SISTEMAS INFORMATIZADOS DE CONTROLE:
Sistemas de informação como ferramenta gerencial para tomada de decisão do gestor. O processo de implantação de sistema de informação na empresa necessita de envolvimento e integração de diversos recursos, tais como: recursos financeiros, humanos, materiais e tecnológicos. Além disto, é necessário mudança de pensamento do gestor para que este processo se concretize com sucesso.

3.1 FINANCEIRO

REFERÊNCIAS

(GITMAN, Lawrence J. Princípios de administração financeira. São Paulo: 1999, p.697).
LEONI, Geraldo; LEONI, Evandro G. Cadastro, crédito e cobrança. 2. ed. São Paulo: Atlas.1998, p 1236).
ASSAF NETO, A.; SILVA, C. A. T. Administração do Capital de Giro. 2 ed. São Paulo: Altas, 1997, p.97).

SILVA, José Pereira da. Gestão e Análise de Risco de Crédito. São Paulo: Atlas, 2000. p. 63).
MASAKAZU, Hoji, Administração Financeira - uma Abordagem Prática, 2002. p. 129).

BIBLIOGRAFIAS

FERRARI, Ed Luiz. Contabilidade geral.9 ed. Rio de Janeiro2009

GITMAN, Lawrence. Princípios de administração financeira. São Paulo: 2010.

MAXIMIANO, Antonio Cesar Amaru. Introdução à administração.5 ed. Atlas 2000.

ROSS, Westerfield. Jordan. Princípios de administração financeira. 2. ed. Atlas. 2008.

BEZERRA, Felipe. Administração financeira e orçamentária (AFO), 2015 Disponível em:
<<http://www.portal administracao.com/2015/01/administracao-financeira-e-orcamentaria.html. Acessado dia 20/04/2017.
Departamento de Ciências Contábeis, Universidade Federal de Santa Maria (UFSM), Santa Maria, RS, Brasil. Sistema & Gestão, revista eletrônica. 2012. Política de cobrança de contas a receber: um estudo de caso no comércio varejista de materiais de construção. Disponível em: <<http://www.revistasg.uff.br/index.php/sg/article/viewFile/V7N3A8/V7N3A8. Acessado em 20/04/2017

Faculdade de educação de Costa Rica, disponível em <<https//www.fecra.edu.br/admin/arquivos/Artigo_ADMINISTRACAO_FINANCEIRA.pdf. Acessado dia 20/04/2017.

LEONI, Evandro Geraldo. Cadastro, crédito e cobrança, 3 ed. São Paulo. Atlas 2011. Siqueira, Denis artigo. Disponível em <<https://www.créditoecobranca/artigo10.asp. Acessando em 19/04/2017.

MEDZO CONSULTORIA FINACEIRA. AGUSTINELI, Juliana. 2014.
Disponível em:
<<https://medzonconsultoriafinanceira.wordpress.com/2014/10/28/principais-diferenca-entre-planejamento-orcamento-e-planejamento-financeiro. acessado em 19/04/2017.

SILVA, Bráulio Wilker, 2013. Objetivos da administração financeira. Disponível em: <<http://www.administradores.com.br/artigos/economia-e-financas/objetivos-da-administracao-financeira/69169/. Acessado em 19/04/2017.