

Pós-graduação

Inatel

**Tópicos avançados em *cloud computing, mobile* e
tendências de mercado.**

2022.

última atualização: 02/05/2022

Disciplina DM 119
Data: junho de 2022
Prof. Rodrigo Pimenta Carvalho

Sumário

Introdução.....	3
Capítulo I – AWS IoT.....	8
O GATEWAY DE DISPOSITIVOS.....	9
O MOTOR DE REGRAS.....	10
AS AÇÕES DAS REGRAS.....	11
AS SOMBRAS DAS COISAS.....	12
CONSTRUÇÃO DE SOLUÇÕES.....	12
CONSTRUINDO UM CASO DE USO COM AWS IoT.....	14
Capítulo II – APIs : Application Programming Interfaces.....	41
O PODER DAS APIs.....	42
ALGUMAS APIs LEGAIS.....	47
FORMAS DE GANHAR DINHEIRO COM APIs.....	56
APIs E IOT.....	57
CONCLUSÕES E CONSIDERAÇÕES FINAIS SOBRE APIs.....	57
Capítulo III - SIP.....	62
VISÃO GERAL.....	62
MENSAGENS DO PROTOCOLO SIP.....	66
SIP EM TELECOMUNICAÇÕES.....	85
DETALHES COM PEERS EM REDES REMOTAS.....	86
PILHAS SIP.....	98
Capítulo IV - OpenSIPS.....	100
RESUMO SOBRE O OPENSIPS.....	101
PRINCIPAIS PARTES NO ARQUIVO DE CONFIGURAÇÃO DO OPENSIPS.....	105
EXEMPLOS DE REQUISITOS DE PROJETOS QUE PODEM SER IMPLEMENTADOS COM A AJUDA DO OPENSIPS.....	108
COMO INSTALAR E CONFIGURAR O OPENSIPS PARA INICIAR O USO DELE.....	118
COMANDOS OPENSIPSCTL.....	124
TABELAS DO BANCO DE DADOS DO OPENSIPS.....	124
BÔNUS: CERTIFICADOS E TLS USADOS NO OPENSIPS.....	127
COMO GERAR OS CERTIFICADOS (NO LINUX).....	127
TESTES COM TLS E SOFTPHONE ZOIPER.....	129
Capítulo V – Micros-serviços.....	131
INTRODUÇÃO.....	131
A ARQUITETURA MONOLÍTICA.....	132
A ARQUITETURA BASEADA EM MICROS-SERVIÇOS, CARACTERÍSTICAS E VANTAGENS EM RELAÇÃO À ARQUITETURA MONOLÍTICA.....	135
PADRÃO DE LINGUAGEM PARA A ARQUITETURA DE MICROS-SERVIÇOS.....	147
CONCLUSÕES.....	149
Capítulo VI - Realidade Aumentada.....	152
O QUE É A REALIDADE AUMENTADA?.....	152
EXEMPLOS INICIAIS.....	154
CONSTRUÇÃO DE APPLICATIVO COM RA E MARCADORES FÍSICOS, EM LINHAS GERAIS.....	154
VUFORIA.....	157
WIKITUDE.....	174
Anexo I – Script opensips.cfg.....	179
Anexo II – Instalação de software necessário ao trabalho prático sobre AWS-IOT.....	186
Bibliografia.....	188

Introdução

Há um tempo algumas disciplinas foram vistas durante a pós-graduação, relacionadas a várias tecnologias. Tais disciplinas vêm sendo planejadas e elaboradas ao longo de alguns anos. E como tudo em tecnologia evolui cada vez mais rápido, em paralelo outros novos assuntos foram surgindo, ganhando atenção do público profissional da área de TI e se tornando alvos de muito interesse a quem deseja trabalhar na área de arquitetura de software, APIs, computação em nuvem, Telecomunicações, bem como desenvolvimento de software para dispositivos móveis. Por exemplo, o surgimento de *Internet of Things* (IoT) desperta muito interesse atualmente. Mas, esse assunto remete a outros que apresentam forte relação com computação em nuvem, *microservices*, APIs, protocolos relacionados com telefonia, etc. Tendo em vista o objetivo de passar aos estudantes algumas das tendências atuais em tecnologias também relacionadas com esta pós-graduação, bem como alguns tópicos avançados, os assuntos que foram se tornando relevantes nos últimos 2 anos foram motivo de estudos e explanações no texto desse documento. Os capítulos nessa apostila introduzem assuntos que podem ser pesquisados e detalhados pelo próprio aluno, sendo possível criar apresentações dos mesmos, com trabalhos feitos em equipe. Isso sim possibilitará a divulgação rápida das tecnologias interessantes, durante o andamento da disciplina DM119.

São muitos os assuntos interessantes em tecnologia, relacionados com computação em nuvem e aplicações para dispositivos móveis que poderiam ser demonstrados nessa disciplina, mas seria difícil ir a fundo em todos eles e trazê-los à classe de aula sem a participação dos próprios alunos. Portanto, no intuito de abordar os assuntos de interesse e compartilhar conhecimentos a um nível adequado em termos de detalhamento, faz-se necessário seguir uma estratégia contando com a colaboração de todos os interessados. Assim, alguns assuntos foram escolhidos e introduzidos nos próximos capítulos. A introdução de cada assunto compõem material suficiente para que seja continuada a pesquisa sobre os mesmos de forma mais aprofundada. O resultado da pesquisa de cada assunto culminará na apresentação dos mesmos, de forma oral e com utilização de slides. Estas apresentações garantirão o compartilhamento das informações aos interessados, de forma satisfatória, sendo que todos os envolvidos terão um ótimo retorno participando desta disciplina.

E é claro que IoT não ficaria de fora dessa disciplina! São muitos os tópicos envolvendo IoT e que cabem bem nesta pós-graduação. Por exemplo, a computação em nuvem veio ajudar e tornar bem possível os projetos em IoT. O Amazon Web Services (AWS) criou toda uma estrutura para computação em nuvem de dados relacionados de alguma forma com as coisas ligadas à Internet. Tal estrutura já está disponível no Brasil, na região de disponibilidade São Paulo (sa-east-1). O nome da mesma é AWS IoT. Já que está disponível no Brasil (há *datacenter* aqui, em 2020, com máquinas para AWS IoT), os recursos usados dela são cobrados em moeda local. Se o recurso de envio de SMS é utilizado a partir do AWS IoT, cobra-se o valor do envio da mensagem do país onde está a estrutura do AWS IoT até o país de destino da mensagem, que pode ser o Brasil no nosso caso. Por outro lado, se um projeto utiliza o AWS IoT localizado na região de disponibilidade de N. Virgínia (EUA) para envio de mensagem a telefones no Brasil, serão cobradas taxas em dólar para envio de mensagens de lá para cá. Mas, os valores são baratos. O **Capítulo I** cuidará do assunto **AWS IoT**.

Esse será um assunto de Computação em Nuvem.

Em algumas disciplinas da pós-graduação, assuntos como SOA , serviços e interfaces de serviços são mostrados. As interfaces dos serviços, como é ensinado, devem ser bem identificadas, planejadas e seguirem contratos entre serviços e clientes dos mesmos. Esses contratos garantem o encapsulamento dos serviços deixando somente as interfaces visíveis a quem demanda os mesmos. Vale lembrar que uma boa prática é criar as interfaces mesmo antes da implementação dos serviços. Esta organização com interfaces, vista em SOA, provê um dos princípios de design de software que ajudam as empresas a manterem seus serviços de software por mais tempo em uso, com menos impacto entre os sistemas, caso algum deles precise ser modificado, mantendo a interface sem modificações, por exemplo. Os contratos de serviços – as interfaces - podem ser escritos em WSDL (para o caso de SOAP) ou em WADL (para o caso de REST), sendo que nos dois casos o que se tem é a lista de métodos que se quer expor dos serviços subjacentes às interfaces. Agora que a TI está utilizando interfaces de serviços como contratos de uso entre os sistemas de software, muitas empresas passaram a criar interfaces para expor publicamente as capacidades de seus sistemas computacionais, com o intuito de deixar profissionais de TI usarem tais sistemas, para obter computação executada no domínio destas empresas, mas também divulgar as mesmas e seus serviços. O conjunto das interfaces desenvolvidas, mais a documentação das mesmas, compõem a API de um serviço ou serviços. A API é então a forma de acessar e usar as capacidades de algum sistema computacional de uma empresa, instituição, escola, etc. E atualmente há cada vez mais o desenvolvimento de APIs no Brasil e no exterior. Há até mesmo uma empresa brasileira especializada nesse assunto. O mais interessante do uso de APIs é a possibilidade da construção de novos sistemas de software utilizando outros sistemas já prontos e públicos, os quais fornecem trabalhos de grande utilidade. O **Capítulo II** cuidará do assunto **APIs**. Esse será um assunto de tendência no mercado e em Computação.

Como essa pós-graduação trata de aplicações para dispositivos móveis e computação em nuvem, vem tratar também de software para *smartphones*. Isso significa que a telefonia está relacionada com os assuntos de algumas das disciplinas. Mais precisamente, a telefonia com VoIP destaca-se atualmente no uso de *smartphones* e aplicações com VoIP para esses dispositivos móveis têm grande relevância. Mas, para que uma aplicação de VoIP funcione, é necessário o uso de um protocolo que permite o estabelecimento de sessão entre os *peers* que pretendem conversar. Somente com uma sessão estabelecida fica possível para as partes enviarem *media* entre elas, quer seja vídeo, quer seja áudio. A sessão, por exemplo, permite que um *peer* saiba qual é a localização do outro, em termos de endereçamento IP. O protocolo mais utilizado no mundo para o estabelecimento de sessão, com troca de informações sobre endereços IP, é o SIP. Mesmo para uma finalidade entre *smartphones* que não seja conversação, mas mensagens de texto por exemplo, pode-se demandar o estabelecimento da sessão. O **Capítulo III** cuidará do assunto **SIP**. Com o protocolo SIP, pode-se criar aplicações de telefonia na rede IP, ou de envio de mensagens ou qualquer outra que necessite estabelecer sessão de comunicação entre os *peers*. Esse será um assunto de tópico avançado no mercado e em Telecomunicações.

As mensagens do protocolo SIP são em formato texto, podem ser lidas pelas pessoas e se assemelham às mensagens do protocolo HTTP. Todo o poder (conjunto de possibilidades) do

protocolo SIP está exatamente no significado de cada atributo encontrado nos cabeçalhos das suas mensagens. Por exemplo, um cabeçalho de mensagem pode ter o dado de contato de quem cria uma chamada. Ou seja, na mensagem SIP responsável em iniciar o estabelecimento de uma sessão, encontra-se no seu cabeçalho o contato do *peer* que requisita o início de uma ligação, ou seja, de quem cria uma chamada. O contato pode ser, por exemplo: pimenta@192.168.21.40, o que significa que o originador da chamada tem login (ID) = pimenta e no momento corrente pode ser alcançado no nodo com IP = 192.168.21.40. Contudo, pode ser necessário mudar algum dado do contato, por exemplo antes que a mensagem SIP que o carrega chegue no seu destinatário. Isto é, supondo que o originador da chamada (chamador) está atrás de um NAT, do ponto de vista do chamado, não faria sentido entregar o contato com o IP privado do chamador ao chamado. Nesse caso, faz-se necessário “corrigir” o IP antes que a mensagem seja entregue. Portanto, alguma entidade deve interceptar a mensagem e fazer a tal “correção”. Para esse tipo de tarefa, de alterar o texto de mensagens SIP em trânsito, podemos usar uma entidade chamada SIP Proxy. O SIP Proxy é um sistema computacional que pode receber uma mensagem SIP e alterá-la, para depois encaminhá-la a outro nodo da rede. Quando um SIP Proxy lida com uma mensagem SIP e depois a encaminha sem manter qualquer informação da mesma, tal proxy é dito ser *stateless*. Mas, se o proxy manter alguma informação da mensagem encaminhada, ele é dito ser *statefull*. Com o uso do protocolo SIP e um SIP Proxy, pode-se criar um videoporteiro, o qual aceitará conexões de *smartphones* rodando *softphones*, por exemplo. Esse é um exemplo a ser comentado nessa disciplina. De fato, em 2016-2018 o Inatel construiu um videoporteiro usando SIP e SIP Proxy para uma empresa de Santa Rita do Sapucaí/MG. Um SIP Proxy também permite que valores sejam lidos e obtidos dos cabeçalhos das mensagens SIP, de tal forma que decisões programadas possam ser executadas pelo próprio proxy. Por exemplo, pode-se aferir que uma chamada está direcionada a um destinatário de uma *black list* e então encerrar a mensagem, sem que ela seja entregue. Isso acarreta então na interrupção da mensagem e evita o estabelecimento da conversação com o *peer* que não tem o direito para tal. Atualmente, existe um SIP Proxy *open source* para o mercado de telecomunicações, chamado OpenSIPS. O OpenSIPS permite fazer várias operações sobre as mensagens SIP e também num outro protocolo que pode ser utilizado juntamente com o SIP, chamado SDP. Os detalhes sobre o **OpenSIPS** serão vistos no **Capítulo IV**. O OpenSIPS pode ser entendido como um servidor SIP que aceita registros de clientes os quais em seguida podem estabelecer sessões entre si. O SIP Proxy coordena as sessões, obedecendo o significado dos valores contidos nos cabeçalhos das mensagens SIP que passam por ele. Cada *peer* num diálogo tem um agente SIP em si, o qual é o responsável pela criação das mensagens desse protocolo. Esse será um tópico avançado relacionado às Telecomunicações com SIP.

Enquanto o protocolo SIP foi criado para possibilitar a criação de sessões de *media* entre *peers* mesmo que eles estejam em redes diferentes (em domínios de redes diferentes), há outras tecnologias atualmente mais direcionadas para a comunicação *peer-to-peer* em redes locais, ou seja, com todos os *peers* no mesmo domínio de rede. O protocolo SIP pode estabelecer uma sessão entre dois usuários da telefonia, mesmo havendo NATs entre eles, porque essa questão pode ser tratada via um SIP Proxy, por exemplo, que sabe identificar a presença de NATs e portanto sabe também como e quando alterar endereços IPs encontrados nas mensagens do protocolo. Por outro lado, uma tecnologia atual para comunicação *peer-to-peer*, chamada AllJoyn, não necessita de uma entidade

computacional intermediária. Apenas os *peers* precisam estar na mesma rede, todos conectados num mesmo roteador Wi-Fi por exemplo, e a comunicação entre eles é feita por uma entidade conceitual chamada *bus*. Essa tecnologia foi criada pela Qualcomm e já existem várias aplicações da mesma no mundo. Essa tecnologia não será introduzida nessa disciplina, portanto está fora do escopo das aulas. Entretanto, é um assunto interessante e muito válido para ser pesquisado por alunos e apresentado de forma oral, caso algum se interesse. Com essa tecnologia os *smarphoes* numa mesma rede podem trocar informações sem grandes dificuldades para os desenvolvedores da aplicação. Esse é um assunto de tendência em Computação. O pessoal do ICC que trabalha para a Qualcomm pode ajudar em pesquisas sobre AllJoyn.

O mercado em geral está usando muito o termo IoT e é claro que um pouco disso está sendo usado para promover vendas de serviços, tecnologias e produtos. Qualquer caixa preta conectada à Internet recebe o título de coisa da Internet e faz parte da Internet das coisas. Mas, na verdade, a visão de IoT abrange realmente coisas variadas que poderão comunicar entre si e com a Internet. Por exemplo, uma rede de sensores numa plantação agrícola, para detectar invasão de insetos, poderá ter seus sensores comunicando entre si e depois enviando dados para algum tipo de *gateway* que finalmente passará dados para a Internet. Outro exemplo pode ser o clássico de coisas conectadas entre si dentro de uma casa e a existência de algum *gateway* na mesma casa para comunicação com a Internet. De fato, na IoT existirão muitos dispositivos pequenos, com restrições de processamentos e bateria, que comunicarão entre si, mas um tipo de *gateway*, com mais poder de bateria poderá ser uma interface com a Internet. Considerando dispositivos pequenos, de pouco poder de bateria, faz-se necessário que haja protocolos de comunicação entre eles visando o bom uso da energia, para economizar bateria o máximo possível. Protocolos já foram desenvolvidos com essa questão em mente. Por exemplo, o IETF já tem a RFC do protocolo CoAP. E em 2015 surgiu uma nova proposta de protocolo ainda mais econômico que o CoAP, segundo seus idealizadores, com intuito de evitar *overhead* que existe no CoAP. Esse segundo é baseado na ideia do SIP e tem o nome de CoSIP. A tecnologia do CoSIP também está fora do escopo dessa apostila, mas pode ser tomada como um desafio a quem se interesse em pesquisá-la e explicá-la em sala de aula oralmente. Portanto, caso algum aluno deseje explicar sobre o CoSIP, isso será de grande valia também, e será um tópico avançado a ser explorado. CoSIP é um tópico avançado em Telecomunicações.

Agora, mudando de baixo para alto nível, uma das tecnologias interessantes da atualidade é a que produz a realidade aumentada. Essa tecnologia aumenta a realidade de tal forma que um espectador de um fato ou imagem tem a sensação que realmente existe algo a mais na realidade, algo que seria impossível existir em condições normais. Com o uso de um *smartphone* e um aplicativo de realidade aumentada, olhando através da tela do celular, é possível ver objetos que não existem, como se o celular estivesse revelando algo além de uma janela imediatamente aberta. Na realidade aumentada percebe-se que os objetos produzidos no campo visual da câmera do celular e desenhados na tela do mesmo têm sim uma certa relação com a realidade. Por exemplo, se um objeto é visto sobre uma mesa e se a mesa é movida em várias direções e ângulos, o objeto da realidade aumentada move-se igualmente, como pode ser visto. Isso dá a sensação que o objeto está ali sobre a mesa. E se alguém passar entre o celular e a mesa, o corpo da pessoa encobre a imagem da mesa e da realidade aumentada, trazendo uma sensação de fato logicamente correto. Ou seja,

objetos produzidos pela realidade aumentada são fiéis ao ambiente visualizado pela câmera do *smartphone*, por exemplo. Para melhor entendimento, imagine então que a mesa é queimada e vira cinzas. Nesse caso o objeto da realidade aumentada desaparece, porque ele estava ligado à imagem da mesa, não à localização geográfica da mesma. Na realidade aumentada explicada aqui os objetos desenhados são mapeados sobre imagens reais, não sobre coordenadas de localização. Então, o novo jogo Pokémon GO gera realmente a sensação de realidade aumentada? Há alguns anos a Qualcomm desenvolveu uma coleção de ferramentas para a produção da realidade aumentada, chamada Vuforia. No Brasil não vemos essa tecnologia sendo usada massivamente, mas muito provavelmente por ter sido mal compreendida, o que pode ter inibido a criatividade para a criação de soluções que podem suprir demandas reprimidas. O exemplo do Pokémon GO serve bem para mostrar o potencial da realidade aumentada, ou pelo menos mostra que existe sim demanda reprimida para soluções com tal tecnologia. Portanto, o **Capítulo VI** tratará do assunto **Vuforia**. Esse será um assunto de tendência do mercado em Computação.

Para encerrar a lista de assuntos interessantes de tecnologias modernas e que são tendências no mercado de TI, esse documento comentará também sobre os *microservices*. Atualmente SOA ainda é muito difundido e usado em várias empresas e o ganho econômico que se tem com isso já é fato consumado. *Microservices* são tipos de serviços que também seguem vantagens definidas pela SOA, como o uso de interfaces bem definidas, tecnologias de invocações de métodos remotos (como o REST), reuso de trabalho computacional, etc. Mas, *microservices* são geralmente definidos com banco de dados isolados. Ou seja, cada serviço tem sua própria base de dados, o que é uma das diferenças em relação à SOA. Além disso os *microservices* são sempre pequenos, ou seja, geralmente desempenham uma única função simples. Portanto, geralmente são compostos de um único método. O assunto de ***microservices*** será tratado no **Capítulo V**. Esse será um assunto de tópico avançado em Computação e tendência de mercado atualmente.

Como visto acima, oito assuntos podem ser pesquisados nessa disciplina a tal ponto que isso servirá de base para continuação de pesquisas sobre os mesmos, ou até mesmo de outros assuntos correlacionados. Por exemplo, ao estudar AWS IoT, estuda-se talvez a tecnologia Node.js, que por sua vez pode remeter ao Node Red. Nesse caso, Node Red seria uma tecnologia correlacionada à AWS IoT, mas que poderia ser pesquisada e apresentada na segunda parte dessa disciplina por um aluno, ou grupo de alunos. Após a realização da primeira parte dessa disciplina (aula de 8 horas), haverá tempo para pesquisar mais sobre tais assuntos e novos conhecimentos sobre os mesmos serão apresentados na segunda aula pelos próprios alunos. Cada assunto escolhido pelo grupo de alunos poder ser apresentado mostrando exatamente o que foi dito na primeira aula, mas deve também trazer informação complementar sobre os mesmos, para o enriquecimento da turma ainda mais. Os alunos serão avaliados através de suas apresentações orais na segunda aula e através de seus materiais escritos (slides). Exercícios em aula poderão ser aplicados também, para a pontuação dos alunos.

Capítulo I – AWS IoT

É verdade que o AWS tem vários serviços úteis à computação em nuvem, como mostrado nessa pós-graduação. Todos esses serviços estão disponíveis para acesso de requisições de trabalhos computacionais a partir de clientes presentes em qualquer ponto na Internet. Mas, não há uma noção de IoT aí, ou seja, não há um planejamento para facilitar a comunicação com coisas na Internet, a fim de contatá-las, para alterar seus estados, por exemplo a partir de software gerente. Por exemplo, imagine que uma bomba de água, para piscinas, está ligada a um microcontrolador que contém um sensor. Quando a água atingir um certo nível, o sensor deverá avisar o microcontrolador que por sua vez deve desligar a bomba. Então o microcontrolador pode enviar o status de piscina cheia para algum serviço no AWS, para o armazenamento ou distribuição dessa informação. Mas, suponha que algum serviço de gerência precise enviar uma mensagem ao microcontrolador para ele alterar o status da piscina e talvez deixar que a bomba coloque um pouco mais de água lá. Como enviar uma mensagem ao microcontrolador, sem usar um *gateway* onde depositar mensagens ao dispositivo e esperar até que ele as consuma? Ou seja, para gerenciar dispositivos esparramados pela Internet, no mínimo tem-se que programar um sistema em algum servidor, que pode conter, por exemplo, um *broker* de mensagens. Se nada disso for usado, pode-se pensar em criar sessões de *media* entre as coisas remotas, talvez usando SIP, para o envio de mensagens a elas. Contudo, a Amazon criou o Amazon Web Services para *Internet of Things*, o AWS IoT, lançado em outubro de 2015. Nesse conceito, o AWS tem a capacidade de manter em si representações “especulares” dos objetos reais na Internet. Essas “imagens” dos objetos são chamados de *shadows* dos objetos. Um *shadow* é uma coisa conceitual contida no AWS, que representa a coisa real, de tal forma que mandar um comando ao *shadow* reflete na mudança de estado da coisa real. Ou seja, toda a infraestrutura para se ter representações de coisas no AWS já existe e ela permite manter uma coleção de coisas gerenciáveis sem a necessidade de programar artifícios como *brokers* de filas, sessões de *media*, etc. Os *shadows* são uma das grandes vantagens de se usar o AWS IoT. Detalhes sobre como usar os *shadows* serão deixados para serem pesquisados pelos alunos.

Segundo [1], “O AWS IoT é uma plataforma de nuvem gerenciada que permite que dispositivos interajam facilmente e com segurança com aplicativos de nuvem e outros dispositivos. O AWS IoT pode comportar bilhões de dispositivos e trilhões de mensagens, e pode processar e rotear essas mensagens para *endpoints* da AWS e para outros dispositivos confiavelmente e com segurança. Com o AWS IoT, as suas aplicações podem acompanhar todos os seus dispositivos, como também comunicar-se com eles, o tempo inteiro, mesmo quando eles não estiverem conectados.” O que significa essa explicação? A plataforma é gerenciada porque ela tem funções, via console web, que permitem executar várias ações sobre os serviços envolvidos ou sobre as coisas na Internet. Os dispositivos podem interagir com aplicativos de nuvem, porque através de SDKs fornecidos pelo próprio AWS IoT os desenvolvedores podem criar funções que enviam mensagens à infraestrutura da AWS facilmente, a partir desses dispositivos. E uma solução criada para algumas coisas na Internet pode ser escalada para bilhões de cópias dessas coisas em funcionamento, porque a infraestrutura do AWS IoT pode escalar automaticamente várias vezes, aumentando toda a

capacidade de serviços na nuvem. Tal infraestrutura permite também o envio de mensagens a todas as coisas representadas no AWS IoT, mesmo que seja um número grande de mensagens. O roteamento das mensagens recebidas no AWS IoT, para serviços internos, ou o roteamento delas para outros dispositivos pode ser feito por meio de regras bem definidas constantes na conta de quem seja o administrador dos serviços. Existe um motor de regras para isso. A conexão com segurança é garantida porque o IoT AWS gera certificados públicos e privados a serem utilizados no momento de criação de tais conexões, com TLS, o que garante que as mensagens sejam transmitidas criptografadas. Não há qualquer mensagem entre o AWS IoT e dispositivos externos sem que as mesmas estejam criptografadas. Ou seja, é impossível enviar mensagens ao AWS IoT ou receber mensagens de lá sem usar criptografia com TLS.

Ao entrar no console de serviços no web site do AWS (Amazon Web Services), fica possível acessar o AWS IoT, que atualmente é a única nova opção do AWS para IoT. Desse ponto em diante, várias configurações, programações e ações podem ser executadas nesse ambiente, tudo visando sistemas para IoT.

O Gateway de dispositivos

Para usar o AWS IoT usa-se um *gateway* de dispositivos, contido no domínio da AWS. Não havia um *gateway* desse no Brasil (até 2018) [a região de disponibilidade São Paulo não continha AWS IoT], mas era possível usar o que fica em N. Virgínia (EUA), por exemplo

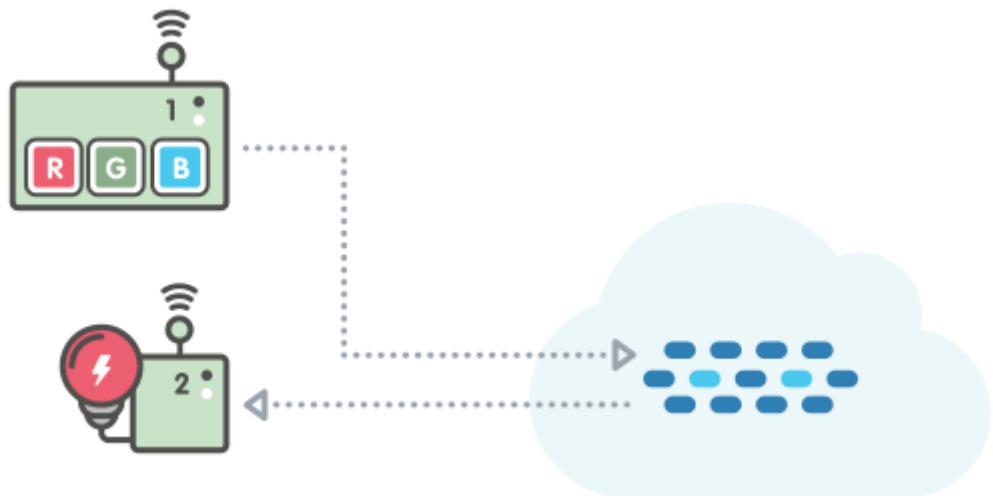


Figura I.

O *gateway* de dispositivos permite que os dispositivos se comuniquem de forma segura e eficiente com o AWS Internet das coisas. As coisas podem se comunicar mesmo se estiverem usando protocolos diferentes. O exemplo da figura I ilustra duas coisas - uma lâmpada ligada, e uma unidade de controle – todos conectados ao *gateway*. A unidade de controle pode publicar comandos para o *gateway*, e a lâmpada pode se inscrever no *gateway* para ouvir (receber) comandos relevantes. Em relação aos protocolos de comunicação entre os dispositivos e o *gateway*, mais detalhes podem ser pesquisados na documentação do [AWS IoT Developer Guide](#) vista aqui [2].

Então, mais pesquisas podem ser feitas, em relação aos protocolos e aos SDKs que a AWS liberar para os desenvolvedores trabalharem nessas comunicações entre os dispositivos e o *gateway*. O resultado dessa pesquisa pode ser demonstrado na apresentação oral a ser feita sobre os assuntos do AWS IoT.

O Motor de Regras

Cada mensagem que chega no AWS IoT pode ser avaliada por um sistema chamado Motor de Regras. Tal motor pode conter regras de negócio e decidir como invocar serviços do AWS para, por exemplo, concluir ações necessárias às regras definidas. Por exemplo, se um processo de negócios exige que mensagens SMS sejam enviadas após algum dispositivo mudar de status, então a regra respectiva (Ex: envie SMS para todos os celulares dos gerentes, quando a piscina do clube já estiver cheia) pode ser implementada no Motor de Regras. Quando o status mudar para piscina cheia, esse motor executará uma regra que implicará no uso do serviço de *push notifications* do AWS, que irá então enviar SMS. Aqui está perceptível o uso de serviços, regras de negócios e então a possibilidade de mapear processos de negócios em serviços invocáveis, o que se torna muito útil em BPM. A figura II ilustra uma regra que faz o seguinte:

- Avalia a mensagem recebida e verifica se valor é “B”.
- Se o valor é “B”, transforma para “G”, antes de enviar à lâmpada. Isso faz uma modificação no estado da lâmpada, um pouco diferente do que é requisitado por “B”.

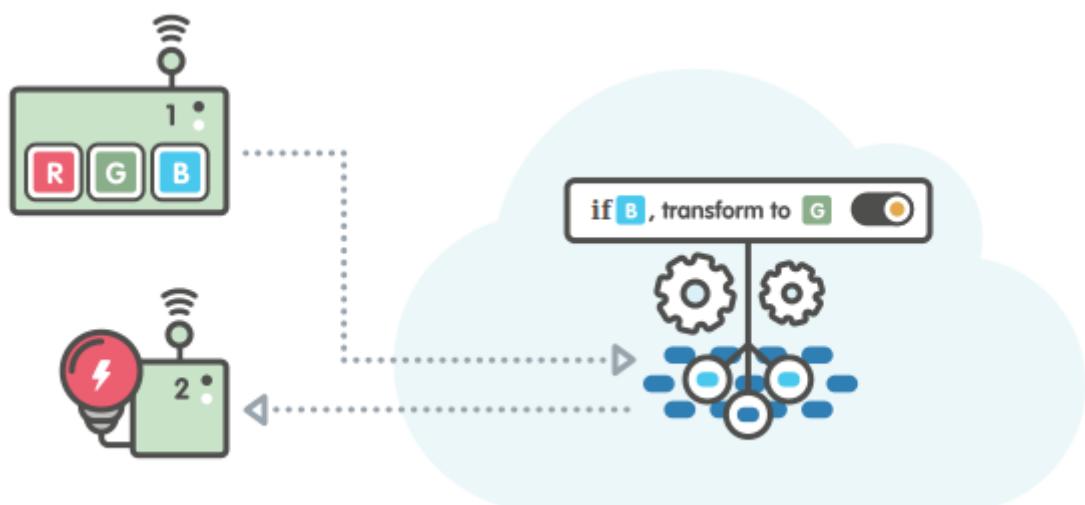


Figura II

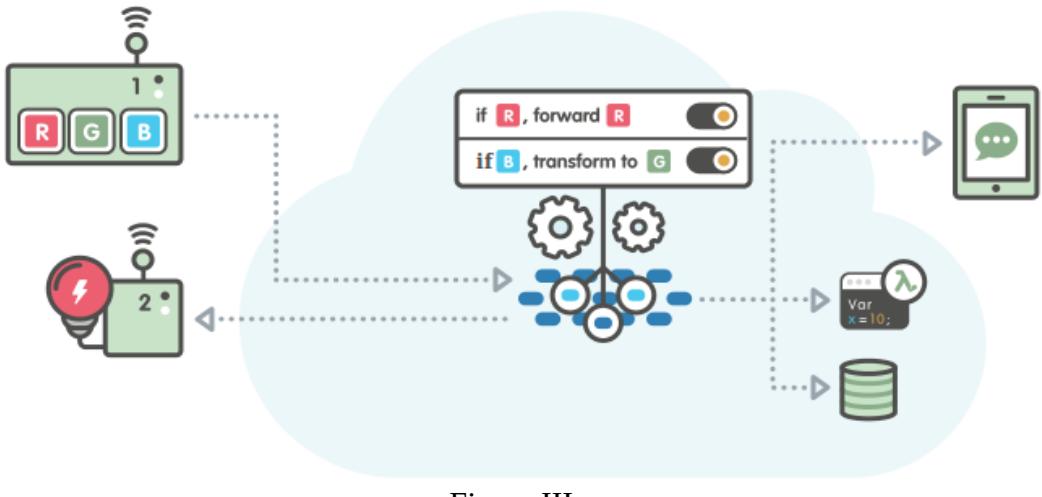


Figura III.

As ações das regras

O mecanismo de regras também pode enviar mensagens para funções Lambda, ou para o serviço de banco de dados DynamoDB. As funções Lambda implementam *microservices* que ficam disponíveis no domínio do AWS IoT. São geralmente serviços com pequenas responsabilidades, totalmente desacoplados de outros serviços. A figura III mostra a adição de uma segunda regra de negócios que avalia se a mensagem tem valor “R”. Em caso positivo, o Motor de Regras envia dado para o DynamoDB, invoca uma função Lambda e providencia o envio de SMS para um dispositivo móvel. Portanto, a qualquer momento é possível alterar o conteúdo no Motor de Regras. Muitos tipos de regras podem ser implementadas no Motor de Regras (Rules Engine). Alguns outros exemplos: enviar mensagem para o Amazon SQS, enviar arquivo para o Amazon S3, enviar dados de mensagens para o *Amazon Machine Learning*. Portanto, mais pesquisa pode ser feita sobre tipos de regras variados, como implementar cada um e suas vantagens. O conhecimento adquirido com estas pesquisas sugeridas poderá ser apresentado oralmente, ao apresentar o AWS IoT.

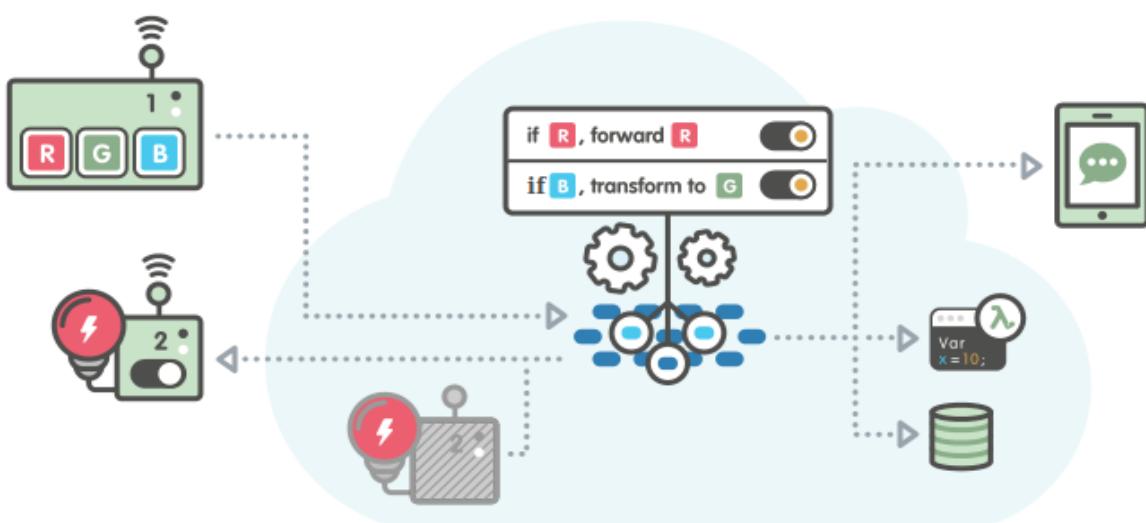


Figura IV

As sombras das coisas

Todos os dispositivos presentes na Internet das Coisas podem ser registrados no AWS IoT. Ou seja, cadastros das respectivas coisas são criados no AWS IoT. As ações a serem dadas sobre as coisas passam a seguir configurações criadas relacionadas com os cadastros. Então, a forma como o AWS IoT agirá sobre uma coisa depende de configurações particulares do cadastro da mesma coisa. E o cadastro de uma coisa pode incluir também a sombra da mesma coisa, ou seja o Shadow. Quando a coisa está desconectada do AWS IoT, por qualquer motivo (Ex: instabilidade de rede) os comandos direcionados à tal coisa são recebidos pelo *shadow*. Assim que a coisa fica online novamente, o *shadow* repassa os comandos a ela. Assim, o controlador da coisa pode continuar exercendo o seu papel, mesmo quando ela está offline. A figura IV mostra a representação da sombra da lâmpada controlada. Cada coisa registrada na nuvem pode ter atributos e um nome. Em casos de falha de comunicação na Internet, a sombra persiste o estado futuro desejado da coisa respectiva.

Construção de soluções

No final das contas, o AWS IoT facilita o desenvolvimento de aplicações que podem atuar como gerentes de coisas na Internet. O exemplo da figura V mostra uma aplicação em um dispositivo móvel, que reflete a cor da luz na lâmpada. O dispositivo móvel nunca tem que se comunicar diretamente com a lâmpada. No máximo, o dispositivo móvel usará uma API REST para ler e modificar o estado do *shadow* da lâmpada. Várias linguagens de programação podem ser usadas para implementar uma aplicação de gerência, a rodar no dispositivo móvel. E o AWS IoT fornece SDK para todas elas. Fornece até mesmo *plugin* do Eclipse para facilitar esse trabalho. Essas ferramentas de programação, fornecidas pelo AWS IoT podem ser pesquisadas em detalhes e o resultado do novo conhecimento a ser adquirido pode ser apresentado oralmente, quando o assunto AWS IoT for demonstrado pelo aluno. Outro assunto interessante para incluir em tal pesquisa é o uso de *Shadows*. A maior contribuição a ser dada em sala de aula, ao apresentar o AWS IoT é explicar os detalhes sobre *Device Shadow !!* Essas pesquisas podem ser todas realizadas no material [2].

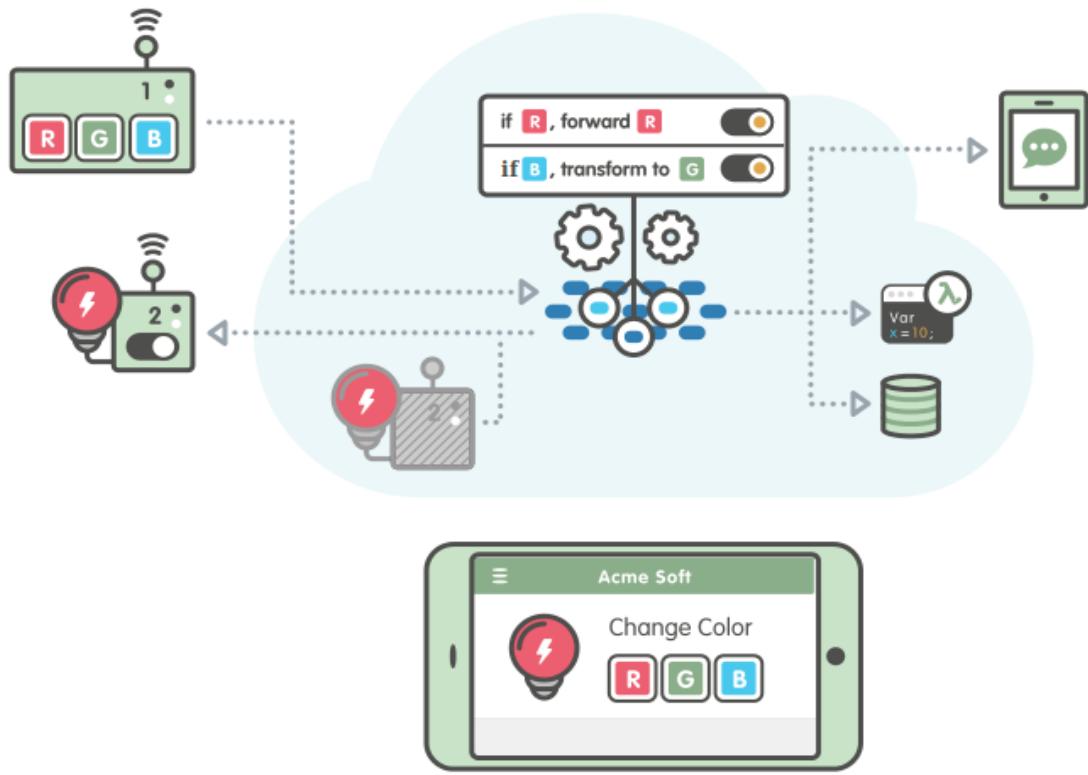


Figura V

Para quem irá apresentar oralmente sobre o AWS IoT, recomenda-se fortemente que antes seja feita leitura do documento “**Core Tenets of IoT**” visto aqui [3]. E caso alguém se interesse em desenvolver para AWS IoT, a página com recursos para o desenvolvedor está aqui [4].

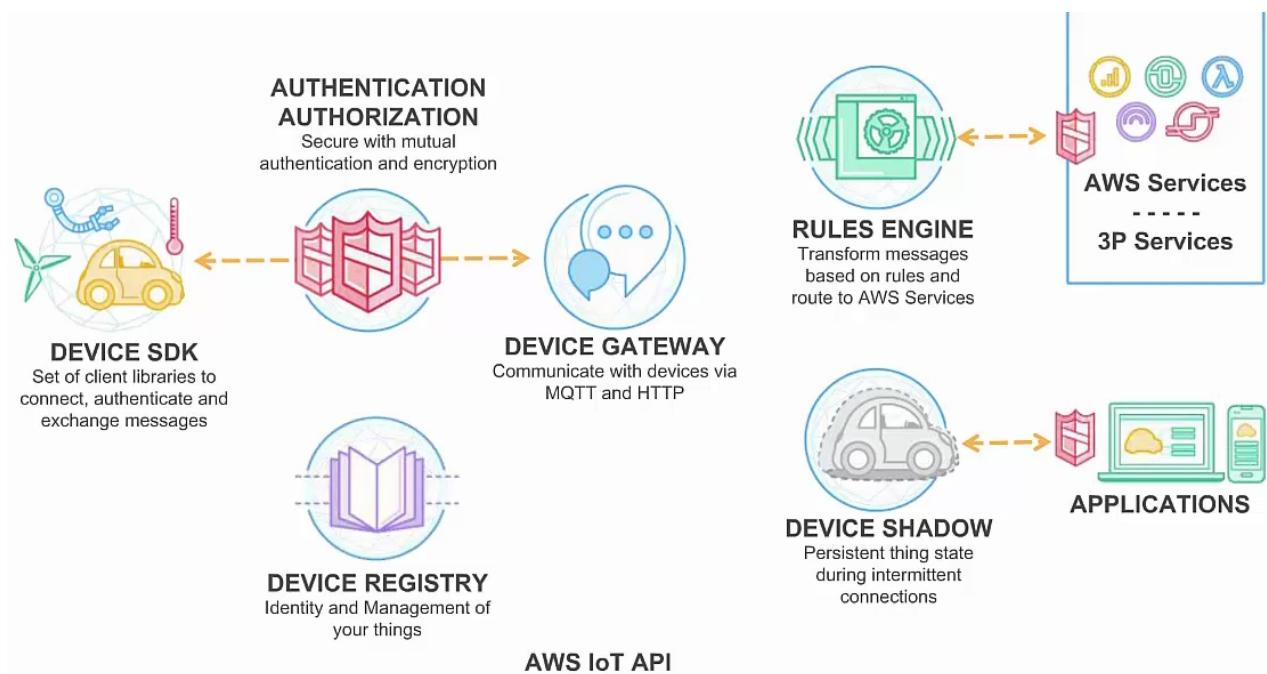


Figura VI

A figura VI mostra um apanhado geral dos componentes do AWS IoT que serão utilizados no caso exemplar a seguir.

Construindo um caso de uso com AWS IoT.

Imagine o seguinte projeto: um sensor de presença ligado a uma placa DragonBoard 410C, instalados dentro de uma residência, devem emitir alertas quando um invasor adentrar o recinto. Ou seja, quando o sensor de presença detectar a invasão, ele enviará um sinal à DragonBoard, que por sua vez enviará uma mensagem ao AWS IoT, contendo o número do celular de quem precisa ser alertado. Com essa mensagem recebida, o AWS IoT irá interpretá-la, obter informação de valor (número de celular) e desencadear um SMS com o aviso de alerta à quem tem interesse.

Essa situação pode ser suportada pelo AWS IoT e portanto as próximas páginas desse capítulo estão escritas no estilo tutorial, para demonstrar detalhadamente como usar as tecnologias em tal caso. O Aluno dessa disciplina pode simplesmente seguir o tutorial abaixo, para exercitar na prática o que pode ser aprendido agora. Vale a pena seguir o tutorial para aprender e fixar bem o conhecimento disponível aqui. Seguir tal tutorial é a forma mais garantida de aprender o que se segue.

Portanto, faça esse exercício! Veja a próxima página:

1 – Entre no Amazon Web Services. Isso consiste em executar os seguintes passos:

- a) Acesse o link <https://rpimentac.signin.aws.amazon.com/console>
- b) Na web page que se abre, insira o seu IAM user name (Ex: **Aluno-X**).
Insira também a senha comum a todos os alunos, que é: **1alunolegal**
Mantenha o Account ID or alias = **rpimentac**
- c) Clique no botão [**Sign in**].
- d) Se abrir um aviso sobre usar **A nova página inicial do console**, aceite usar.

2 – Certifique-se que você usará os recursos da região de disponibilidade da Virgínia do Norte.

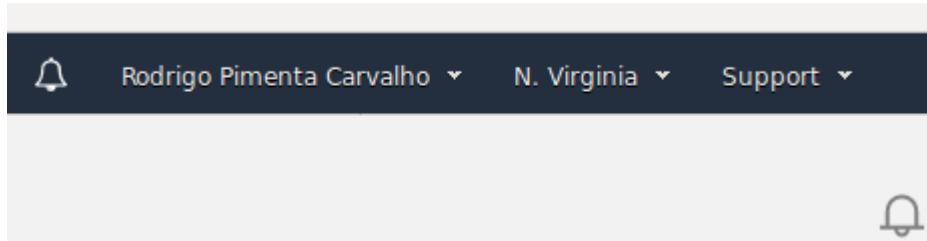


Figura 1

A figura acima mostra o nome do usuário que acabou de acessar o Amazon Web Services e ao lado desse nome a região de disponibilidade escolhida. Mantenha em **N. Virginia**.

3 – Acesse o AWS IoT. Basta pesquisar por **iot core e acessar o serviço.**

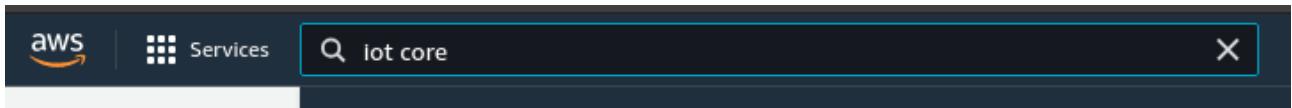


Figura 2

4 – Agora vê-se uma página como essa abaixo, com um menu de opções à esquerda:

A screenshot of the AWS IoT service landing page. The left sidebar has a 'AWS IoT' section with links like 'Monitorar', 'Atividades', 'Conectar', etc. The main content area has a title 'AWS IoT' and subtext 'Soluções de IoT para os setores industrial, de consumo e comercial'. It includes a 'Comece a usar o AWS IoT' section with a 'Conectar dispositivo' button and a 'Preço' section with 'Calculadora de custos' and 'Detalhes de preços do AWS IoT Core'.

Figura 3

O próximo passo deve ser o cadastro da coisa. Ou seja, na IoT as coisas gerenciadas/tratadas, se manipuladas através do AWS IoT, precisam ter cadastros no Amazon Web Services. Então, aqui faz-se o cadastro de uma coisa no AWS IoT. Nesse mesmo momento, outras coisas já podem estar

cadastradas no AWS IoT. Os outros cadastros já presentes não exercem qualquer interferência lógica no próximo cadastro a ser feito. Veja a figura seguinte. Deve-se escolher a opção Gerenciar->Coisas.

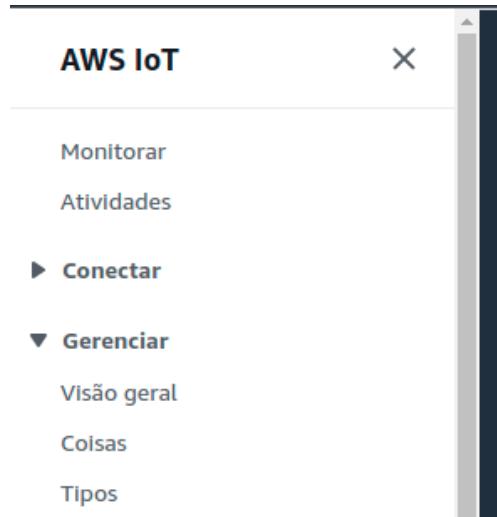


Figura 4

5 – Será apresentada uma página web semelhante a esta vista na figura 5 abaixo. Então, com o botão [Criar itens], crie o cadastro da sua coisa. Como dito acima, despreze qualquer coisa já cadastrada aqui. Essas coisas já cadastradas estão relacionadas a outros dispositivos de outros projetos, de outras pessoas, por exemplo. Mas, todas essas pessoas tiveram permissão de acesso a esse AWS IoT. Ou seja, o administrador de uma conta no AWS dá permissão a todas essas pessoas, para que elas mantenham o cadastro de suas coisas. Essas pessoas podem ser, por exemplo, alunos de um mesmo curso sobre AWS IoT. É apropriado manter diferentes nomes para as coisas cadastradas numa mesma conta de administrador do AWS IoT. Veja detalhes a seguir:

Nome
Coisa_Pimenta
COISA_LIXO

Figura 5

6 – Ao clicar no botão [Criar itens] – uma janela será apresentada como esta na figura seguinte. Siga as instruções logo após a figura.



Figura 6

Escolha [Criar um único item]. O cadastro para uma única coisa é suficiente. Contudo, o aluno poderia estudar como trabalhar com o cadastro de muitas coisas ao mesmo tempo e explicar em sala de aula sobre isso, oralmente com uma apresentação, por exemplo.

7 – Cite o nome que você quer dar a sua coisa. Por exemplo: DragonBoard410C, Coisa-30, Coisa-X, etc. Se você é a aluna “Maria Fialho”, por exemplo, então a coisa deveria se chamar Coisa-Maria_Fialho, preferencialmente. Além de citar um nome, cite também um atributo e um valor para a coisa. Por exemplo, um atributo poderia ser o nome do dono da coisa física, ou seja, o nome de quem está cadastrando-a. É interessante criar mais esse atributo, que deve diferir entre as coisas, para que isso seja usado na geração de certificados para criptografia. Tal diferença repercutiu na diferença de conteúdo nos certificados, o que é conveniente. Os outros valores podem ficar em branco. O nome que você escolher agora será visível a outros usuários desse AWS IoT (alunos desse curso).

Especificar propriedades do item Informações

Um recurso de item é uma representação digital de um dispositivo físico ou de uma entidade lógica no AWS IoT. Seu dispositivo ou entidade precisa de um recurso de item no registro para usar recursos do AWS IoT, como device shadows, eventos, trabalhos e recursos de gerenciamento de dispositivos.

Propriedades do item

Nome da coisa

Coisa_Maria_Fialho

Insira um nome exclusivo contendo apenas letras, números, hifens, caracteres de dois pontos ou sublinhados. Um nome de coisa não pode conter espaços.

Configurações adicionais

Você pode usar essas configurações para adicionar detalhes que podem ajudar a organizar, gerenciar e pesquisar suas coisas.

► **Tipo de coisa - opcional**

▼ **Atributos de coisas pesquisáveis - opcional**

Adicione atributos pesquisáveis para permitir que sua coisa seja agrupada e pesquisada sem usar a indexação de frota.

Atributo pesquisável

Valor - *opcional*

NomeDoDono

Maria_Fialho

Remover

Adicionar novo atributo

Você pode adicionar até mais 2 atributos.

► **Grupos de coisas - opcional**

► **Grupo de faturamento - opcional**

Figura 7

Escolha “Sombra sem nome (clássico)” e Clique no botão [Próximo].

8 - Em seguida devem ser criados os certificados (público e privado) para serem utilizados na criptografia das mensagens MQTT com TLS. Portanto, aqui cria-se um novo *resource*. As coisas reais só poderão se comunicar com o *gateway* AWS IoT se as mensagens estiverem criptografadas. E essa criptografia exigirá o uso de um certificado privado. As mensagens a serem criptografadas estarão usando o protocolo MQTT.

Na janela visível nesse momento, vista na figura 8, escolha gerar um novo certificado automaticamente e clique no botão [Próximo]. É possível cadastrar a coisa sem os certificados. Mas, isso seria em vão, porque eles teriam que ser criados posteriormente, antes que se tente fazer a coisa real se comunicar com o *gateway*. Um certificado com chave privada e pública será criado. A chave privada vai para a coisa na Internet (nesse caso a COISA-X ou “Coisa_Maria_Fialho” por exemplo) e a chave pública fica no próprio *gateway* de dispositivos.

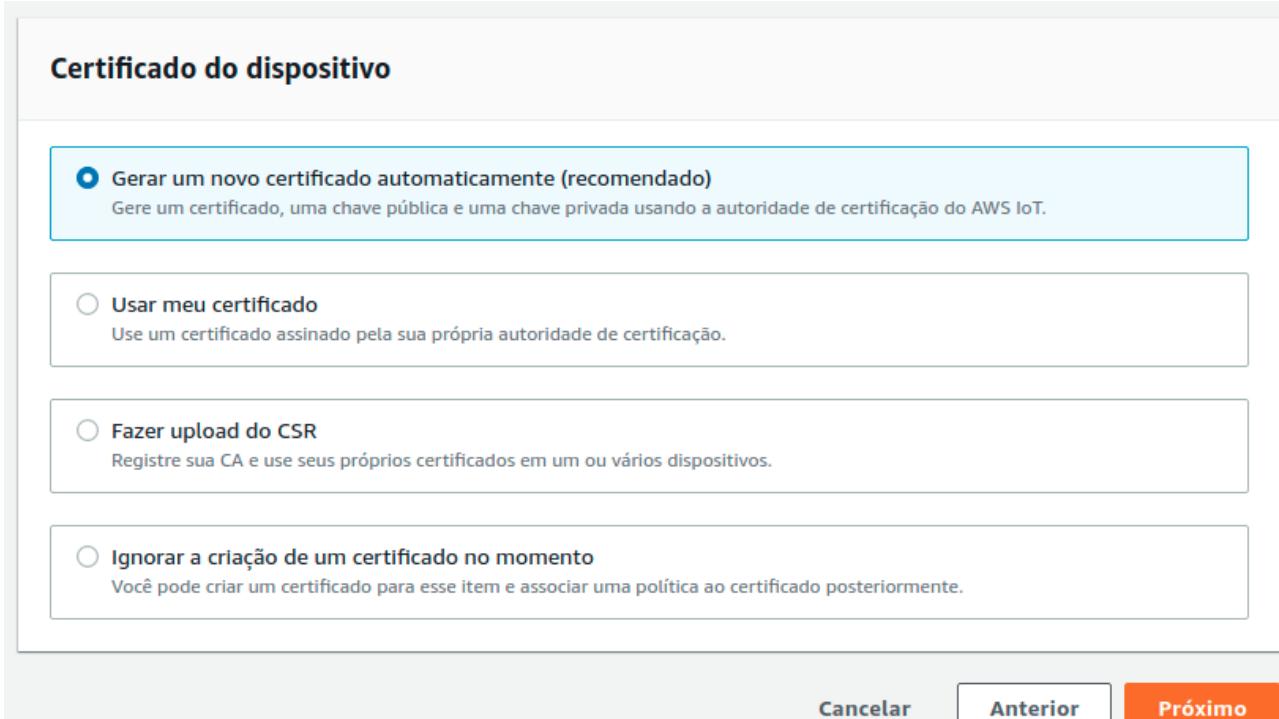


Figura 8

9 – Nesse momento, despreze a criação de Políticas a serem relacionadas com os certificados. Isso será feito mais tarde. Então, clique em **[Criar item]**.

ATENÇÃO! Após clicar no botão **[Criar item]** serão disponibilizados *links* para downloads de certificados e chaves. Faça o download obrigatoriamente antes de fechar a janela com tais *links*.



Figura 9

Ao todo são 5 arquivos diferentes para se obter com downloads agora. Não desative o certificado!

Ao fazer os **downloads**, salve os arquivos com nomes pertinentes. Ex: *Certificado.pem*, *ChavePublica.key* e *ChavePrivada.key*. Assim será possível usar o arquivo correto, quando necessário. **Atenção:** não esquecer onde os arquivos estão sendo salvos por você !! Veja exemplos

de nomes na figura seguinte:

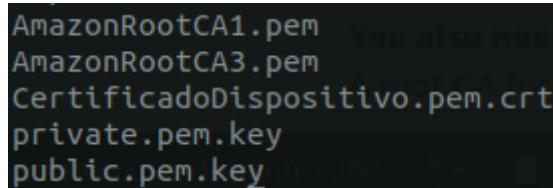


Figura 10

Após fazer o download dos arquivos de certificados e chaves, pode clicar no botão [**Concluído**].

Agora você deve estar vendo 2 avisos como esses na figura abaixo:

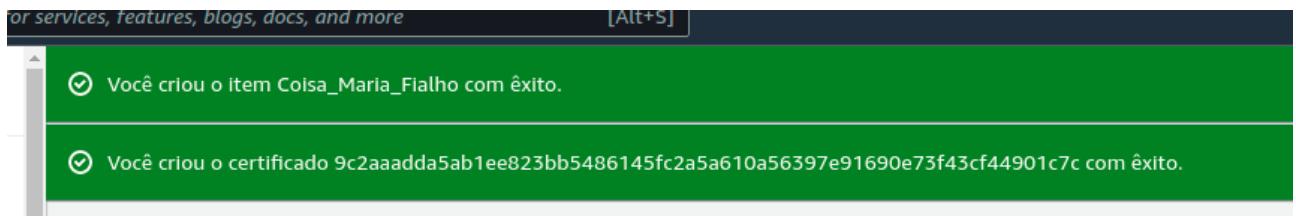


Figura 11

Esses arquivos salvos serão usados mais tarde.

Finalmente a sua coisa está cadastrada.

Agora é um bom momento para recapitular o que foi feito:

- Um cadastro de coisa da Internet foi criado no AWS IoT. Essa coisa recebeu o nome de Coisa-X.
- Na página de Cosas podem ser vistas todas as coisas cadastradas até agora por todos os usuários com permissão de acesso nessa mesma conta do Amazon Web Services.
- Para trabalhar com a coisa física, foram criados arquivos de certificados.

De curiosidade: O objetivo de um certificado e uma chave privada é permitir que a coisa se identifique e autentique seu acesso aos recursos do AWS IoT. Ou seja, quando um software rodando na COISA-X física requisitar uma ação em algum serviço do AWS IoT, tal software deverá apresentar-se ao AWS IoT de tal forma que esse software seja reconhecido como confiável e permitido a realmente usar o AWS IoT. Para tal, o software deve usar o certificado e a chave privada para acesso ao AWS IoT, o que implicará em envio de mensagens também criptografadas com TLS. Por outro lado, quando a COISA-X se comunicar com o AWS IoT ela deverá ter a certeza de que está realmente se comunicando com o *gateway* correto. Ou seja, se um sistema falso se passar por *gateway* na Internet, tal sistema deverá ser identificado como inválido. Nesse caso, o sistema na COISA-X necessita de um rootCA. O rootCA é um arquivo que faz o papel de autoridade certificadora do AWS IoT. Quando o sistema na COISA-X se comunicar com o *gateway*, o *gateway* deverá fornecer dados condizentes com o rootCA presente na COISA-X.

.....

Vamos observar alguns detalhes sobre a coisa cadastrada.

Clique sobre a **Coisa-X**. Isso trará uma janela onde é possível reeditar o cadastrado criado, observar outros atributos da coisa e criar para ela outras informações (vide figura 12).

Chave	Valor	Tipo
NomeDoDono	Maria_Fialho	Pesquisável

Figura 12

Não deixe de reparar que nesse momento é apresentado o identificador universal da coisa registrada. Ex: `arn:aws:iot:us-east-1:336605837729:thing/Coisa_Maria_Fialho`. Esse identificador (ARN) funciona como um endereço/apontador para uma única coisa registrada no domínio do AWS. Nenhuma outra coisa no AWS terá esse mesmo identificador. É o **identificador de recurso**.

10 – Na janela aberta, representada pela figura anterior, clique na aba **Certificados**. Uma nova informação será mostrada, como aquela vista na figura 13. Nesse momento fica visível todos os certificados relacionados com a coisa. Mas, como somente um certificado foi criado para ela, somente um estará presente aqui. Repare que o certificado é identificado pelo seu nome, que é um valor alfanumérico. Ex: [9c2aaadda5ab1ee82...](#)

ID do certificado	Status
9c2aaadda5ab1ee823bb5486145fc2a5a610a56397e91690e73f43cf44901c7c	Ativo

Figura 13

11 – Entre no certificado para ver maiores detalhes sobre ele. Basta clicar sobre seu ARN, que é um link como visto acima. Exemplo na figura abaixo:

The screenshot shows the AWS IoT Certificates Details page. At the top, there is a large ID: 9c2aaadda5ab1ee823bb5486145fc2a5a610a56397e91690e73f43cf44901c7c. Below it, there is a link labeled "Informações". A "Ações" button is also present. The main section is titled "Detalhes" and contains the following information:

ID do certificado	Status
9c2aaadda5ab1ee823bb5486145fc2a5a610a56397e91690e73f43cf44901c7c	Ativo
ARN do certificado	Criado
<code>arn:aws:iot:us-east-1:336605837729:cert/9c2aaadda5ab1ee823bb5486145fc2a5a610a56397e91690e73f43cf44901c7c</code>	April 10, 2022, 16:56:52 (UTC-0300)
Assunto	Válido
CN=AWS IoT Certificate	April 10, 2022, 16:54:52 (UTC-0300)
Emissor	Expira
OU=Amazon Web Services O=Amazon.com Inc. L=Seattle ST=Washington C=US	December 31, 2049, 20:59:59 (UTC-0300)

At the bottom, there are three tabs: "Políticas" (highlighted in orange), "Coisas" (in blue), and "Não conformidade" (in green).

Figura 14

O certificado tem um ARN, como qualquer recurso no universo do AWS.

12- Tome nota do nome do certificado da sua coisa. Essa informação será importante mais adiante, para permitir distinguir esse certificado de outros criados por outras pessoas nessa mesma conta do Amazon Web Services. Considerando o exemplo da janela acima, a informação que se quer tomar nota é:

9c2aaadda5a... (é suficiente tomar nota do **início do nome do certificado**.)

Um certificado deve sempre estar associado a uma *policy*. Com os certificados criados e ativos, tem-se que criar a política de uso de recursos do AWS IoT, que ficará relacionada com os certificados. Por exemplo, um certificado garante que uma coisa pode mesmo se autenticar e acessar recursos no AWS IoT. Mas, após conseguir o acesso, quais serão os direitos de tal coisa? Ou seja, o que a coisa realmente poderá fazer com o AWS IoT? As regras sobre o que a coisa pode fazer constituem uma política que deve estar associada ao certificado. Nesse momento deve ser criada uma Política.

13 – Volte no menu principal do AWS IoT. Menu de opções localizado à esquerda da página web.

14 – No menu principal, acesse **Proteger → Políticas**.

▼ Segurança

Introdução

Certificados

Políticas

Figure 15

Despreze qualquer política já existente e crie uma nova. Para isso, use o botão [Criar Política] que está visível. Então vem uma página como a figura 16.

Criar política Informações

As políticas do AWS IoT Core permitem gerenciar o acesso às operações do plano de dados do AWS IoT Core.

Propriedades da política

O AWS IoT Core oferece suporte a políticas nomeadas, permitindo que várias identidades façam referência ao mesmo documento de política.

Nome da política

PolicyMariaFialho

O nome da política é uma sequência alfanumérica que também pode conter os caracteres ponto (.), vírgula (,), hifen (-), sublinhado (_), sinal de adição (+), sinal de igual (=) e arroba (@), mas não pode ter espaços.

► **Etiquetas - opcionais**

Documento da política Informações

Uma política do AWS IoT contém uma ou mais declarações de política. Cada declaração contém ações, recursos e um efeito que concede ou nega as ações pelos recursos.

Builder **JSON**

Documento da política

```
1 Version: "2012-10-17",
2 Statement: [
3   {
4     Effect: "Allow",
5     Action: "iot:/*",
6     Resource: "*"
7   }
8 ]
9 ]
10 ]
```

Figura 16

Nessa página , dê um nome para a sua política. Por exemplo, PolicyX. Preencha o Documento da

Política como visto na imagem acima. Para isso, escolha editar o Json.

Isso significará que a política sendo criada permitirá todos os tipos de ações sobre o AWS IoT. É o mesmo que dizer que a coisa física poderá interagir com o AWS IoT e usufruir das capacidades dele. O documento é gravado em Json. Esse texto Json pode ser visto agora, se desejado. Clique no botão [Criar].

15 – Associar a nova Política ao certificado já criado. Agora associa-se a PolicyX ao certificado **9c2aaadda5a... (use o seu certificado !)** Fazendo isso, a coisa que acessar o AWS IoT usando tal certificado terá permissão para executar as funções definidas nessa política.

Portanto, acesse o certificado e use a opção [**Anexar políticas**], vista na imagem abaixo, para fazer a associação. Não escolha o certificado de outra pessoa. Use somente o seu.

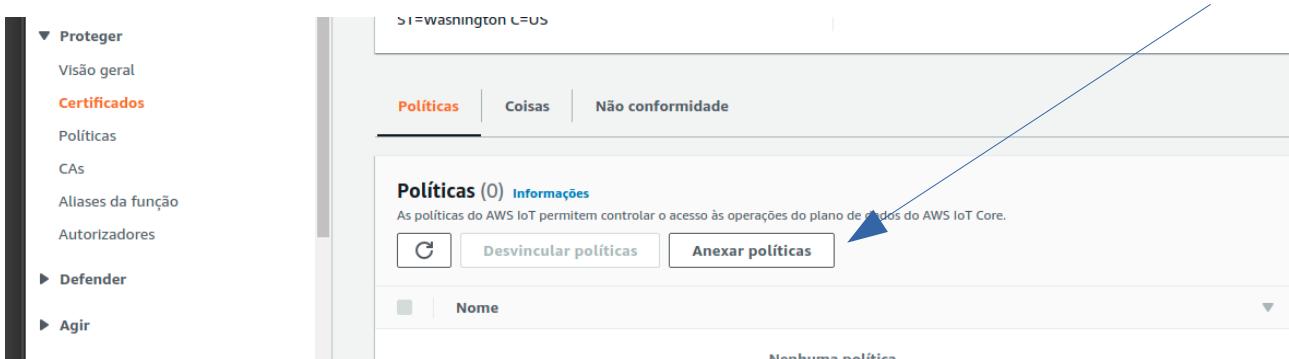


Figura 17

Obviamente, escolha a PolicyX e clique no botão [**Anexar políticas**]. Pronto, a política já está associada ao seu certificado de segurança.

Recapitule, com a sequência abaixo, o que já foi feito até agora:

(Uma coisa foi registrada no AWS IoT, com um nome específico) → (Um certificado de segurança foi criado para tal coisa. Esse certificado é composto por arquivos que podem ser salvos e mantidos com quem vá implementar a comunicação da coisa física com o AWS IoT) → (Uma política de uso do AWS IoT foi definida) → (A política foi associada ao certificado da coisa, de tal forma que isso garante os direitos e privilégios que a coisa tem no AWS, se acessar seus serviços com tal certificado).

De curiosidade, uma política fica definida através de um código JSON que pode ser editado. Portanto, com a prática em JSON, pode-se criar políticas mais elaboradas. Mas, tem-se que saber que atributo e valores pode-se usar.

Para editar (se necessário) o código da política criada recentemente, basta acessar a página da Política. Para isso, basta navegar pelo menu principal do AWS IoT, até ver a informação exemplificada na figura seguinte:

The screenshot shows the AWS IoT Policies interface. On the left, there's a sidebar with various navigation options like 'Atividades', 'Conectar', 'Gerenciar', 'Fleet Hub', 'Greengrass', 'Conectividade sem fio', 'Proteger', and 'Políticas'. The 'Políticas' option is highlighted in red. The main area displays a policy named 'PolicyMariaFialho'. It shows the ARN, the active version (version 1, created on April 10, 2022), and the last update (also April 10, 2022). Below this, there are tabs for 'Versões', 'Destinos', 'Não conformidade', and 'Etiquetas', with 'Versões' being the active tab. Under 'Versão ativa: 1', the JSON content is displayed:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:/*",
      "Resource": "*"
    }
  ]
}
```

Figura 18

16 – Nesse ponto é conveniente aprender a testar o que já está pronto.

Volte no menu principal do AWS IoT. Vá em coisas e clique na sua coisa. Depois, repare nos seus atributos. Clique no menu **Interagir**. E depois clique em **[Visualizar Configurações]** Vem uma página como a imagem mostrada abaixo:

The screenshot shows the AWS IoT Device Data Endpoint configuration page. At the top, it says 'AWS IoT > Configurações'. The main section is titled 'Configurações' and 'Endpoint de dados do dispositivo'. It explains that devices can use the device data endpoint to connect to AWS. Below this, it states that each device has an API REST endpoint and that MQTT clients and AWS IoT SDKs use this endpoint. The 'Endpoint' field is shown with the value 'a91cdtjena4dx-ats.iot.us-east-1.amazonaws.com'.

Figura 19

Nessa figura pode-se ver o *endpoint* para o qual as coisas poderão enviar suas requisições. Essas requisições são então recebidas no AWS IoT. Ou seja, existe uma API atrás desse *endpoint*, pela qual é possível interagir com o AWS IoT. Esse *endpoint* será necessário num código *javascript* a ser feito adiante. Em tal API, suas capacidades serão alcançáveis em endereços com seguinte Prefixo do tópico MQTT:

`$aws/things/Coisa_Maria_Fialho/shadow` (esse é um exemplo particular. Cada aluno terá o seu)

Para ver mais detalhes sobre isso, acesse a sua coisa e escolha ver Device Shadows. Depois clique em *Shadow Clássico*. Serão vistas informações como essas abaixo (em Tópicos MQTT):

The screenshot shows the AWS IoT Device Shadow details page for the item 'Coisa_Maria_Fialho'. At the top, there's a breadcrumb navigation: AWS IoT > Gerenciar > Itens > Coisa_Maria_Fialho > Shadow clássico. Below the navigation, the title 'Shadow clássico' is displayed. A section titled 'Detalhes do device shadow' contains the following information:

ARN	Última atualização
arn:aws:iot:us-east-1:336605837729:thing/Coisa_Maria_Fialho	April 10, 2022, 19:07:00 (UTC-0300)
Prefixo do tópico MQTT	Versão
\$aws/things/Coisa_Maria_Fialho/shadow	1
URL do device shadow	
https://a91cdtjena4dx-ats.iot.us-east-1.amazonaws.com/things/Coisa_Maria_Fialho/shadow	

Below this, there are two tabs: 'Documento do device shadow' (selected) and 'Tópicos MQTT' (highlighted in orange). The 'Tópicos MQTT' tab leads to a table titled 'Tópicos MQTT Informações' which lists the MQTT topics for this device shadow:

Nome	Ação	Tópico MQTT
/get	Publicar	\$aws/things/Coisa_Maria_Fialho/shadow/get
/get/accepted	Assinar	\$aws/things/Coisa_Maria_Fialho/shadow/get/accepted
/get/rejected	Assinar	\$aws/things/Coisa_Maria_Fialho/shadow/get/rejected
/update	Publicar	\$aws/things/Coisa_Maria_Fialho/shadow/update

Figura 20

Pelo visto, tem-se vários tópico MQTT que poderão ser usados, se necessário. Dois desses tópicos MQTT serão usados para testes em breve.

Como explicado antes, cada coisa cadastrada contém um *shadow*. Para atualizar o *shadow*, o que repercute em atualização na coisa real, pode-se usar REST e enviar as mensagens de atualização para um *endpoint* definido então. No caso da CoisaX, o *endpoint* especificado é demostrado na página exemplificada na figura 19. Ou seja, uma ação executada no próprio AWS IoT pode atualizar o *shadow* da coisa, mas as requisições de atualizações são todas envidas para o *endpoint* respectivo. O *gateway*, ao verificar o recebimento de uma mensagem num *endpoint* passa essa mensagem ao interessado. Ou seja, passa para um serviço capaz de atualizar o *shadow*. Todas as mensagens enviadas ao *gateway* são armazenadas no *broker* de mensagens, em filas, até que sejam entregues aos interessados. Ou seja, a forma como o *gateway* armazena as mensagens ainda não tratadas é através de *message broker*.

Por outro lado, a coisa física também pode interagir com o *gateway* do AWS IoT, para atualizar status do *shadow* respectivo. Nesse caso, a coisa física precisa enviar mensagens ao *gateway*. Essas mensagens são enviadas para tópicos específicos, com o protocolo MQTT. A coisa que envia uma

mensagem ao *gateway*, para atualizar um *shadow*, pode fazer isso usando REST. Neste caso, a coisa deve usar a URL composta por: *endpoint* demonstrado + porta 8443 + um tópico. Essa mensagens enviadas ao *gateway*, a partir de coisas reais, também ficam enfileiradas no mesmo *message broker*.

É interessante reparar que há tópicos para atualização dos estados do *shadow* e também para obter o estado ou deletar o *shadow*. Para tal, basta que uma mensagem seja enviada para esses tópicos, com o devido conteúdo. Por exemplo, se a coisa na Internet é uma lâmpada e é necessário mudar seu estado para cor vermelha, a seguinte mensagem em JSON poderia ser enviada a partir de uma aplicação :

```
{  
    "state": {  
        "reported": {  
            "color": "red"  
        }  
    }  
}
```

E essa mensagem seria enviada para um tópico similar a esse exemplo:

\$aws/things/myLightbulb/shadow/update

Para o desenvolvimento de uma aplicação que vá interagir com o AWS IoT para controlar coisas na Internet, existem os seguintes SDKs fornecidos pelo Amazon Web Services:

[Arduino Yún SDK](#)

[AWS IoT Device SDK for Embedded C](#)

[AWS IoT C++ Device SDK](#)

[AWS IoT Device SDK for Java](#)

[AWS IoT Device SDK for JavaScript](#)

[AWS IoT Device SDK for Python](#)

O *endPoint* nesse curso é o mesmo para qualquer aluno, já que ele representa o mesmo *gateway* de dispositivos, porque todos estão usando a mesma conta de AWS IoT.
Tome nota do *endpoint* (salve-o no notepad, por exemplo).

REPARAR MUITO BEM QUE ESTÁ DEFINIDO UM TÓPICO ONDE RECEBER MENSAGENS DA SUA COISA. É O TÓPICO PARA UPDATE .

Cada aluno tem o seu próprio tópico onde receber mensagens. Se não fosse assim, as mensagens de diferentes alunos iriam se misturar numa mesma fila do *broker*. Tome nota também do seu tópico para updates.

Ex:

endpoint a91cdtjena4dx.iot.us-east-1.amazonaws.com

tópico para update \$aws/things/Coisa-X/shadow/update

17 – Então agora volte no menu principal do AWS IoT e acione a opção para testes → Cliente MQTT. Veja abaixo a página web que deve ser aberta:

Cliente de teste MQTT Informações

Você pode usar o cliente de teste MQTT para monitorar as mensagens MQTT que estão sendo transmitidas na sua conta da AWS. Os dispositivos publicam mensagens MQTT identificadas por tópicos para comunicar seu estado ao AWS IoT. O AWS IoT também publica mensagens MQTT para informar os dispositivos e aplicativos sobre alterações e eventos. Você pode assinar tópicos de mensagens MQTT e publicar mensagens MQTT em tópicos usando o cliente de teste MQTT.

The screenshot shows the AWS IoT MQTT test client interface. At the top, there are two tabs: 'Assinar um tópico' (selected) and 'Publicar em um tópico'. Below the tabs, there is a section titled 'Filtro de tópicos' with a sub-section 'Informações'. It contains a text input field labeled 'Insira o filtro de tópicos'. Below this, there is a link 'Configuração adicional' and a prominent orange button labeled 'Inscrever-se'.

Figura 21

Na figura 21 está apresentada a página que simula um cliente MQTT. Com esse cliente, pode-se fazer uma subscrição a um tópico de um *endpoint* ou publicar uma mensagem em um tópico. Portanto, fica possível testar se está tudo ok com o cadastro da coisa registrada. No caso, a CoisaX. Por exemplo, uma mensagem pode ser enviada a um tópico do *endpoint* dessa coisa. E se essa mensagem puder ser recebida através desse mesmo *endpoint*, então isso mostrará que até agora está tudo correto com o cadastro da coisa. Com esse simulador de cliente, uma mensagem será enviada e a mesma será recebida pelo próprio simulador. Ou seja, o simulador publica, mas também consome a mesma mensagem, por estar subscrito como assinante do mesmo tópico em questão.

Agora, preencha o campo **Assinar um tópico** com o tópico escolhido. Ex: **\$aws/things/Coisa-X/shadow/update**. Em seguida, clique no botão [**Inscrever-se**]. Feito isso, o cliente MQTT está pronto para receber mensagens desse tópico. Agora resta publicar a mensagem.

Como visto na figura seguinte, preencha a mensagem com um texto e clique no botão para publicá-la. **[Publicar]**. Se tudo estiver certo com o cadastro da Coisa-X, esse simulador de cliente MQTT receberá essa mesma mensagem publicada imediatamente.

The screenshot shows the AWS IoT MQTT test client interface with the 'Publicar em um tópico' tab selected. It includes fields for 'Nome do tópico' (with placeholder text '\$aws/things/Colsa_Maria_Fialho/shadow/update') and 'Carga da mensagem' (containing a JSON object: { "message": "Saudações do console do AWS IoT" }). Below these fields are links for 'Configuração adicional' and a prominent orange button labeled 'Publicar'.

Figura 22

Ao publicar a mensagem, ela deverá ser recebida e demonstrada nessa mesma página web, como visto na figura 23, por exemplo:

The screenshot shows the AWS IoT Things shadow update interface. At the top, there is a URL field containing '\$aws/things/Coisa_Maria_Fialho/shadow/update'. Below it are four buttons: 'Pausar' (Pause), 'Limpar' (Clear), 'Exportar' (Export), and 'Editar' (Edit). The main content area displays a message entry under the heading '\$aws/things/Coisa_Maria_Fialho/shadow/update' with a timestamp 'April 15, 2022, 09:06:43 (UTC-0300)'. The message content is a JSON object: { "message": "Saudações do console do AWS IoT" }.

Figura 23

Se a mensagem acima foi demonstrada, então o cadastro da Coisa-X está Ok até aqui. Go ahead!

18 - O próximo passo é criar o cliente (código) que deve executar na coisa física. Para esse curso, a coisa será seu próprio computador. Um computador também pode ser uma coisa da IoT, obviamente. Ou seja, o computador simulará um dispositivo qualquer na IoT (como uma placa BBB ou DragonBoard) e o código a ser executado no computador representará a inteligência que se quer colocar no dispositivo (na coisa).

O que deveria ser feito agora é o download do SDK da linguagem de preferência, já fornecido pelo AWS IoT. Tal SDK já conterá exemplos prontos de como fazer a conexão com o *gateway*, em termos do código fonte cliente a ser executado na coisa. Para a demonstração de código seguinte, foi escolhida a linguagem javascript. *"The aws-iot-device-sdk.js package allows developers to write JavaScript applications which access the AWS IoT Platform via MQTT. It can be used in Node.js environments as well as in browser applications."*

Para esse curso, você mesmo(a) instalará no seu computador o kit de desenvolvimento em JavaScript para AWS IoT. Assim você poderá editar e executar JavaScript na sua máquina, visando comunicação com o AWS IoT. Você executará o seu JavaScript usando o node.js, que você também já deve instalar no seu computador. A comunicação de um código, feito com o SDK para JS do AWS IoT, com a plataforma IoT da AWS, pode ser feita com o protocolo MQTT, com as mensagens criptografadas usando-se o TLS. Nesse caso a porta de autenticação é a 8883.

Considere o anexo II dessa apostila agora, caso não tenha feito isso conforme solicitado previamente por e-mail.

Segue abaixo um exemplo em JS, que pode ser testado com node.js, que faz a comunicação com a plataforma AWS IoT e pode enviar uma mensagem à mesma. A mensagem pode estar escrita em JSON. Para o envio da mensagem, o código faz o papel de um cliente que publica dados num *gateway*. Escreva o código abaixo num editor de texto qualquer e salve no seu computador com o

nome Coisa-X.js. Copie e cole para adiantar. Para facilitar, salve o seu arquivo js no mesmo local onde foram salvos os arquivos de certificados obtidos via download anteriormente.

```
const deviceModule = require('<path para pasta AWSIOT>/node_modules/aws-iot-device-sdk').device;
function processTest() {
    //
    // The device module exports an MQTT instance, which will attempt
    // to connect to the AWS IoT endpoint configured in the arguments.
    // Once connected, it will emit events which our application can
    // handle.
    //
    const device = deviceModule({
        keyPath: 'private.pem.key',
        certPath: 'CertificadoDispositivo.pem.crt',
        caPath: 'AmazonRootCA1.pem',
        clientId: 'Aluno-X',
        region: 'us-east-one', // the AWS IoT region you will operate in (default 'us-east-1')
                               // Região onde os certificados foram criados!
        baseReconnectTimeMs: '1000',
        protocol: 'mqqtts',
        port: '8883',
        host: 'a91cdtjena4dx-ats.iot.us-east-1.amazonaws.com' //Copiado MQTTS endPoint
    });

    //Esse código usa REST e o endPoint acima, para invocar serviço.
    //Dessa forma fica possível enviar dado para um tópico de fila, usando mensagens MQTT.

    console.log('preparando...');
    //
    // Do a simple publish demo
    device
        .on('connect', function() {
            console.log('connect');
            device.publish('$aws/things/Coisa-X/shadow/update',
                JSON.stringify(
                    {
                        "nome" : "<seu nome>", "fone" : "+55359990000"
                    }
                )
            );
            console.log('Mensagem foi publicada!');
        });
}
processTest();
```

Todas as linhas grifadas no código acima contêm valores particulares de usuário e máquina em uso (número de telefone particular, paths particulares, topic particular, etc). Você deve alterar todos esses valores grifados para valores condizentes com o seu usuário de AWS IoT e paths no seu computador, nesse momento. Para clientId use Aluno-X, por exemplo.

O JSON, usado como mensagem enviada pelo código acima, contem a informação que o dispositivo físico (coisa) deve enviar ao AWS IoT, que é seu nome e número de seu celular, nada mais.

A execução do código acima fará o mesmo efeito do teste executado anteriormente. Ou seja, uma mensagem JSON será enviada para um tópico de um endpoint no AWS IoT. Conforme o tópico escolhido nesse exercício, o AWS IoT saberá que a mensagem JSON tem o objetivo de atualizar o shadow da coisa respectiva.

19 - Teste agora o cliente JS. Abra a página de teste do seu AWS IoT , como já foi feito antes. Faça a subscrição do seu cliente MQTT no mesmo tópico, como feito antes. Isso deixará o seu cliente MQTT de testes pronto a receber mensagens do seu outro cliente JS. Nesse caso, o JS enviará uma mensagem JSON para o gateway, que será obtida e demonstrada na tela do cliente MQTT. De olho na tela do cliente MQTT, execute o cliente JS no seu computador, da seguinte forma no prompt de

comando:

node Coisa-X.js

Se a mensagem JSON aparecer no cliente MQTT, ótimo. Você completou 50% do exercício.

Esse exercício será avaliado pelo seu professor, valendo pontos. Até aqui você já faturou 50% dos pontos desse exercício. Portanto, salve a tela do seu computador (anexe a uma mensagem) e envie para o e-mail pimenta@inatel.br. Essa imagem a ser gerada deve ser semelhante àquela vista na figura 24 abaixo. Ou seja, será possível ver qual aluno já completou o exercício até esse ponto e será possível ver qual é o tópico do mesmo aluno. Então, gere uma imagem que contenha o nome e sobre nome completo. Nesse exercício não é obrigatório demonstrar o número real do celular. Pode-se usar qualquer número aí, para fins de privacidade.

Então, nesse momento a página do cliente MQTT simulado deve ter o seguinte aspecto (ou semelhante a ele):

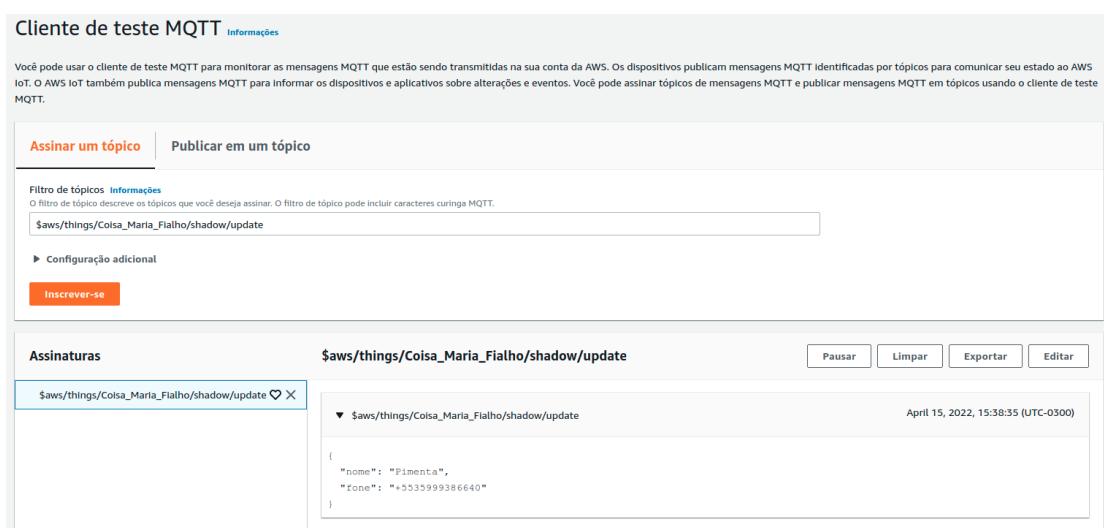


Figura 24

Se isso funcionou, então você já tem um código correto que pode mandar um conteúdo em JSON exatamente para o tópico de UPDATE da sua coisa registrada. Essa mensagem pode então levar atributos e valores a serem atualizados na Shadow, por exemplo.

20 - Com tudo ok é hora de seguir em frente e configurar a plataforma do AWS IoT para que ela, usando o número de celular da mensagem, envie um SMS à pessoa respectiva. Para tal, tem-se que criar uma regra no *gateway*. A regra será a seguinte:

- sempre que uma mensagem chegar no *gateway*, qualquer que seja seu conteúdo, ela será encaminhada a uma função Lambda. O serviço Lambda permite a criação e execução de *microservices* contidos na mesma plataforma do AWS IoT. Portanto, a mensagem recebida será entregue a uma função programada, como parâmetro de entrada. A função Lambda deve ter um nome, como qualquer função em Java ou C por exemplo e deverá desencadear algum trabalho útil. Para tal, deve ser feito o que está explicado a seguir. Ainda na mesma região de disponibilidade do AWS IoT (Ex: N. Virgínia), deve ser acessado o serviço Lambda. Veja abaixo como :

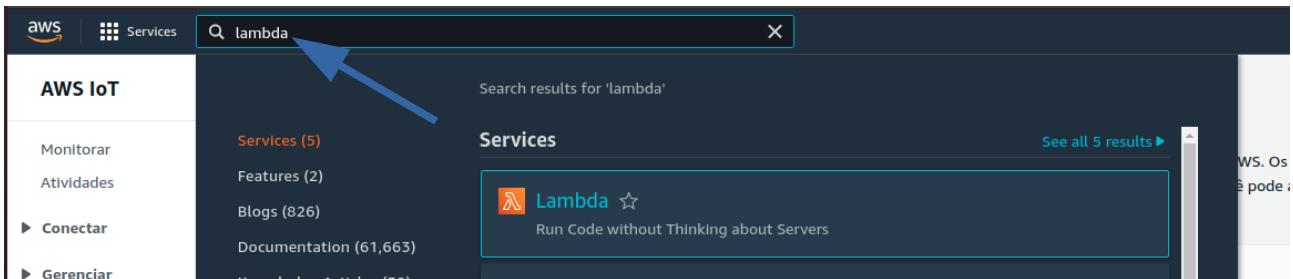


Figura 25

Isso mostrará a gama de serviços do AWS. Escolha Lambda no menu, como visto abaixo:

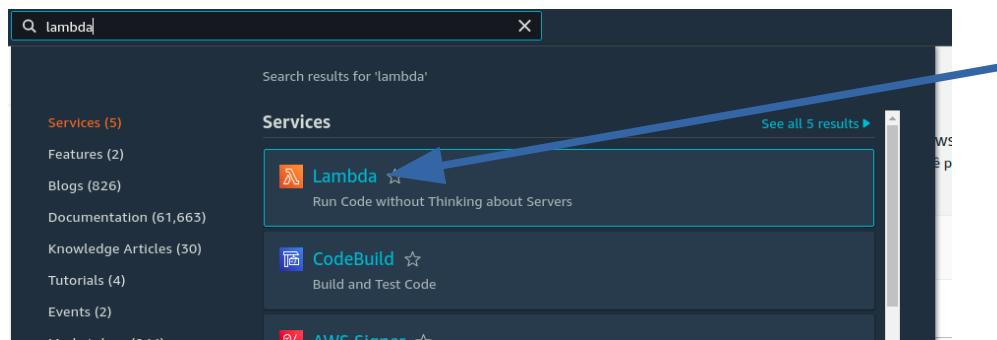


Figura 26

Vem uma janela com um botão para criar uma função Lambda.



Figura -27

Agora basta clicar no botão [Criar Função], que poderá ser criada em uma de muitas diferentes linguagens de programação. Para esse exemplo em estudo, foi escolhida a linguagem *javascript*. Despreze qualquer função Lambda que já existir pronta. Ela pertence a outro usuário.

Na janela que se abre, faça o preenchimento dos campos. Veja figura 28.

Cada função Lambda deve ter definido um “*Lambda execution role*” (papel de execução) . Essa definição explicita exatamente que serviços do AWS a função Lambda poderá acessar e usar. No exemplo seguinte de *role*, há a definição que permite o uso do serviço SNS (qualquer uma das suas capacidades). E pode ser qualquer instância de SNS, não uma específica previamente criada. Portanto, a função Lambda que receber esse *role* poderá somente desencadear uma ação, que será o uso de SNS (*Simple Notification Service*).

Será usado um *role* já pré-definido pelo instrutor desse curso, para agilizar o processo. O nome do *role* é *PushAllNotificationsBestRole*. Cite um nome para a função também. Veja exemplo abaixo.

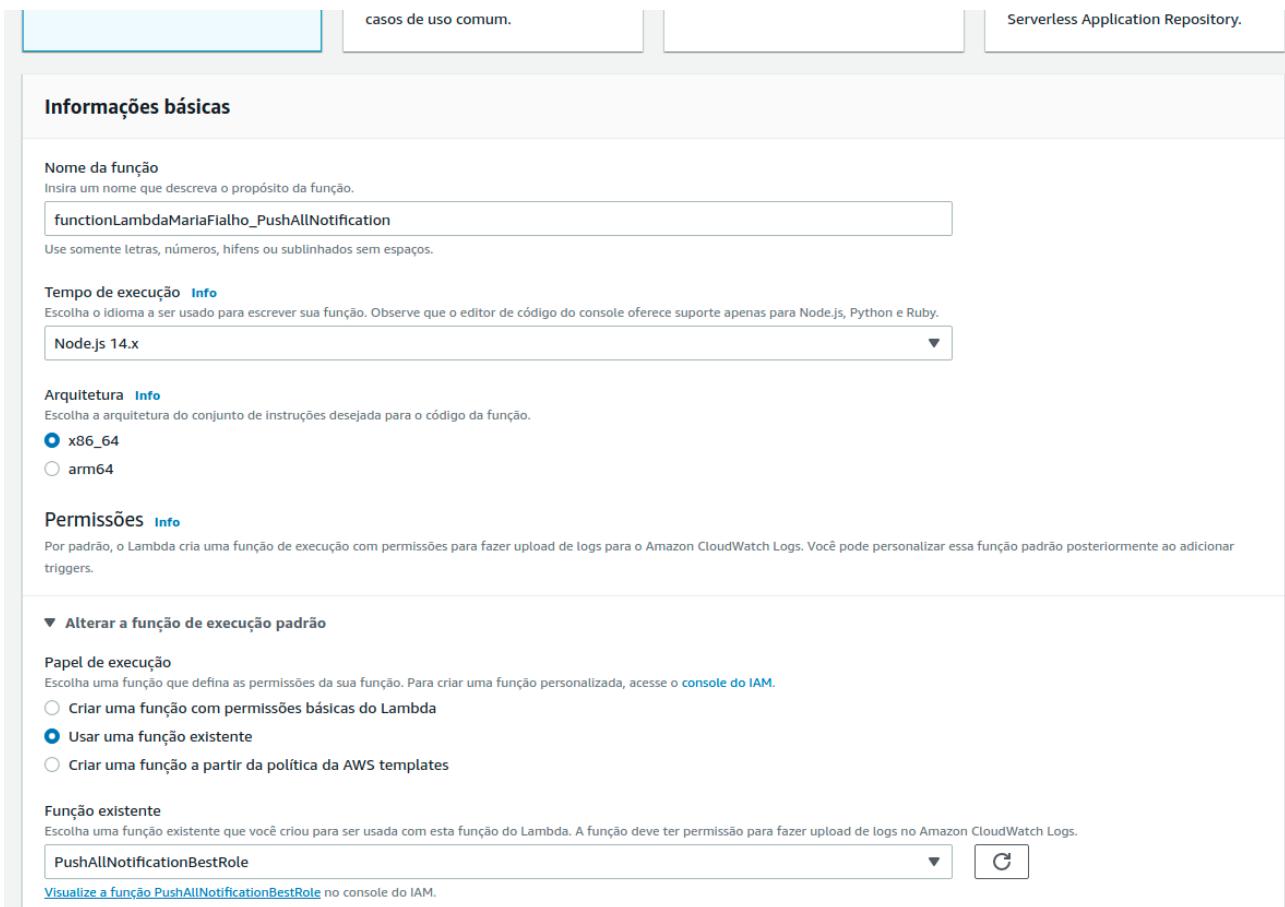


Figura 28

Clique no botão [Criar função].

21 – Adicione o código da função. Na página que se abriu, mais embaixo, escreva o seguinte código (Copie e cole para agilizar. Ao colar, verifique se o hífen de “aws-sdk” permaneceu):

```
console.log("Loading function");
var AWS = require("aws-sdk");
exports.handler = function(event, context) {
    var eventText = JSON.stringify(event, null, 2);
    var jsonObj = JSON.parse(eventText);
    console.log("Received event:", eventText);
    var sns = new AWS.SNS();
    var params = {
        Message:"Olá. Prezado(a) aluno(a) " + jsonObj.nome + ", você está aprendendo sobre AWS IoT?",
        Subject: "Pergunta em aula.",
        PhoneNumber: jsonObj.fone
    };
    sns.publish(params, context.done);
};
```

A página ficará como na figura 29.

É um código bem simples, como deve ser um *microservice*! Esse código recebe um texto JSON como parâmetro, o qual conterá um nome e um número de telefone celular. Esses dados serão extraídos do JSON e passados a um objeto do serviço SNS. No AWS existe um serviço chamado SNS (*Simple Notification Service*), o qual pode, dentre outras coisas, enviar um SMS a um destinatário qualquer (Ex: celular no Brasil). Esse SMS pode apresentar título e conteúdo definidos

no corpo da função Lambda, como visto no código anterior. O SNS pode ser 100% manipulado via console web do AWS, mas também pode ser totalmente manipulado via sua própria API. O Código acima mostra alguns métodos da API do SNS. Usa os métodos 'new' e 'publish', ou seja, um para criar uma instância de SNS e outro para SMS através dele.

Figura 29

22 – Teste a função Lambda criada. A função Lambda pode ser 100% testada no próprio console web do AWS IoT, mesmo antes do código JS (cliente que roda na COISA-X) ser finalmente executado. Vamos testar a função Lambda criada.

Clique no botão para [Test].

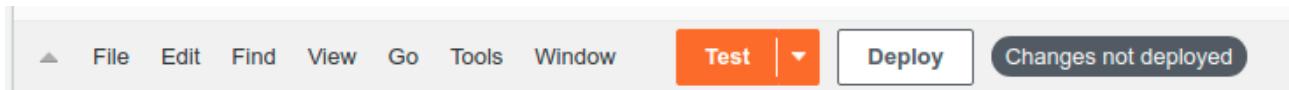


Figura 30

Na janela Input Test Event (Configurar Evento de Teste) que se abre deve ser escrito o JSON com o nome e o número de celular.

(Copie e cole para agilizar) Ex:

```
{
  "nome": "Seu nome",
  "fone": "+5535000003953"
}
```

Veja na figura 31 como a janela deve ser preenchida, para a execução do teste.

Repare que pode-se dar qualquer nome ao evento e que o texto JSON já presente na janela deve ser substituído pelo código acima. Para substituir, sobrescreva o código já presente na janela com um Ctrl+V.

Ou seja, um *template* bem simples (do Hello World) será usado, com código alterado.

Configurar evento de teste

Um evento de teste é um objeto JSON que simula a estrutura de solicitações emitidas pelos serviços da AWS para invocar uma função do Lambda. Use-o para visualizar o resultado da invocação da função.

Para invocar a função sem salvar um evento, configure o evento JSON e, em seguida, escolha Testar.

Ação de evento de teste

Criar novo evento Editar evento salvo

Nome do evento

Máximo de 25 caracteres, formados por letras, números, pontos, hifens e sublinhados.

Configurações de compartilhamento de eventos

Privado
Esse evento está disponível apenas no console do Lambda e para o criador do evento. É possível configurar um total de 10. [Saiba mais](#)

Compartilhável
O evento está disponível para usuários do IAM na mesma conta que têm permissões para acessar e usar eventos compartilháveis.
[Saiba mais](#)

Modelo - *opcional*

JSON do evento

```

1  [
2   "nome": "Seu nome",
3   "fone": "+5535000003953"
4 ]

```

Formatar JSON

Figura 31

Após preencher o código como visto acima, clique no botão [Salvar], visto na figura abaixo:



Figura 32

E clique no botão [Deploy].

23 – No meio da página estarão visíveis as abas, como mostrado na figura seguinte:



Figura 33

Clique então na aba [Testar] e depois no botão [Testar] laranjado. O teste será executado! Já pode olhar no seu celular. Deve haver um novo SMS nele. Você também verá o seguinte aviso na página web aberta:

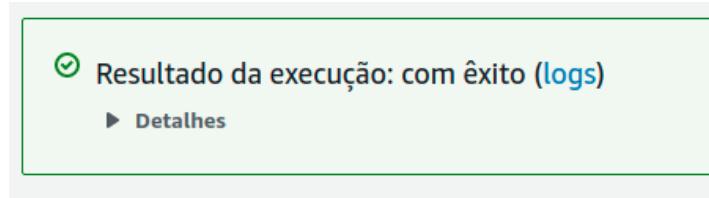


Figura 34

Se esse teste falhou, volte no JSON citado no seu teste (MeuTestX) e verifique se ele está 100% correto, com seu nome e seu número de celular completo, com código de país. Ex: +5535999..... Para voltar ao JSON do teste, será necessário permanecer na aba [Testar], como mostrado na figura abaixo e editar o código do teste:

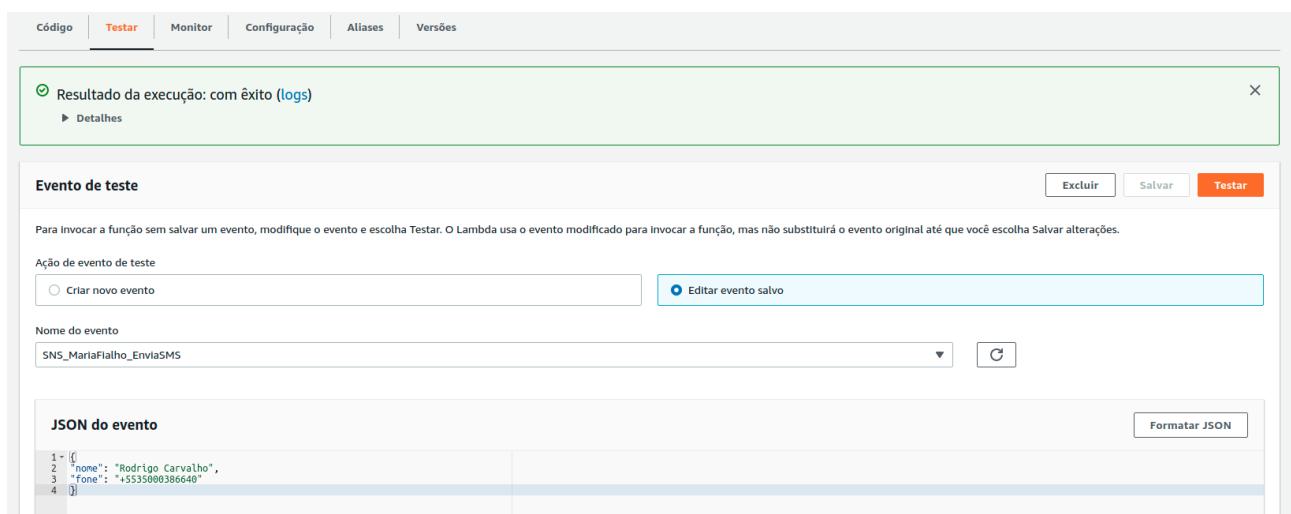


Figura 35

24 – Clique no botão [Salvar] demonstrado na figura 35. E em seguida clique no link **Services** mostrado na figura 25. Volte para o serviço IoT Core.

Nesse ponto já existe um cliente (COISA X) programado para enviar mensagem JSON ao *gateway*, a partir do computador. No *gateway* nos temos um tópico onde as mensagens são depositadas. As mensagens depositadas devem ser entregue à função Lambda criada agora. Para que ocorra a entrega à função, deve-se criar finalmente a regra no motor de regras.

Uma vantagem desse motor é que suas regras podem ser escritas com *queries*, como em SQL. É bom reparar então que o AWS IoT permite criar regras de negócios e relacioná-las com *microservices*. Ou seja, as facilidades providas pelo AWS IoT são realmente muito condizentes com o que se vê em BPM + SOA. Para o caso em estudo, a regra a ser criada terá o seguinte código:

```
SELECT * FROM '$aws/things/Coisa-X/shadow/update'
```

Ou seja, qualquer mensagem será selecionada, independente do seu conteúdo. As regras de negócio no AWS IoT são criadas com a sintaxe do SQL, como visto aqui. No caso atual, serão selecionadas

as mensagens enviadas pelo tópico '\$aws/things/Coisa-X/shadow/update', que é exatamente o mesmo tópico usado até agora nesse caso de estudo.

Para construir a regra, deve ser feito o seguinte:

Clique na opção **Agir->Regras** do menu principal à esquerda. Veja na figura 36.

Despreze qualquer *rule* já existente e clique no botão [**Criar**]. Então vem uma página como aquela mostrada na figura 37. Preencha os campos como mostrado nessa figura 37.

Depois preencha os outros campos como mostrado na figura 38.

▼ Roteamento de mensagens

Regras

Destinos

Figura 36

Criar uma regra

Crie uma regra para avaliar mensagens enviadas por suas coisas e determine o que fazer quando uma mensagem é recebida (por exemplo, gravar os dados em uma tabela do DynamoDB ou Invocar uma função Lambda).

Nome
Colxa_MariaFialho_MotorDeRegras

Descrição
Regra para coletar mensagens de um tópico determinado (de UPDATE) e encaminhar a mensagem a uma função lambda desejada.

Figura 37

Instrução da consulta da regra

Indique a origem das mensagens que você deseja processar com esta regra.

Versão do SQL a ser usada

2016-03-23

▼

Instrução da consulta da regra

```
SELECT <Attribute> FROM <Topic Filter> WHERE <Condition>. Por exemplo: SELECT temperatura FROM 'iot/topic' WHERE temperatura > 50.
```

Para obter mais informações, consulte [Referência de SQL do AWS IoT](#).

```
1 | SELECT * FROM '$aws/things/Coisa_Maria_Fialho/shadow/update'
```

Figura 38

Lembre-se que o *topic filter* a ser escrito acima pode ser encontrado na página de interação com a coisa x.

Em seguida, veja mais abaixo na página web aberta e clique no botão **[Adicionar ação]**. Use o botão mostrado na figura abaixo.

Definir uma ou mais ações

Selecione uma ou mais ações para acontecerem quando a regra acima corresponder a uma mensagem de entrada. As ações definem atividades adicionais que ocorrem quando as mensagens chegam, como armazená-las em um banco de dados, invocar funções da nuvem ou enviar notificações. (*obrigatório)

Adicionar ação

Figura 39

25 – Escolha função Lambda, como mostrado abaixo:

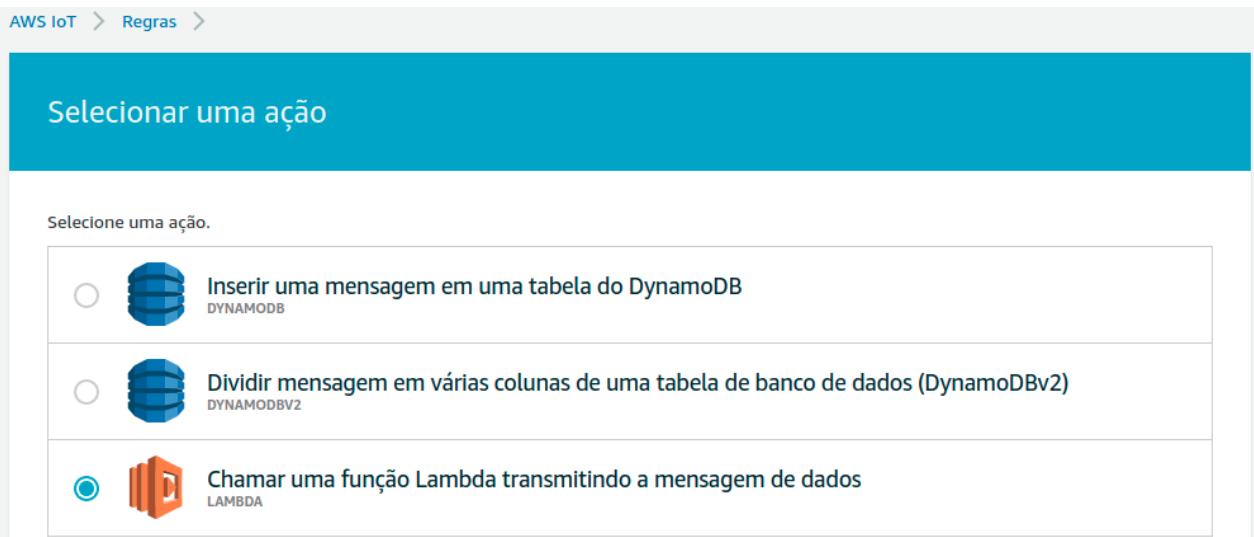


Figura 40

Clique no botão [Configurar Ação].

Na janela que se abre, escolha a sua Ação (sua função Lambda) Veja exemplo abaixo:

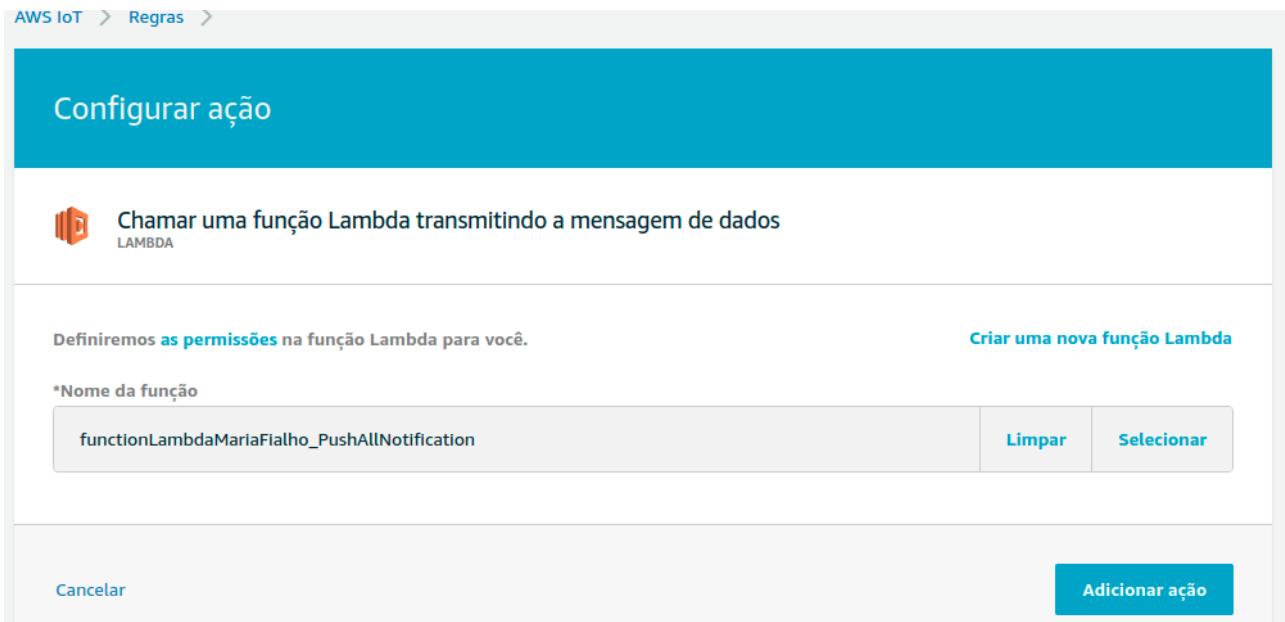


Figura 41

Agora clique no botão [Adicionar ação].

Finalmente, clique no botão [Criar regra].

Isso já deixará a regra (*Rule*) relacionada com a função Lambda previamente criada. Dessa forma, será para a função x_PushAllNotification que a *rule* irá enviar a mensagem que será recebida no *gateway*. Por fim, a função irá então executar e usar uma instância de SNS que enviará o SMS para o celular alvo, fechando assim todo o ciclo de vida desse caso de uso do AWS IoT.

Falando-se em coisas para a Internet, muitas delas poderão aproveitar de alguma forma serviços já prontos e disponíveis em vários lugares do mundo. A forma de aproveitá-los pode ser através de requisições REST, por exemplo. Mas, tem-se que conhecer as interfaces construídas para expor as capacidades de tais serviços. Mas, isso é assunto para outro minicurso ou a pós-graduação do Inatel.

Caso algum aluno queira, poderá realizar pesquisa e apresentação oral sobre o assunto MQTT.

Para terminar esse capítulo relacionado com Computação em Nuvem, é hora de dizer que “empresas optam cada vez mais por infraestrutura em nuvem”, como dito em [1]. É apenas uma questão de tempo até que algum departamento da empresa onde você trabalha/estuda volte a atenção para IaaS e alocar algum colaborador para verificar como usufruir isso e quanto de economia isso trará para a empresa. E muito em breve algum departamento da sua empresa já estará usando recursos computacionais que ficam executando fora dela, em nuvem, devido à força que isso tem em reduzir custos com TI. Por tudo isso, os assuntos de Computação em Nuvem são cada vez mais importantes e o de AWS IoT terá o seu espaço nas discussões futuras em TI. Mas, ainda pensando na empresa onde você trabalha, que tal fazer um estudo dos custos dela com infraestrutura em TI e comparar com o custo que seria gasto se parte dessa estrutura fosse substituída pelo uso da IaaS?

26 – Finalmente rode o seu JS da sua coisa e verifique se você recebe o SMS corretamente

agora.

27 – Para faturar os outros 50% dos pontos desse exercício, modifique o código da sua função Lambda, de tal forma que o texto da mensagem seja:

```
Message:"Olá. Prezado professor, meu nome é " + jsonObj.nome + "e eu completei o meu exercício 100%.",
```

Em seguida, modifique o código no seu *javascript*, no seu computador, para que o número de celular seja : +5535999386640.

E execute o *javascript* novamente no seu computador. Um SMS será enviado ao celular do professor, para que ele saiba que você completou o seu exercício corretamente. Não altere o seu tópico depois de definido, para que o professor veja que sua mensagem realmente passou pelo seu tópico e tudo correu bem. Ou seja, o tópico usado no final desse exercício deve ser igual aquele enviado por e-mail aos 50% desse exercício.

Capítulo II – APIs : Application Programming Interfaces

O assunto de APIs é muito empolgante e relativamente novo no Brasil. Mas, o que é API e porque é empolgante?

As APIs são as interfaces computacionais, documentadas, que estão visíveis para quem olha na direção de certas empresas, do ponto de vista de TI. Ou seja, no mundo de TI (software, sistemas de banco de dados, *web services*, *web servers*, REST, SOA, SOAP, GUIs, etc), uma empresa pode ser vista através de uma interface em software. Ou seja, a interface representa as capacidades computacionais da empresa e portanto deixa disponível o acesso a certos métodos muito úteis a quem precisa processar ou adquirir dados relacionados com o negócio principal de uma empresa. Uma analogia pode ajudar aqui. Considere um carro. Qual é o negócio primordial de um carro? R: transportar pessoas e cargas. Quais são as capacidades de um carro? R: aceleração, armazenamento de combustível no tanque, iluminação com farol, alertas de porta aberta, etc. Quais são as informações fornecidas por um carro? R: nível do tanque, status da velocidade atual, quilometragem percorrida, alarme de porta aberta, status do farol, etc. Então, o painel interno do carro poderia ser visto como uma das interfaces do mesmo, onde é possível dar comandos ao automóvel e receber informações sobre o veículo ou sobre status dos comandos dados. Mais precisamente, com esta interface é possível por o carro a trabalhar. Da mesma forma, uma interface de uma empresa permite colocar algumas de suas capacidades em funcionamento para gerar trabalhos (dados) úteis. Por exemplo, a API pública das capacidades do site Extra permite que sejam usados os serviços de comércio eletrônico e divulgação de produtos do mesmo a tal ponto que, em termos de desenvolvimento de software, fica possível colocar à venda produtos de terceiros em tal site, aproveitando assim da marca Extra e sua credibilidade. Portanto, mesmo que ninguém da empresa dispenda esforços para introduzir novos produtos em seu site, ela ainda pode contar com este tipo de evolução, porque haverá terceiros interessados nisso e que irão sim atualizar a gama de produtos à venda. E é claro que com o uso da API a empresa poderá lucrar, já que estará fornecendo algum serviço com suas capacidades nativas.

As APIs são então como interfaces, mas muito bem documentadas para que terceiros consigam entender e usar as mesmas, sem que a empresa dona tenha que gastar esforços com a disseminação das mesmas. Atualmente as empresas já estão percebendo as vantagens em expor suas capacidades comuns de seus negócios principais, afim de adquirirem mais visibilidade no mercado e/ou lucros diretos. E para os desenvolvedores de software, surge uma grande gama de serviços disponíveis, muitos gratuitos, o que possibilitará sim a criação de mais e mais soluções em software inovadoras. Até já existe uma empresa no Brasil especializada exatamente no assunto “trabalhar com APIs”.

Portanto, as APIs são a nova forma de “colar” (interconectar) as empresas entre si, como uma conexão entre serviços e sistemas que os demandam, criando oportunidades a quem dominará esta nova tecnologia para essa nova filosofia de negócios: os desenvolvedores de software, principalmente de alto nível.

Esse assunto é empolgante ainda mais quando se vê quantas APIs e quais delas já existem! A gama

de possibilidades de usos é realmente grande e assim que a população de desenvolvedores de software perceber a riqueza contida nas mesmas, muitos partirão para a criação de software como se estivessem brincando de montar objetos com peças acopláveis.

O poder das APIs

A adoção das APIs pela população vem ocorrendo através de atividades do cotidiano, mesmo que as pessoas não tenham se atentado para isso. Mas, além da adoção ocorrer, ela é bem aceita naturalmente. Então, se por um lado os desenvolvedores de software precisam entender as APIs e visualizar as vantagens das mesmas, por outro a população de usuários comuns precisa acessar e usar software útil que facilite a vida deles, justamente porque tais softwares estarão usufruindo o poder das APIs. Por exemplo, algumas atividades do cotidiano podem ser destacadas aqui, da forma como eram feitas no passado e da forma como são feitas agora:

Atividade do cotidiano no passado	Atividade substituta, mas de mesmo efeito básico.
Tomar notas em pedaço de papel.	1 - Usar o Evernote.
Tirar retrato	2 - Fotografar e compartilhar a imagem.
Leitura em livro	3 - Leitura em e-readers (kindle)
Escutar músicas de livros	4 – Escutar música pelo Spotify
Vídeo em VHS	5 – Ver vídeo no Youtube/Netflix
Leitura de mapas em papel	6 – Usar Waze.

Repare que as atividades de 1 a 6 utilizam softwares que permitem o compartilhamento de dados usando APIs, não necessariamente todas públicas. Desses softwares pode-se encontrar APIs públicas para o uso dos mesmos de forma programática (ex; APIs do Facebook no caso 2), ou alguns deles podem estar usando APIs de outros serviços, para que seja possível concluir os seus próprios. Os desenvolvedores podem ter utilizado APIs para a produção desses sistemas e agora os usuários finais estarão tirando proveito de tais APIs, mesmo que eles não saibam o que é uma interface de programação de aplicação/aplicativo.

Alguns aspectos em comum podem ser percebidos nessas aplicações:

- uso de nuvem. (não se sabe onde estão os softwares, em qual *datacenter*)
- obtenção de informações sociais – cadastro de usuários. (cria oportunidades ilimitadas)

Com esses exemplos acima, fica mais evidente o grande poder que as APIs têm em, principalmente, agilizar a criação de boas ideias em software, com reutilização de trabalhos já consolidados por várias empresas. E se as empresas deixarem cada vez mais APIs expostas ao público, cada vez mais elas irão acelerar o uso de suas tecnologias, sendo que isso irá acelerar a criação de aplicações inovadoras, o que fomentará cada vez mais rápido o mercado de TI, gerando mais e mais economia.

Mas, API por si só não tem o maior dos valores. Na verdade, a grande riqueza atual que as APIs podem trazer está na exposição não só de serviços computacionais, mas de dados abertos (Open Data). Ou seja, uma API pode permitir desencadear serviço computacional, mas pode também permitir acesso às informações úteis `a população em geral. Por exemplo, dados tornados abertos sobre as cidades, sobre a política, sobre a economia, sobre o relevo e sobre o clima, geralmente de domínio dos governos outrora, tornam-se fontes de dados valiosos e que fomentam a criação de soluções inovadoras para facilitar o cotidiano, tais como soluções inteligentes para facilitar o transporte, a agricultura, a escolha de novos governantes, etc. É com essa possibilidade de criação de soluções relevantes às populações que veio o termo “inteligentes” de “cidades inteligentes”.

A cidade de Los Angeles (sua prefeitura), por exemplo, lançou um serviço em web site, chamado Street Wize, que mostra qualquer problema presente em qualquer uma das ruas da cidade, para evitar transtornos no trânsito. O site deles pode ser visto aqui [12]. O conteúdo do Street Wize é baseado em Open Data – dados acessíveis através de API. Tal conteúdo é provido por uma empresa/órgão chamado Ersi e o site respectivo pode ser visto aqui [13]. O Ersi tem experiência desde 1969 em produção de mapas e trabalhos relacionados com sistemas de informações geográficas. Segundo o web site deles, o Ersi é a instituição mais importante do mundo em produção de mapas. O Ersi também tem API para acesso a seus mapas e disponibiliza um SDK para desenvolvedores em Xamarin. Ou seja, o desenvolvedor que está acostumado a trabalhar no Xamarin e deseja fazer uma aplicação para celular que irá usar mapas, pode usar tal SDK (nome: ArcGIS) e usufruir da API do Ersi e, por exemplo, importar mapas à aplicação. O Xamarin é uma ferramenta de programação, onde o desenvolvedor escreve em C# e o código gerado pode resultar em artefatos nativos para IOs, Android e Windows. Ou seja, escreve-se código apenas uma vez para essas 3 plataformas. A ferramenta mais atual para isso se chama Xamarin Studio e pode ser vista aqui [14]. **Se a Xamarin Studio for tão boa quanto o próprio site clama, então vale a pena um estudo aprofundado sobre a mesma e uma apresentação oral poderia ser feita sobre ela, nessa disciplina.**

O Ersi fornece vários tipos de mapas, através de APIs. A documentação da API pode ser vista em [15]. Nessa documentação, há as explicações de como usar os variados tipos de mapas e como importar algum deles numa aplicação para dispositivo móvel, por exemplo. Os mapas foram feitos de tal forma que, através da API, pode-se colocá-los sobrepostos no aplicativo, como um conjunto de camadas. Por exemplo, pode-se usar um mapa de alta resolução para mostrar uma região. E se for necessário mostrar os nomes das ruas num local, aí é só desenhar sobre o mapa outro mapa especializado em ruas e estradas, mas de resolução menor. Dentre os tipos de mapas disponíveis, podem ser encontrados mapas sobre:

World Imagery

“This map service presents satellite imagery for the world and high-resolution imagery for the United States and other areas around the world.”

REST URL for ArcGIS Web APIs:

http://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer

SOAP API URL: http://services.arcgisonline.com/ArcGIS/services/World_Imagery/MapServer?

wsdl

World Street Map

“This map service presents highway-level data for the world and street-level data for North America, Europe, Africa, parts of the Middle East, Asia, and more.”

REST URL for ArcGIS Web APIs:

http://server.arcgisonline.com/ArcGIS/rest/services/World_Street_Map/MapServer

SOAP API URL: http://services.arcgisonline.com/ArcGIS/services/World_Street_Map/MapServer?wsdl

World Topographic Map

“This world topographic map includes boundaries, cities, water features, physiographic features, parks, landmarks, transportation, and buildings.”

REST URL for ArcGIS Web APIs:

http://server.arcgisonline.com/ArcGIS/rest/services/World_Topo_Map/MapServer

SOAP API URL: http://services.arcgisonline.com/ArcGIS/services/World_Topo_Map/MapServer?wsdl

World Transportation

“This reference map service provides a transportation and street name labels reference overlay that is particularly useful on top of imagery.”

Se esse mapa for colocado sobre o mapa de *imagery*, pode-se ver a imagem de satélite de um local, mais os nomes de ruas, estradas, o sentido correto do trânsito em algumas ruas e mais a numeração dos endereços das residências em cada rua. Ou seja, fica possível aferir onde é a localização (com precisão de quarteirão) de um imóvel com número X, por exemplo.

REST URL for ArcGIS Web APIs:

http://server.arcgisonline.com/ArcGIS/rest/services/Reference/World_Transportation/MapServer

SOAP API URL:

http://services.arcgisonline.com/ArcGIS/services/Reference/World_Transportation/MapServer?wsdl

Na página [16] está disponível a documentação para uso de SDKs do ArcGIS, para várias plataformas, como o Android. Por exemplo, na página [17] há um tipo de “hello word” de sistema que importa mapa do Ersi para uma aplicação Android. Esse pode ser um ponto inicial para o entendimento sobre o uso dos mapas do Ersi. Uma pesquisa sobre o Ersi, sobre seus mapas disponíveis, sobre a utilidade dos mesmos, sobre como importá-los em aplicações móveis e que tipos de vantagens existem em usar tais mapas, seria um ótimo trabalho para essa disciplina e uma

apresentação oral sobre esses assuntos seria viável, sob o aspecto de uso de APIs. Até mesmo uma apresentação oral sobre alguma solução em software já feita em qualquer lugar do mundo, a qual se beneficiou dos mapas do Ersi, é adequada para mostrar o uso de APIs nesse caso e as vantagens de tal tecnologia.

A possibilidade de trabalhar com camadas de mapas libera o desenvolvedor para apresentar em seu mapa apenas as informação que é útil para o contexto da sua aplicação.

A abertura de dados através de APIs vem ganhando cada vez mais espaço na sociedade. Inicialmente algumas cidades tentaram criar APIs durante *hackathons*, segundo [18], o que não surtiu muito efeito. Mas, atualmente estão ocorrendo atitudes bem sucedidas para tornar dados das cidades disponíveis publicamente. E isso tem sido impulsionado principalmente pela demanda de tornar os dados dos governos mais transparentes. A ideia de *open data* através de APIs vem se espalhando pelo setor público e o maior impacto vem sendo localmente, quando os cidadãos interagem com o governo mais frequentemente.

Com o acesso às informações das cidades e dos governos, os cidadãos estão ficando empoderados a tal ponto de tomarem melhores decisões de mobilidade. E isso pode até ajudar a diminuir a corrupção. Segundo [18] “Uma das cidades líderes (em 2016) foi Barcelona. A cidade espanhola desenvolveu uma reputação global de publicação de APIs em departamentos governamentais que incluem dados de trânsito, meio ambiente, território e negócios. A cidade também tem uma plataforma de infraestrutura *open source* que usa APIs para acessar dados de sensores que monitoram temperatura e qualidade do ar, coleta de lixo, fluxo de pedestres etc.”

Com a abertura de dados através de APIs, várias aplicações inovadora surgirão certamente e isso irá aquecer a economia. Conforme [18] “estima-se que o potencial econômico de *open data* é de mais de 3 trilhões de dólares em valor adicionado à economia global.”.

Outro exemplo de API que pode ser apontado aqui é o trabalho da empresa Cnova, visto em [19]. Essa empresa é especializada em criar soluções de comércio eletrônico para outras empresas. Mais, precisamente ela cria *marketplaces*. Por exemplo, as empresas Extra, Ponto Frio e Casas Bahia têm seus respectivos sites para vendas de produtos. Mas, esses sites não precisam vender somente mercadoria do *core business* de cada empresa. Na verdade, esses sites podem ser incrementados com listas de produtos que nem mesmo são dessas empresas. Ou seja, empresas terceiras podem incluir no domínio do Extra, Ponto Frio e Casas Bahia seus próprios produtos, divulgando-os nos sites destas empresas que contêm *marketplaces* na Internet. Esse trabalho incremental é feito via software utilizando APIs que permitem tal adição de outros produtos nesses sites. E essas APIs são criadas pela Cnova. Assim, se o Extra é cliente da Cnova e tem um *marketplace* para receber novos produtos à venda, então ele poderá contar com a venda de mercadoria de outras empresas e lucrar mais ainda. Portanto, nesse caso, as APIs são usadas para divulgar ainda mais o negócio comercial de uma empresa e consequentemente aquecer os negócios da mesma. O lançamento do *marketplace* para o Ponto Frio e Casas Bahia no Brasil foi feito em 2015. Em 2012 foi a vez do Extra e do Barateiro.

Alguns dos web site feitos pela Cnova estão na figura 29:



Figura 29

Ao acessar esses sites e procurar pela palavra *marketplace*, tem-se acesso ao link para o primeiro passo na direção de publicar produtos nessas empresas.

Exemplos de marcas que vendem dentro do *marketplace* do Extra:

Marisa, Polishop, Centauro, Ri Happy.,

Já a marca Sony usa o *marketplace* do Ponto Frio, por exemplo.

Segundo o site 99 APIs, visto em [20], “A API do *Marketplace* da Cnova visa facilitar a integração entre lojistas e plataformas ao *Marketplace* da Cnova, possibilitando a venda de produtos no Extra.com.br, CasasBahia.com.br, Pontofrio.com.br e Cdiscount.com.br”. Ainda vendo o 99 APIs, a API do *marketplace* do Cnova pode ser encontrada no link [21]. Para interagir com a API do Cnova, pode-se programar com o SDK disponibilizado por ela. Há SDKs para as tecnologias mostradas na figura 30. Não necessariamente o uso de uma API tem que ser a partir de uma aplicação para *mobile*.



Figura 30

Através da API da Cnova é possível cadastrar produtos, gerenciar os estoques e preços e ser notificado quando pedidos são feitos e confirmados. Todos os recursos disponibilizados pela API são baseados em *Webservices Restful* e as mensagens trafegadas no padrão JSON. Um aluno dessa disciplina poderia acessar o link em [21] e estudar como usar a API da Cnova. As informações adquiridas e entendidas seriam apresentadas oralmente, com a ajuda de slides, durante a aula, o que seria muito conveniente para mostrar o uso de um exemplo de API e mostrar vantagens de se fazer tal uso. Tal conhecimento a ser exposto pode ser de grande valor, por exemplo, às empresas incubadas.

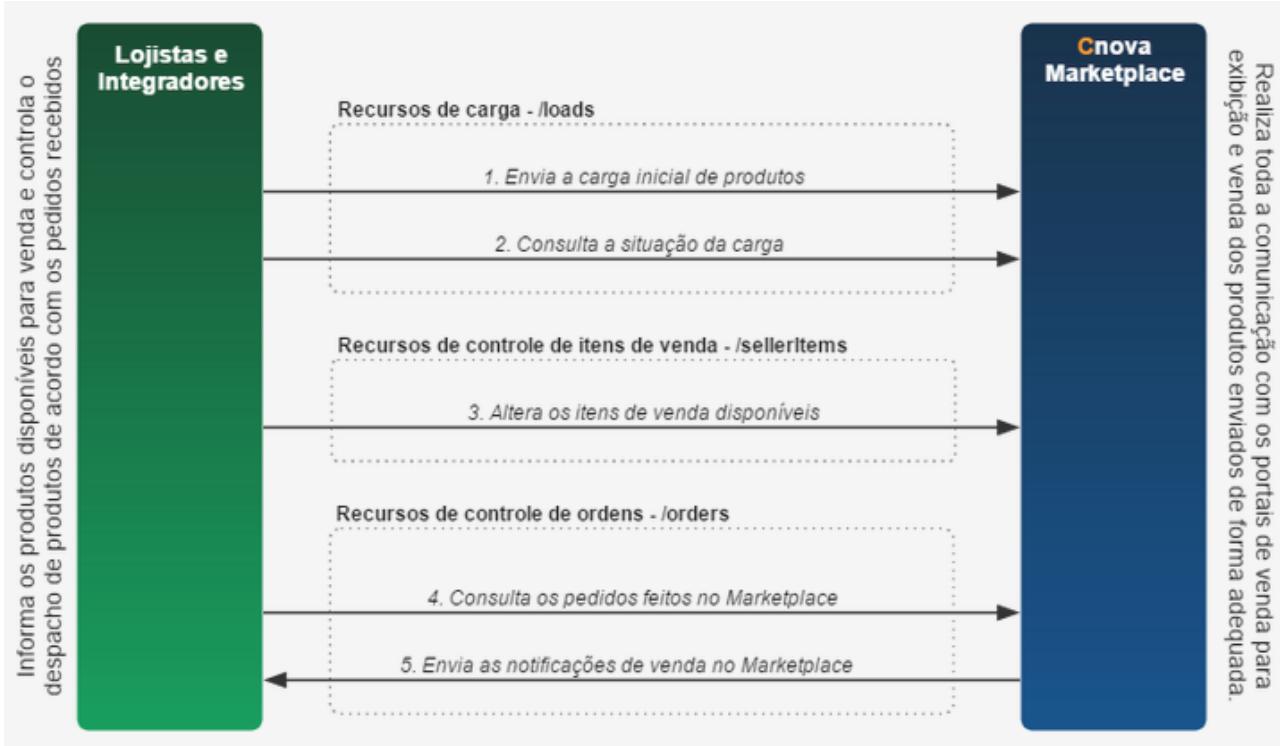


Figura 31

A figura 31 mostra, numa visão bem macro, como um logista pode interagir com o *marketplace* da Cnova através de requisições enviadas à API respectiva. É interessante perceber também que a API da Cnova está muito bem documentada no site [21] e é fácil navegar pela documentação, assim como tem que ser uma API que deve ser realmente aceita e se tornar popular.

A Netflix também é um exemplo de empresa que faz uso de API. Inicialmente, quando a Netflix lançou o seu serviço de *streaming* de vídeo, o desafio era colocar aplicativos da Netflix em vários dispositivos e plataformas diferentes. Ou seja, dispositivos com IOs, Android, Windows, sistemas operacionais para TV, etc. Como avançar em tantos dispositivos? A solução encontrada foi expor as capacidades dos serviços da Netflix, no intuito de aguardar profissionais do mundo de TI produzirem aplicações com a API do Netflix. Hoje mais de 800 dispositivos diferentes rodam aplicativos Netflix, mas 90% deles foram produzidos pela própria Netflix. A própria Netflix consome a sua API mais que outros *players* do mercado. Isso mostra que não é só devido ao fato de expor a computação de uma empresa via API, que haverá um grande sucesso de novos serviços surgindo usando tal API.

Algumas APIs legais

O site [20] tem uma seleção de APIs brasileiras e outras internacionais. Nele, pode-se ver uma lista de APIs mais populares. Esse é um site muito útil, porque ele tem um breve resumo da utilidade de cada API e um link direto para a mesma. O link direto para uma API dá acesso a tudo o que o desenvolvedor precisa para trabalhar com a mesma. Ex: documentação e SDKs. Vale a pena acessar o site [20] agora e dar uma olhada no grande conjunto de APIs registradas nele! Elas também estão

organizadas por categorias. Ex: sociais.

Geralmente as APIs estão associadas às empresas e é fundamental conhecer os produtos ou serviços das mesmas empresas, para que o objetivo das APIs respectivas seja entendido mais facilmente. Por exemplo, o Facebook é uma empresa bem conhecida e o negócio dela já está entendido por muitas pessoas. Para quem já conhece o Facebook e sabe o que ele faz, fica mais fácil entender suas APIs. Por exemplo, a API para login (com senha) usando funções do Facebook, mesmo para aplicações que não são dessa empresa, é fácil de entender, em termos de utilidade. Outro exemplo é o Amazon Web Services. Para quem já está intencionado a usar capacidades do AWS e sabe um pouco sobre computação em nuvem, não haverá dificuldade em entender a utilidade de certas APIs do Amazon. E o AWS realmente tem APIs para toda a sua capacidade instalada em nuvem. Um exemplo de uso dessas APIs foi visto no capítulo I, quando a API do SNS foi utilizada no código da função Lambda.

A primeira API legal da lista montada aqui é a do **Google Classroom**. Pelas salas de aula, o modelo de ensino mais antigo e conhecido é aquele com um professor falando para uma plateia de alunos, todos na mesma sala. Ou seja, um orador e o resto como ouvinte. Há pouco tempo em sala para debates. Será que essa é a forma mais eficiente atualmente de montar salas de aula? Segundo o web site da Sensedia, visto em [22], com o Google Classroom “...os professores podem criar salas, convidar alunos, enviar tarefas e corrigir, tudo sem sair do ecossistema Google e com integrações bastante suaves. Tudo isso empacotado em uma bela interface Material Design” Vale a pena conferir o vídeo sobre a ferramenta (2 min), disponível no link [23]. A ideia do Google Classroom é simples. Através do web site respectivo, um professor pode se registrar e criar uma sala de aula. Ele também pode convidar alunos a participarem da sala, como num grupo de discussões na Internet. Cada aluno também pode se inscrever na sala, por conta própria, desde que conheça o código correto da sala. É como usar uma chave para passar pela porta de uma sala. Com a sala montada e os alunos participando, o professor pode enviar pedidos de exercícios a serem feitos fora da classe. Os alunos também podem enviar as soluções dos exercícios para essa sala criada e o professor passa então avê-los e pontuá-los. No final das contas, o Google Classroom é um organizador de documentos e tarefas entre professores e alunos. Mas, existe a API que permite, dentre outras coisas, colocar um botão numa aplicação web (*Classroom share button*) para o envio de documento a uma sala de aula. Ou seja, um desenvolvedor pode criar seu próprio aplicativo de salas de aula, usando as capacidades do Google Classroom, mas com a GUI que desejar ou a que seja mais conveniente segundo hábitos da população a ser atingida. A API do Google Classroom está disponível no web site [24] e há SDKs para várias linguagens, dentre elas: Android, IOs Java, Javascript, PHP, Python, Ruby, etc. Uma pesquisa sobre como usar essa API, para uma dessas linguagens, poderia ser feita e um protótipo poderia ser implementado e demonstrando oralmente, como uma apresentação em sala de aula. Segundo o Google (em 2016) 10 milhões de estudantes e professores estavam usando o Google Classroom.

Outro serviço disponível na web, com o mesmo propósito do Google Classroom, mas muito mais poderoso e complexo é o **Moodle**. Essa é a maior plataforma mundial de gerenciamento de aprendizagens (aulas, professores, alunos e pais de alunos), de código aberto. Os educadores (Ex: professores) podem criar seus próprios websites preenchidos com seus cursos, a qualquer momento, de qualquer lugar. Com o Moodle, alguém poderia criar um website para organizar todos os cursos

do Inatel, por exemplo, porque isso seria uma nova organização entre professores, alunos e pais de alunos. O Moodle pode ser baixado para um servidor web do Inatel por exemplo, onde todas as suas páginas e dados ficariam hospedados. Isso seria um excelente trabalho de conclusão de curso para essa pós-graduação, que é na área de computação antes de mais nada. Como o Moodle é gratuito, existe uma grande comunidade de usuários e desenvolvedores por trás dele. Vale a pena ver o vídeo que se encontra no link [25]. Nele, os professores podem criar cursos, questionários, solicitar atividades por escrito, dar notas, etc. Os alunos podem aderir a cursos de professores, podem acessar o material de cursos, ver notas, engajar em trabalhos em grupo, etc. A *homepage* do website Moddle fica no link [26]. O Moodle oferece uma API que facilita por exemplo a criação de *plugins* para suas capacidades. O site do Moodle oferece um *demo* que pode ser experimentado a qualquer momento.

No campo da música também existem muitas APIs interessantes. Por exemplo, o site Spotify oferece API para seu conjunto de funções, vistas aqui [35]. Atualmente, as pessoas estão cada vez mais gostando das possibilidades de montar listas de músicas, de comprar músicas pela Internet, de compartilhar listas de músicas, de vender e divulgar trabalhos musicais na rede, etc. Tudo isso é possível através de sites do gênero musical, como o Spotify. E há muitos outros. Sendo que muitos apresentam APIs para facilitar o desenvolvimento de novas aplicações criativas nesse ramo da música, usando as capacidades dos serviços das empresas respectivas desses sites. Um desses sites é o **Discogs**. Ele tem uma API, vista no link [27] que permite fazer a busca por discos, até mesmo de vinil, permitindo encontrar álbuns bem antigos através de filtros de data, país, gênero musical, etc. É possível também publicar discos antigos para serem vendidos no *marketplace* da Discogs. Já o **Spotify** é um dos serviços de música sob demanda mais populares atualmente, segundo o website 99Apis, contendo um número de APIs diferentes para vários tipos de serviços oferecidos por essa empresa. Segue abaixo alguns exemplos de aplicativos para Android, *web* e IOs já construídos com APIs do Spotify:

- *Music Tonight* : aplicativo que apresenta uma lista de artistas que estarão tocando hoje à noite em algum lugar perto de você. Assim fica mais fácil saber previamente quais as opções agradarão nessa noite e planejar melhor onde ir. (Provavelmente é útil em cidades grandes como São Paulo, NY, Paris, Stockholm, Helsinki, Bruxelas, Tóquio, etc.). Aqui o aplicativo analisa quais são os artistas tocando nas proximidades (através da **API SeatGeek**), à noite. Depois monta uma lista com os mesmos artistas já indicando as músicas ou música mais popular de cada um, através da API do Spotify. Assim rapidamente o usuário do aplicativo fica bem informado sobre os shows que irão acontecer na mesma noite em suas proximidades. Segundo o site [28] “*the SeatGeek maintains a comprehensive directory of live events in the United States and Canada (and to a lesser degree, internationally)*”. O código do Music Tonight está no Github.

- *Spotify Artist Explorer*: aplicativo que mostra lista de artistas musicais com relacionamentos profissionais a outros artistas também musicais. Por exemplo, escolhe-se um artista no aplicativo e ele imediatamente mostra outros artistas relacionados, talvez por gênero musical, por participação em comum em bandas, etc. Daí em diante o aplicativo toca 30 segundos de uma música popular do artista escolhido da lista de artistas relacionados encontrados. A API do Spotify tem capacidade e acesso à informações suficientes para prover o que o *Spotify Artist Explorer* precisa. O código fonte

desse aplicativo também está disponível no Github. Mais informações sobre esse aplicativo fantástico podem ser vistas no link [29]. E um video muito bem explicado sobre isso está no link [30].

- *Klarafy*. Esse software ajudará as pessoas a gostarem de música clássica. Como funciona: cite o nome de uma música que você gosta. Com a ajuda de API do Spotify, o software encontrará músicas clássicas que mais lhe agradarão. Por exemplo, se for citada uma música de “metal pesado”, o sistema encontrará uma música clássica que é melhor escutar mais alta, com batidas fortes e não sugerirá uma de piano, por exemplo. Conforme a *webpage* [31] que comenta sobre o Klarafy, esse sistema “*...works on the credible premise that someone who likes loud, powerful metal is more likely to enjoy a loud and powerful piece by Wagner than soft and delicate piano music. So Klarafy seeks out the affinities, similarities or links between your favorite music and classical music. In order to perform the ‘translation’, Klarafy scans your Spotify playlist for three criteria: your favorite music genre, the prevailing musical mood and finally specific points of departure such as particular instruments, voice types etc.*” Na página [31] há um video sobre o Klarafy.

No ramo da música, com o Spotify, existem muitas aplicações interessantes e seria impossível apresentar todas nessa disciplina, mas um aluno poderia fazer uma pesquisa mais aprofundada no assunto e mostrar mais aplicações e suas utilidades, enfatizando o uso das APIs do Spotify, mostrando quais e como foram usadas. Por fim, uma apresentação sobre o assunto poderia ser feita, mostrando também algum exemplo de código fonte cliente de alguma API do Spotify.

- *Roadtrip Mixtape*. Essa é mais uma legal. Cite no aplicativo uma rota a ser viajada e o software irá montar uma lista de artistas que pertencem às regiões por onde a rota atravessa. Simples ideia (não tão simples para implementar), mas muito legal! Assim, é possível viajar escutando artistas das regiões por onde se passa. Muito curioso esse sistema, que possibilita um turista conhecer o estilo de música popular por onde estiver andando, por exemplo no exterior. A ideia desse software é a seguinte: foi usado o banco de dados de localização de artistas do The Echo Nest [32]. Esse banco de dados dava a localização de qualquer artista. Veja aqui como podia ser uma chamada à API do The Echo Nest, para recuperar a localização de artista para o RadioHead:

"http://developer.echonest.com/api/v4/artist/profile?api_key=N6E4NIOVYMTNDM8J&name=radiohead&format=json&bucket=artist_location".

(Atualmente as funções da API do The Echo Nest foram passadas para a API do Spotify, o que ocorreu em 2016.) Com isso, foi coletado a localização dos 100000 artistas mais famosos do catálogo do Spotify, dos EUA. Esses artistas são de 15000 cidades diferentes. A latitude e longitude para cada uma dessas 15000 cidades foi então determinada com o *Yahoo Placefinder geocoder* [33]. Para uso e apresentação de mapa, foi usada a versão 3.9 do Google maps API. Para a apresentação de 30 segundos de demonstração de músicas de artistas foi usado a API Spotify Web. Então o sistema vai determinando em qual cidade está o viajante. Com os dados previamente montados, para cada cidade define-se a lista de artistas. E então é só tocar músicas de 5 artistas a cada 15 minutos. A cada 15 minutos o sistema analisa novamente em que cidade o viajante está e monta nova lista de mais 5 artistas populares. O código desse software está disponível no Github. Os assuntos Google maps API e Yahoo Placefinder geocoder API são ótimos para uma apresentação oral nessa disciplina, explicando as vantagens da utilização dessas APIs. Um aluno poderia mostrar

código cliente usando essas APIs ou exemplos de aplicações no mercado que as utilizam. Vale a pena ver a demonstração no site [34].

Algumas APIs direcionadas para desenvolvimento ou desenvolvedores de software não poderiam ficar de fora desse documento, obviamente. Considere as APIs do Github e do Linkedin então. Explicações sobre essas APIs poderiam ser dadas pelos alunos, em apresentações orais, em sala de aula. A API do Github está nessa web page: [36].

No mundo dos utilitários e software como serviço (SaaS), podem ser citadas as seguintes APIs, dentre outras:

- *Twilio* : A API do Twilio permite que desenvolvedores realizem e recebam chamadas telefônicas e SMS. O Twilio não tem qualquer prestação de serviço que não através de sua API. Ou seja, ele foi construído para ser puramente uma API. É portanto uma API da área de comunicação ou telecomunicações. Por exemplo, se alguém quer criar um software como o skype, poderia ser usada essa API. O Twilio é uma API de comunicação suportada em nuvem. A API do Twilio parece ser muito rica em possibilidades de comunicações e uma apresentação oral sobre a mesma, mostrando detalhes, como usá-la e o que pode se feito com ela seria de grande valia para essa disciplina, o que poderia ser feito por um aluno, por exemplo. O aluno poderia mostrar se essa API usa ou não usa SIP, que é um dos protocolos mais usados para o estabelecimento de sessões de *media* entre *peers*. Se não usa SIP, então como a sessão é estabelecida? A resposta deixará a apresentação muito interessante! De alguma forma o Twilio usa o SIP, conforme figura 32, mais adiante nesse documento, que foi obtida no próprio site do Twilio. Essa arquitetura poderia ser explicada na apresentação oral, por exemplo.

- *Dropbox*.

- *Evernote*.

- *Salesforce*. Essa empresa é uma das líderes em soluções de CRM (gerenciamento de relacionamento com clientes). O sistema dela, todo em nuvem, permite as empresas organizarem muito melhor os dados de seus clientes, bem como processos de vendas ou negociações em andamento. Com Salesforce um vendedor trabalha melhor e consegue estar mais próximo do cliente, não perdendo o *timing* de negociações iniciadas no passado. A solução da Salesforce é SaaS. Exemplo na prática: "...Quando você está vendendo um serviço ou produto, dependendo do que seja não é uma coisa imediata e demanda tempo e acompanhamento até que seja o momento certo. Por exemplo a venda de um serviço de limpeza de condomínios demanda que o contrato vigente esteja no final e que eles estejam de fato procurando uma nova empresa de limpeza. Para você não perder esse *timing* você usa uma ferramenta de CRM para te ajudar a lembrar o que foi dito e conversado entre quais contatos da empresa e quando seria ideal voltar o assunto. Dai por diante." Esse exemplo foi dado por Raphael C, no chat com o atendimento da Salesforce, em 19/09/2016. (contato no Salesforce: rciabattari@salesforce.com - Raphael C). A solução é típica para grandes empresas. Por exemplo a Embraer. Para o aluno que já tem noção de CRM, pode ser uma boa opção de pesquisa a análise da API do Salesforce, para depois comentá-la oralmente durante apresentação nessa disciplina. A pesquisa, se for feita, pode iniciar na página [37].

Outras APIs que não podem ficar sem comentários são aquelas para IoT e/ou computação em nuvem. No capítulo anterior foi visto um caso de uso de serviços do AWS IoT. E muitos daqueles serviços podem ser acessados/usados através de APIs (acesso direto aos serviços) sem passar por ações feitas diretamente no console do AWS IoT. Na verdade, quase tudo no AWS conta com APIs “públicas”. Por exemplo, o serviço de SNS usado na função Lambda foi possível através da API. A API desse serviço está documentada na página [38]. Outro serviço do AWS que tem API é o SQS (serviço de fila simples). Já na página [39] está disponível a API do SQS.

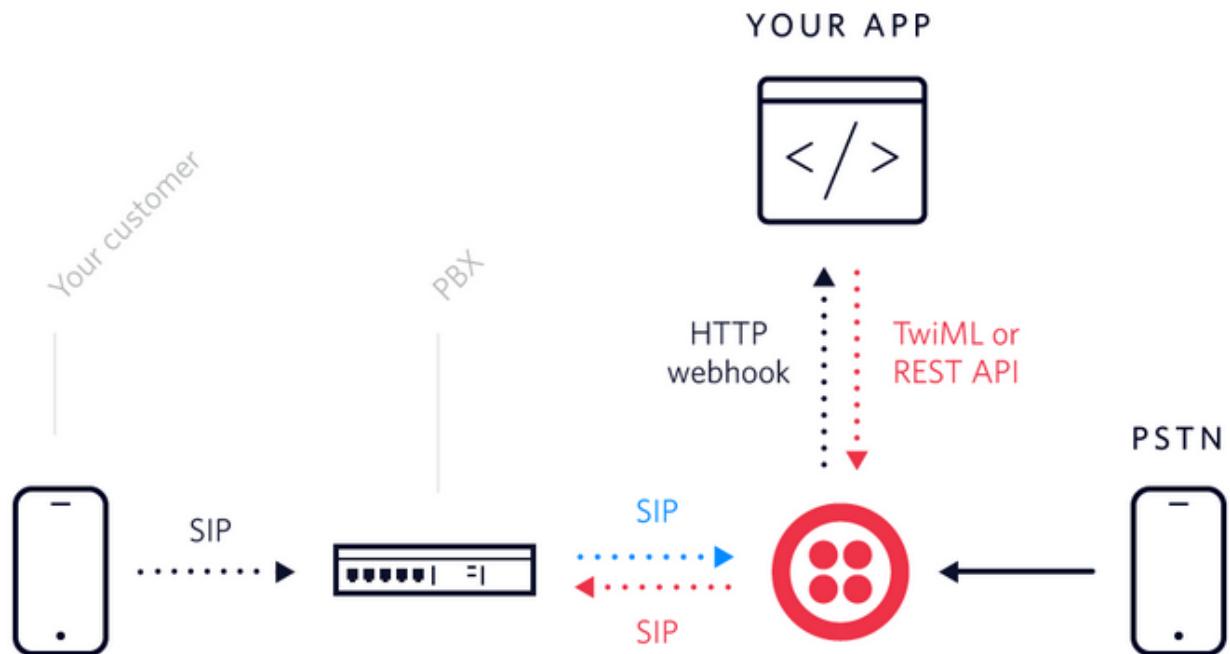


Figura 32

Finalmente, tudo o que foi visto via console, no caso de uso do capítulo I, para AWS IoT, poderia ter sido feito via código fonte, se a API do AWS IoT tivesse sido usada. Para saber sobre essa API, basta acessar a página [40]. **A gama de API do AWS é realmente grande! Um aluno dessa disciplina poderia escolher qualquer uma das APIs do AWS e comentar sobre ela, em forma de apresentação.**

Na indústria automobilística a Ford foi uma das primeiras a criar um programa para desenvolvedores de software, para que eles criem aplicativos que se conectem com os carros. Para tal, a Ford tem a sua API pública para desenvolvimento dos aplicativos, que pode ser vista nessa página [41]. Em 9 de setembro de 2016 a Ford patrocinou um tipo de “hackathon” (maratona de programação) com o intuito de ver surgir aplicações de sua API, com propostas de software para facilitar a vida das pessoas relacionado com os automóveis dessa montadora. O nome desse evento foi “Ford SYNC AppLink Mobility Challenge” e durou 24 horas. Foram 35 times e 75 desenvolvedores de 10 países. Os vencedores levaram 20 mil euros, dentre outros prêmios. Foram 8 trabalhos premiados. De acordo com a Ford ela “... provides software and technology to the app developer community through the [SYNC AppLink Developer Program](#) – with more than 15,000

registered users. The free [SYNC Emulator software](#) enables developers to create and test apps for compatibility with the automaker's connectivity system – without needing to access an actual vehicle.” Um estudo detalhado sobre todas as oportunidades que a Ford oferece a desenvolvedores de software, como os alunos do Inatel, considerando os eventos patrocinados, os prêmios oferecidos, as ideias já construídas com suas APIs e tudo mais o que vem ocorrendo e pode se tornar campo de trabalho para os profissionais da Computação, seria de enorme valia ao ser apresentado oralmente nessa disciplina por um aluno, por exemplo. A justificativa para apostar em tal estudo é o fato de que a Ford está atualmente expandindo em duas áreas: automobilística e também como uma empresa de mobilidade, ou seja, com plano de se tornar líder em conectividade, mobilidade, veículos autônomos, experiência de usuário e em dados e análises. Pelo visto, a atuação da empresa não será mais somente montar carros, mas sim prover serviços com capacidades de melhorar a vida das pessoas em termos de mobilidade, o que certamente vai demandar projetos em telecomunicações. Os 8 trabalhos premiados foram:

- Auto: sistema para pagamento automático de estacionamento, sem o motorista se preocupar com a ação de pagar, com valores de taxas.
- BeeRides: permite o aluguel de carros estacionados em aeroportos, para evitar que os mesmos paguem a taxa de estacionamento e ainda proverem retorno financeiro aos donos que podem estar viajando. Os clientes em potencial são os viajantes que chegam ao aeroporto, obviamente.
- Ellis Car: analisa dados de estradas e motoristas, forma um conjunto de sugestões com esses dados analisados e passa dicas aos motoristas para eles dirigirem de forma mais econômica em termos de consumo de combustível.
- Hal : permite criar conferências com a telecomunicação sendo feita a partir do carro. Ex: conversar enquanto dirige.
- Make My Day : (não entendi esse).
- POMP: auxilia a entrega de combustível nos carros, sem que o motorista tenha então que ir até o posto de combustível.
- Smart HitchHike: aplicativo para associar carros em trânsito a pedintes de carona. Mostra para os pedintes onde estão os carros em potencial para oferecer carona.
- Truckfly: ajuda empresas localizarem caminhões vazios mais próximos. Ajuda então os caminhoneiros encontrarem seus próximos pontos de paradas e planejar o dia de trabalho.

Com as APIs e sistemas da Ford, fica possível, por exemplo, interagir com o sistema do carro via voz e ele interagirá com o seu dispositivo móvel. Então, por exemplo, através da interface do automóvel, pode-se pedir a execução de aplicativos já presentes no seu celular. Isso livra o motorista para manter as suas mãos no volante e os olhos na estrada. O que a Ford quer é a possibilidade do usuário continuar usando dentro de seu carro os aplicativos que ele já está acostumado a usar fora do veículo, mas sem diminuir a segurança no momento em que o carro está sendo guiado. Além disso, também é possível criar novas aplicações para serem instaladas nos celulares e executadas com a ajuda da interface do carro. Ou seja, os aplicativos continuam rodando

no celular, mas o carro irá permitir interagir com os mesmos, sem a necessidade de usar as mãos ou os olhos. Dois bons vídeos para explicar esta ideia são [42] e [43]. E mais informações podem ser vistas na página [44]. A comunicação entre dispositivo móvel e o carro usa Bluetooth.

Mudando para o assunto de casas conectadas, a empresa NEST “...tem produtos para manter sua casa conectada e segura. Com uma política de APIs públicas eles pretendem engajar desenvolvedores em torno do seus dispositivos, criando novas soluções para casas inteligentes”, segundo [45]. Mas, a empresa Samsung tem algo muito parecido. Essas empresas criaram sensores que podem ser esparramados numa casa e todos eles se comunicarão com um *hub* que os acompanha no mesmo pacote de produto vendido. Com o *hub*, pode-se obter informações dos sensores através de um *smartphone*, por exemplo. O vídeo visto em [47] mostra o que pode ser feito com o kit da Samsung para tornar uma casa inteligente. Repare que esse vídeo da Samsung mostra uma solução que pode até ter sido feita usando exatamente as tecnologias vistas no capítulo I dessa apostila. Recentemente (2016) a BMW incorporou em alguns modelos de carros a possibilidade de se comunicar com esses sensores da Samsung para, por exemplo, deixar o motorista verificar os status de uma residência, enquanto dirige pelas estradas. Para trabalhar com APIs da Samsung, pode-se acessar a página [48].

Pensando em viagens, turismo e Internet, vem em mente a imagem de *web sites* famoso para a reserva de hotéis, não é mesmo? Ainda mais que hoje em dia há muita propaganda na televisão sobre esses serviços na web. Ex: Trivago, Decolar.com, etc. Um dos meus sites preferidos para reserva de hotel é o Booking.com. E para identificar passagens aéreas há o Skyscanner. Mas, caso alguém queira criar o seu próprio site de reservas de hotel, poderá, por exemplo, usar a API pública do TripAdvisor. Na verdade, atualmente é possível criar uma agência de viagens completamente baseada em APIs do ramo. Ou seja, uma agência no celular, com as capacidades de reserva de hotel, reserva de passagem aérea, etc.

A evolução das agências de turismo percorreu um caminho através das seguintes tecnologias/ferramentas:

We site de companhias aéreas e de hotéis → *Web services* das empresas aéreas e de hotelaria → APIs públicas.

Segundo [49] “Como então a maioria das plataformas conseguem os horários de todos os voos consultando sempre o melhor para o seu consumidor final? Hoje em dia temos diversas empresas que fazem este trabalho consumindo as APIs das empresas aéreas e disponibilizam através de sua própria as informações para dono do negócio, um exemplo é a empresa **Amadeus**. Eles possuem um serviço que **qualquer empresa pode contratar** e assim possuir todos os dados de voos para disponibilizar para o seu cliente”.

A British Airways, por exemplo oferece um conjunto de APIs gratuitas. Para usá-las é necessário fazer um registro e depois obter uma chave de uso. Isso pode ser visto na página [50]. As APIs são divididas em grupos de tipos específicos. Por exemplo, existe o grupo de APIs sobre Ofertas de Voos. E nesse grupo há a API *BA Destinations*. Essa API retorna uma lista de destinos disponíveis, através da British Airways, organizada por aeroportos. A API ***Flight Offer Basic*** que pertence ao

mesmo grupo, retorna voos mais baratos existentes para rotas específicas, de um período de 12 meses. A API *BA Destinations*, por exemplo, tem um *endPoint* que atende requisições em REST. Um exemplo da sintaxe de uma requisição segue abaixo:

<https://api.ba.com/rest-v1/{resourceVersion}/{resourceName};{filters}>

E um exemplo de filtros está logo abaixo:

`flights;departureLocation=LHR;startTime=06:00;endTime=11:00`

Pense numa aplicação legal com essas APIs. Crie o software e divulgue para a British Airways. Se houver possibilidades de lucro com tal aplicação, a British Airways fará uma proposta comercial vantajosa para ambas as partes.

Seria interessante ver uma apresentação oral sobre empresas relacionadas com o turismo e suas APIs públicas. A lista de empresas pode contar com os nomes : Expedia, Booking.com, TripAdvisor e Hotel Urbano. Cada conjunto de APIs dessas empresas têm capacidades específicas que ajudaram a garantir uma fatia do mercado para essas empresas. O estudo poderia focar nas características mais vantajosas de cada API e mostrar um pouco da forma como utilizá-las em software.

Para finalizar esta pequena amostra de lista de APIs legais, uma que desperta o meu interesse é aquela do portal Transparência Brasil. “A Transparência Brasil, uma ONG de combate à corrupção, disponibiliza APIs com as informações que ela coleta sobre políticos para que desenvolvedores criem Apps que ajudem a população não só a votar com maior consciência e com base em dados, mas também a fiscalizar os políticos do congresso nacional”, segundo [51]. No website da Transparência Brasil está escrito: “Com nossas APIs, você pode desenvolver aplicações para ajudar eleitores no exercício de seus votos”. E é exatamente isso o que eu penso que alguns desenvolvedores de software deveriam fazer ou continuar fazendo!! Imagine criar uma aplicação para gerar *push notifications* avisando sempre que um político cometer uma desonestade !!! Ou seja, depois de receber muitas notificações de um mesmo político, certamente seria fácil desistir dele numa eleição futura. Isso seria como um boicote à vida profissional do político desonesto. Mas, muito mais pode ser criado em termos de aplicações no intuito de diminuir a corrupção. Por exemplo, um aplicativo poderia calcular quantos anos você deveria trabalhar, recebendo seu salário atual, para acumular o mesmo valor somado de roubos de algum político desonesto durante poucos anos. E imagine um aplicativo simulando uma urna eletrônica com todos os políticos das próximas eleições já disponível. A população poderia ir simulando em quem iria votar, no aplicativo, mudando de ideia sempre que desejado. Isso seria uma coleta de dados fabulosa, para antever quem teria mais chances de vencer as eleições. Mas, tem-se que verificar se no Brazil esse tipo de estatística pode ser criada por qualquer pessoa.

Esse portal juntamente com a Sensedia fez um *hackathon* com as APIs em agosto de 2014. E segundo o site [52] “Os vencedores foram os criadores do Quem Financia, nome provisório de uma extensão do navegador Chrome que varre a página que o internauta estiver lendo, identifica os nomes de candidatos à Presidência e ao governo e oferece ao usuário a opção de saber quem financia aquele político, além de sua declaração de bens.”

Mas, infelizmente parece que este portal contém dados de políticos somente até fatos de 2014. Faz-

se necessário pesquisar no Brasil que outro portal teria esses tipos de dados, mas atualizados. Uma apresentação com o resultado da pesquisa, bem como exemplos de uso de APIs para esse tipo de trabalho seria muito útil para enriquecer essa disciplina. Talvez uma pesquisa atualizada possa ser iniciada no site <http://dados.gov.br/>.

Formas de ganhar dinheiro com APIs

As APIs podem ser vistas como um assunto puramente técnico, por profissionais de Computação, mas elas também são assunto de discussão entre outros profissionais mais interessados em negócios entre empresas, publicidade, fidelização de parcerias e dinheiro. Portanto, se o assunto não é estritamente técnico, uma discussão sobre APIs pode sim atrair a atenção de profissionais variados.

A forma mais óbvia de se ganhar dinheiro com APIs é a cobrança direta. Ou seja, a API fica disponível para uso, mas quem usa deve pagar por número de transações (requisições e respostas), ou por quantidade de dados trafegados, ou assinatura. É como um aluguel de API. Empresas que usam esse modelo são, de exemplo, Amazon (Amazon Web Services), PayPal (Pagamentos) e SendGrid (envio de e-mails). O serviço do SendGrid podeia ser estudado e apresentado em detalhes oralmente por um aluno nessa disciplina.

Outra forma de ganhar dinheiro com APIs, inclusive já comentada nesse documento, é o Canal de distribuição: programa de afiliados e *marketplaces*. Algumas empresas estão interessadas em aumentar a variedade de produtos ofertados e vendidos. Elas utilizam as APIs para permitir que uma terceira empresa introduzam mais ofertas em seus canais de vendas. Além disso, essas empresas usam APIs para facilitar o trabalho de seus afiliados, que podem assim obter dados importantes de seus bancos de dados. Ex: verificar disponibilidade de produtos, etc. Isso garante mais venda, o que dá mais dinheiro em retorno.

Já outras empresas usam as APIs como *marketing*. Se as APIs são úteis para muitos desenvolvedores que consequentemente farão aplicativos úteis para muitas pessoas, então os nomes de empresas por de traz das APIs ficam cada vez mais populares. Isso pode atrair cada vez mais usuários para os serviços de tais empresas e isso é bom para aquelas que têm como estratégia principal de negócio a atratividade de mais e mais usuários. Exemplos de empresas desse tipo são: Facebook, Twitter e Linkedin. Nesse modelo de negócio as APIs podem ser usadas gratuitamente.

Outra forma de usar as APIs, deixando-as livre ao público, é a criação de um canal facilitador para uso do software de uma empresa. Por exemplo, se uma empresa contém software ou dados úteis a outras empresas clientes, então a empresa detentora do conhecimento pode criar APIs que facilitarão suas clientes criarem soluções em software que poderão usufruir de tal conhecimento. Ou seja, aqui a API garante a fidelidade do cliente, tudo em termos de software. É como um “chiclete” que agarra uma empresa na outra. Um exemplo de empresa que usa essa técnica de fidelização é a Salesforce. A fidelização de mais e mais cliente é obviamente uma forma de conseguir mais retorno financeiro.

Finalmente, as APIs podem trazer lucros não exatamente às empresas que as criaram, mas aquelas

que irão utilizá-las. Crie um negócio ou software que irá utilizar APIs, sobre pela utilização de tal nova solução e então obtenha o dinheiro tirando vantagem de outros serviços computacionais disponíveis via as APIs. Um modelo de negócio ou solução, a ser criado, mesmo que apenas teórico inicialmente, mas utilizando um conjunto de APIs, poderia ser um exemplo a ser mostrado em sala de aula, oralmente por um aluno, por exemplo para mostrar quão importante é a tendência atual em usufruir de APIs. Ou seja, um aluno dessa disciplina está livre para usar a imaginação e propor um modelo de negócio a ser apresentado.

APIs e IoT

Existe uma certa importância das APIs em IoT. Em IoT muitas coisas estarão conectadas na Internet justamente para que, dentre outros motivos, seja possível enviar dados coletados à pontos de interesse. Por exemplo, sensores de tremores de terra podem enviar dados todos os dias à algum servidor na nuvem, para que fiquem registradas as informações de uma região. Mas, o envio das informações pode ser através da utilização de APIs, ou seja, APIs com funções para entrega de dados, ou para execução de algum processo sobre os dados. Então, as APIs vieram para facilitar a comunicação das coisas com softwares. Quanto mais a IoT crescer, mais haverá novas APIs aparecendo no mercado. Quanto mais APIs aparecerem no mercado, mais fácil será lidar com as coisas na Internet interagindo com elas, do ponto de vista computacional. E as APIs podem cuidar da parte de segurança na comunicação, além de serem geralmente simples em termos de interface de serviços.

Conclusões e considerações finais sobre APIs

Como visto, já existe um fenômeno de APIs por aí e quem é da área de Computação irá lidar com elas muito provavelmente, mais cedo ou mais tarde. Mas, nem sempre foi assim. No início, as empresas começaram a modularizar o código de seus projetos de software, para prover o reuso de código, reúso de partes de projetos em código, chamados módulos. Em seguida veio a Orientação por Objetos, com o intuito de organizar o código, mas também de permitir o reaproveitamento de lógicas dentro do mesmo projeto. Com a evolução das técnicas de computação, surgiram os *web servers* provendo uma forma de publicação de informações através dos web sites. Mais adiante, surgiu SOA, dessa vez com o foco principal em reutilização de serviços, mas usando técnicas (princípios de *design*) e teorias para padronizar isso e garantir o retorno financeiro com a aplicação delas. Daí, o mercado já havia experimentado organização de código, reaproveitamento de código e publicação de informações via Internet. Mais adiante, surgiram então as APIs, com foco principal em criar interfaces entre sistemas de software, para facilitar ainda mais o reúso de programas/funções em programas de computador. Inicialmente as APIs eram usadas somente internamente dentro da mesma empresa, facilitando retorno econômico provido pela SOA. Esse retorno significa economizar com o pessoal de TI. Menos trabalho a ser feito, por exemplo. Finalmente e mais recentemente, as APIs despontaram como a grande aceleradora de projetos inovadores e a cola entre as empresas, facilitando muito o desenvolvimento de software que

reutiliza o que já existe pronto em TI em vários fornecedores. Atualmente o alvo principal das APIs é o mercado de pessoa física, pelo tamanho do mercado. E essas APIs atuais ainda estão suportadas em grande parte por SOA e mais recentemente por *microservices*.



Figura 33. Fonte: [11]

Atualmente existem muitas APIs no mercado de Computação e para uma visão geral sobre elas (saber quais existem) pode-se procurar por *websites* que trazem listas com as APIs. Por exemplo, boas fontes de explicações sobre APIs são os *websites* da Sensedia, MundoAPI, 99APIs e ProgrammableWeb. Na verdade, esse capítulo nem “arranhou de leve” o assunto de APIs, por ter sido muito superficial em detalhes. De fato, esse capítulo é apenas para dar um primeiro contato com APIs. O conteúdo daqueles *sites* é muito vasto, sendo suficiente para uma disciplina inteira sobre o assunto. Vale a pena consultar essas fontes na Internet para aprender muito mais sobre APIs. Falando nisso, os alunos que irão pesquisar sobre APIs devem mesmo acessar essas fontes, porque o material é realmente completo. Muitos assuntos não comentados aqui ainda podem ser foco de pesquisa nessa disciplina e material para apresentação oral por alunos. Exemplos de assuntos que podem ser explorados, caso alguém queira:

- Design de APIs RESTful
- Os fundamentos da segurança de APIs
- O papel de SOA e/ou microservices no mundo das APIs
- APIs em *Ecommerce*

Cada um desses assuntos tem conteúdo suficiente para uma apresentação oral de 1 hora. E a página [54] da Sensedia pode ser o ponto de partida para a pesquisa na web sobre esses 4 assuntos.

Depois de uma pesquisa em APIs fica evidente algumas das vantagens delas:

- As APIs entregam funcionalidades a *websites*:

- APIs estão por trás das aplicações/softwares online (SaaS);
- APIs suportam as aplicações móveis (Apps Mobile);
- APIs conectam objetos físicos, através dos conceitos extremamente interessantes de Internet das Coisas;
- Permitem integrar sua plataforma à parceiros (como em e-commerce ou SaaS diversos);
 - Gera **inovação** sobre seus próprios dados, de uma maneira que não poderia sozinho;
 - Gera desenvolvimento de sua própria plataforma em novos meios;
 - Gera mais mercado e **expandi o público-alvo**;
 - **Organiza o setor de TI**, em que os consumidores das APIs são as outras equipes internas da empresa, e as APIs expostas funcionam como portais de troca de dados entre setores.

Um dado interessante sobre as APIs é o quanto rápido vem crescendo a atuação delas no mercado de TI. O gráfico abaixo mostra o número de APIs públicas disponíveis até 2018.

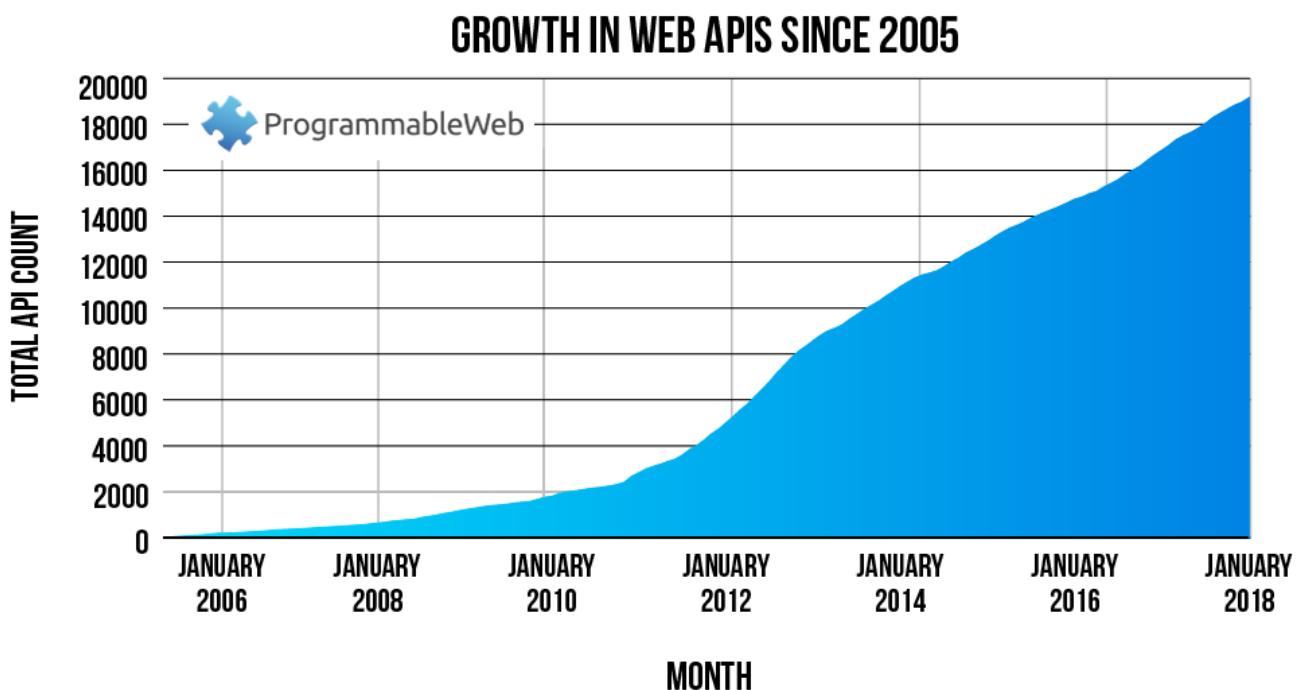


Figura 33. Fonte [46].

Todo esse esforço por traz das APIs tem um fundamento financeiro, é claro. Nenhuma empresa cria uma tecnologia nova se não for para dar retorno financeiro, direta ou indiretamente. E as APIs estão aí como uma estratégia muito inteligente de marketing, publicidade, fidelização de parceiras, clientes, etc. Para se ganhar dinheiro com as APIs algumas estratégias são desenvolvidas conforme o fluxograma abaixo. Em primeiro lugar, alguém faz a API, geralmente empresas com interesse por traz da tecnologia. Elas são de fato as donas das APIs e podem a qualquer momento retirá-las da web, se assim for desejado. Através das APIs os ativos, como dados e serviços computacionais, são expostos, dando acesso às capacidades da empresa, capacidades estas geralmente relacionada com o

core business de cada empresa. As APIs são então as interfaces de acesso às capacidades de uma empresa. Essas capacidades são então consumidas por clientes que tiram proveito delas, geralmente através de negócios direcionados às pessoas físicas que de alguma forma irão gastar dinheiro ao usar as APIs, direta ou indiretamente.

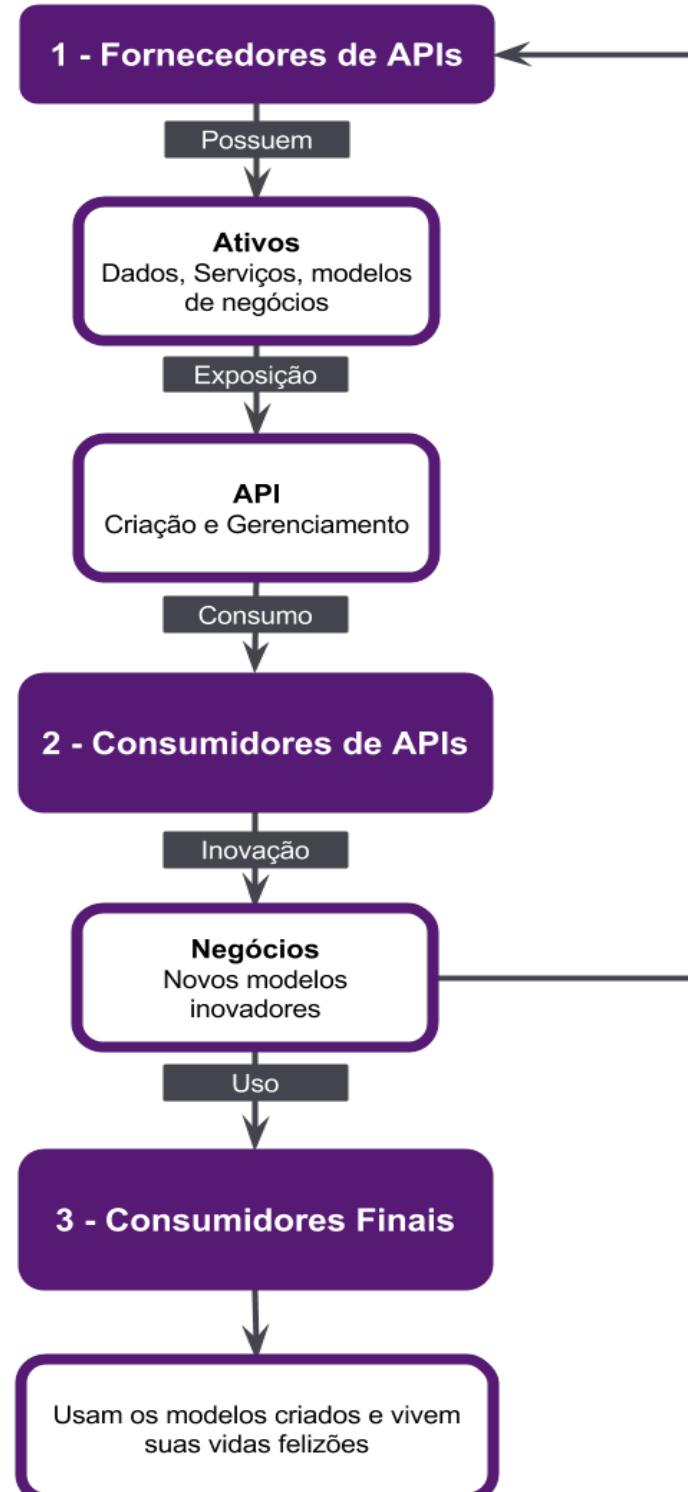


Figura 34 . Fonte [46].

E todos saem ganhando: pessoas físicas com serviços inovadores, desenvolvedores de software com facilidades em usar capacidades já instaladas em empresas e essas empresas que podem ser melhor divulgadas e vender seus serviços.

Muito mais poderia ser mostrado aqui, mas não haveria tempo para tal, como explicado. Contudo, está aberto o campo para as pesquisas. Qualquer API do Facebook, explicada com alguns detalhes, poderia ser um bom trabalho a ser mostrado nessa disciplina, por um aluno, por exemplo (página para início da pesquisa aqui : [56]). Esse capítulo deu várias sugestões de pesquisas a serem feitas, mas qualquer assunto relacionado às APIs é bem-vindo à sala de aula. Então basta entrar naqueles sites citados e se deparar com a vasta gama de assuntos a explorar. O site ProgrammableWeb tem mais de 12000 APIs documentadas e já foi propriedade da [Alcatel-Lucent](#). Uma grande contribuição para essa disciplina seria ler desse site o que há de explicações sobre segurança em API. Tal pesquisa pode iniciar na página [55].

Procure também o termo “APIX 2017”, “APIX 2018”, “APIX 2019” ou “APIX 2020” no Youtube, para acessar uma lista de vídeos muito bons sobre APIs! O APIX é o evento API Experience da Sensedia.

Boa sorte!

Capítulo III - SIP

Visão geral

Em telecomunicações o protocolo SIP está em evidência. A utilidade desse protocolo é propiciar a formação de sessões de *media* entre pares que desejam se comunicar. Numa rede IP, para que *devices* consigam se comunicar, enviando pacotes de áudio e vídeo um para o outro, eles precisam conhecer o endereço IP da outra ponta e também quais são as capacidades dela em relação ao tratamento de dados de áudio e vídeo. Ou seja, se A precisa se comunicar com B, A deverá conhecer o IP de B e vice-versa. Além disso, A e B precisarão negociar alguns parâmetros relacionados com o processamento da *media* em cada ponta. Mais precisamente, eles deverão concordar sobre os tipos de *codecs* de áudio e vídeo e também sobre portas para onde as *medias* devem ser enviadas. Em alto nível, o que deve acontecer, até que A e B consigam trocar *media* de áudio e vídeo com sucesso (sessão estabelecida), é o seguinte:

1 - A precisa descobrir qual é o IP de B, exatamente no momento em que A deseja chamar B. Se A souber qual é o IP corrente de B, então A poderá enviar sua *media* para o lugar correto. Ou seja, A enviará pacotes de áudio e vídeo para o nodo com IP correto. Mas, não adianta gravar o endereço IP de B, obtido em outras comunicações no passado (sessões estabelecidas em outras conversações), porque o IP de um dispositivo móvel pode mudar, quando ele mudar de rede ou até mesmo se estiver usando DHCP e for 'resetado'. Não é possível conhecer o IP previamente, a menos que o usuário na outra ponta passe o IP de B a A, de alguma forma antes de iniciar a comunicação. Mas isso seria inconveniente, desconfortável. O que é necessário aqui é a presença de um tipo de servidor entre A e B, que conheça o IP de cada um. Ou seja, um servidor poderia dizer a A qual é o IP de B e vice-versa. Assim, de forma automática, as pontas (*peers*) obteriam os dados necessários para iniciar uma comunicação.

2 – Portanto, A e B se registram no servidor (servidor SIP) dizendo a ele qual é o próprio IP de cada um naquele momento, antes de qualquer tentativa de comunicação com outro *peer*. Cada *peer* que irá usar o protocolo SIP tem em si um agente chamado UAC (*user agent client*) que conhece a pilha do protocolo SIP e portanto sabe como providenciar um registro no servidor SIP. O próprio UAC obtém o IP corrente e o passa ao servidor SIP, juntamente com um tipo de apelido para o *peer* respectivo. Por exemplo, B se regista no servidor SIP, o qual manterá um mapeamento do tipo <apelido, IP>. Assim, o servidor SIP pode conter em sua base de dados algo do tipo <8000, 192.168.21.40>. Isso significa que o usuário com o dispositivo B tem apelido (*sip user*) = 8000 e seu dispositivo tem IP = 192.168.21.40. Da mesma forma o dispositivo A pode enviar seus dados de registro ao *sip server*. Um *sip server* pode ser um Proxy, o que será visto no próximo capítulo. E um *sip server* contem uma base de dados, para registrar todos os dispositivos registrados nele. A base de dados se chama Registrar. No Registrar, uma tabela pode então conter o mapeamento de *sip users* e IPs. Para um *peer* se registrar no *sip server*, ele envia uma mensagem SIP, a mensagem SIP REGISTER para o servidor.

3 – Pelo visto, se o *device A* quer se comunicar com o *device B*, então no mínimo A deve saber que ele irá entrar em contato com o *sip user 8000*. Já o valor 8000 pode ser informado ao usuário de A, pelo usuário de B, porque o identificador de um *sip user* (valor 8000 por exemplo) não muda com o tempo. Ou seja, o usuário de B pode dizer ao usuário de A, durante um diálogo presencial : “ Olá, o meu login ou ID é 8000. Quando precisar falar comigo, chame-me no número 8000”. Como essa informação precisa ser passada do usuário de B ao de A somente uma vez, não se torna um inconveniente. Portanto, o que o UAC de A deve fazer é informar ao *sip server* que ele quer estabelecer uma sessão de *media* com B. Então, A envia uma mensagem SIP ao *sip server*, a mensagem SIP INVITE. Essa mensagem tem informações de A e implica em pedir ao *sip server* para chamar B. Então, a mensagem SIP INVITE leva consigo o ID = 8000 como o destinatário da chamada.

4 – O servidor SIP repassa a mensagem SIP INVITE ao dispositivo B, que nesse momento tem um agente pronto a recebê-la. É o UAS (*user agent server*). Portanto, um dispositivo que usa o protocolo SIP contém UAC e UAS. Até aqui somente o *sip server* conhece o IP de B e portanto é o *sip server* que realmente deve repassar a mensagem SIP INVITE. A mensagem SIP INVITE carrega consigo algumas informações sobre A. Por exemplo, quais são os *codecs* de áudio e vídeo presentes em A, quais são as portas em A onde B deve enviar seus pacotes de áudio e vídeo, qual é o IP atual de A e outras coisas.

5 – B recebe a requisição de A (SIP INVITE), encaminhada pelo *sip server*, e analisa quais são os *codecs* de áudio e vídeo que A contém. Se B contém também, no mínimo, um *codec* de áudio e um de vídeo contidos em A, então ocorre uma coincidência suficiente para ambos os *peers* conseguirem trocar pacotes de *media* úteis aos dois. Ou seja, se A e B podem usar os mesmos *codecs*, então eles poderão conversar entre si. Não faria sentido cada ponta usar *codecs* diferentes, o que causaria silêncio e ausência de vídeo.

6 – O *device B* toca (*ring*) e seu usuário pode atender a chamada, por exemplo. Se ele atender, o UAS de B envia imediatamente uma mensagem de confirmação de chamada aceita. Ou seja, o UAS informa que a chamada foi aceita. Essa informação segue até o *sip server* na forma de uma mensagem SIP OK. O *sip server* repassa a mensagem SIP OK para o dispositivo A. A mensagem SIP OK carrega consigo os dados do dispositivo B: IP, porta onde ele deseja receber os pacotes de *media* (áudio e vídeo), e outras informações.

7 - Quando o UAC de A recebe o SIP OK, ele verifica que B concordou em trocar *media* usando um dado par de *codecs* de áudio e vídeo. Ou seja, B aceitou um par de *codecs* que é existente em A também. Então isso permite a troca de *media* e a conversação pode fluir, com vídeo por exemplo.

8 – A e B agora conhecem o IP e porta corretos de destino (IP do *peer*) para onde devem enviar pacotes de *media*. Por exemplo, se A declarou que espera receber *media* de áudio na porta 4848 e que seu *codec* de áudio é o Speex, então B estará enviando pacotes de áudio para o IP de A, na porta 4848, codificados com Speex. Da mesma forma A estará enviando sua *media* para uma porta presente em B, com o *codec* Speex no caso de áudio. Desse ponto em diante A e B conhecem seus dados, um do outro, e podem trocar a *media* de forma direta, sem passar pelo *sip server*. Isso se chama *direct media*.

Resumidamente, o que o protocolo SIP fez nesse caso foi carregar informações de IPs, portas e *codecs* entre os *peers*. Mas, um *sip server* foi necessário.

O protocolo SIP tem mensagens formadas por cabeçalhos e corpo. Nos cabeçalhos estão as informações que permitem as mensagens serem roteadas corretamente entre os *peers*, passando por *proxies*, por exemplo. Os cabeçalhos indicam se as mensagens são do tipo INVITE, REGISTER, OK, ou outra qualquer. Mais detalhes serão vistos mais adiante. Além dos cabeçalhos, o corpo da mensagem SIP carrega, na verdade, outro protocolo. As informações com os IPs, portas e *codecs* para a troca de *media* é toda carregada no corpo da mensagem, mas para tal o corpo da mensagem carrega o protocolo SDP. Ou seja, a mensagem SIP carrega dados de SDP e os dados de SDP carregam as informações sobre portas, IPs e *codecs*. A sintaxe das mensagens SIP é parecida com aquela do protocolo HTTP. Como o protocolo é da camada de aplicação e tem a sintaxe semelhante àquela do HTTP, fica fácil ler as mensagens do SIP.

Atualmente o protocolo SIP é o mais utilizado no mundo, para a formação de sessão de *media*, em redes IP. Provavelmente devido ao uso de VoIP. Mas, a sessão de media não tem que ser sempre para voz e vídeo, ela pode ser apenas para vídeo em sistemas que acessam câmeras IP de vigilância, por exemplo. Além disso, pode-se usar o SIP para apenas criar sessões de trocas de mensagens texto. De fato, o protocolo SIP tem a SIP MESSAGE, que é uma mensagem SIP para carregar apenas texto em seu corpo. Isso é útil, por exemplo, para um software de *Instante Messages*.

Mesmo que o protocolo SIP não seja o mais novo dos assuntos escolhidos nessa disciplina e de fato deve ser o mais antigo, a sua importância em TI e Telecomunicações vem crescendo a cada dia, o que é motivo suficiente para constar nesse documento. Vale a pena iniciar entendimento nesse assunto! Quem tiver um conhecimento básico em SIP e conseguir trabalhar com esse protocolo certamente tará portas abertas no futuro. Um pouco de justificativa para estudar SIP está contida nas próximas explicações dessa visão geral sobre o protocolo.

Em primeiro lugar o protocolo SIP tem sido escolhido como o protocolo para todos os sistemas de VoIP nos últimos anos. E como esse protocolo tem sido usado por anos, os seus erros iniciais e limitações já foram descobertos e resolvidos. Ou seja, agora o SIP é bem estável.

Atualmente, existem os projetos das Smart Grids e o protocolo SIP tem sido apontado como a solução de transporte para as informações que serão geradas e trafegadas. As *smart grids* são redes elétricas inteligentes. São consideradas inteligentes porque serão dotadas de dispositivos e capacidades de comunicação a tal ponto que informações serão geradas nas unidades consumidoras de energia e tais informações serão enviadas às empresas produtoras da energia elétrica. Ou seja, sensores e medidores modernos serão instalados nas unidades consumidoras, para coletar dados e medidas sobre características do consumo da energia. Os dados serão enviados às empresas produtoras, via rede de Telecomunicações, e se transformarão em informações importantes para, por exemplo, tomada de decisões sobre próximos planejamentos de distribuição de energia. Então, as *smart grids* serão redes de duas vias, ou seja, enviarão energia num sentido e receberão dados em outro. Essa inteligência a ser criada nas redes de distribuição de energia elétrica permitirá também a coleta de energia gerada por micro produtores. Ou melhor, unidades consumidoras poderão gerar energia, por exemplo através de painéis solares ou eólica, injetar essa energia na rede elétrica e

reduzir a despesa com a energia gasta mensalmente. Tudo isso gerará dados a serem enviados às empresas que produzem a energia. Aqui o foco do assunto é a energia elétrica, mas uma rede inteligente pode ser construída também na distribuição de água, por exemplo. Como pode ser percebido, uma grande quantidade de dados será criada e faz-se necessário um protocolo adequado para transmiti-los. O protocolo SIP, por exemplo, mostra-se viável para essa tarefa.

Conforme dito em [57], algumas características do protocolo SIP são de grande interesse às *smart grids*. Por exemplo:

- Alta escalabilidade. Por exemplo, atualmente existem milhões de assinantes de telefonia via VoIP e as redes de telecomunicações trabalham bem. Então o SIP já se adequou bem a muitos usuários.
- Segurança. O SIP implementa mecanismos de segurança, como o TLS.
- Travessia fácil de *firewalls* e NATs usando STUN, TURN e ICE. Realmente é muito importante que seja fácil atravessar NATs!
- Possibilidade de mapear um único usuário para muitos dispositivos. Então, um usuário poderá empregar um mesmo identificador (*sip user ID*) a muitos dispositivos na sua rede inteligente, como termostatos, lâmpadas, etc. Nas *smart grids* os equipamentos eletrônicos também poderão gerar informações, de consumo por exemplo.

O papel do protocolo SIP tem forte relação com essa pós-graduação, já que dispositivos móveis (*smartphones*) são referenciados em várias disciplinas e eles podem executar *softphones* que usam VoIP, que por sua vez usa SIP. Ou seja, o SIP é realmente um protocolo de Telecomunicação em voga. Suas utilidades caem muito bem nos contextos de aplicativos usando dispositivos móveis e redes IP.

Em 2016-2018, o Inatel Competence Center (ICC) desenvolveu um projeto de vídeo porteiro para uma empresa do ramo de segurança residencial. E uma das funções desse porteiro é estabelecer telecomunicação com *smartphones* de pessoas interessadas em conversar com quem esteja diante do vídeo porteiro, quando o mesmo é utilizado. Ou seja, quando alguém aciona o video porteiro, na rua, o mesmo liga para terminais móveis e uma sessão de *media* é estabelecida, de tal forma que a pessoa no *smartphone* pode conversar com quem está frente ao porteiro e ainda ver o vídeo dela. E não importa em que rede a pessoa com o *smartphone* se encontra. Ou seja, não importa se a pessoa chamada está na rede local da residência ou numa rede remota alcançável pela Internet. Nesse caso, a sessão de *media* de áudio e vídeo é estabelecida sobre redes IP e portanto usa o SIP. Nesse vídeo porteiro há um SIP proxy dentro dele. Ou seja, todos os softwares terminais que se comunicam com SIP, sob o domínio desse porteiro, estarão registrados e online em tal SIP proxy. Além disso, caso uma pessoa chamada decida abrir o portão de sua residência remotamente, o aplicativo no *smartphone* envia uma SIP MESSAGE ao SIP proxy que então envia um comando ao portão local. Esse aplicativo de *smartphone* tem um UAC e um UAS SIP e envia as mensagens criptografadas com a execução do TLS.

É por esses e outros exemplos que fica clara a viabilidade de obter conhecimento em SIP, para quem trabalha com software e Telecomunicações.

Existem muitos outros exemplos de uso de SIP em projetos de software/Telecomunicações e muitos *gateways* para dispositivos da IoT já podem estar usando SIP. Vale a pena trazer para a sala de aula mais exemplos do uso de SIP em Telecomunicações, reforçando a importância em aprender esse protocolo, o que seria uma boa apresentação oral por um aluno!

O SIP é o resultado do trabalho do IETF para a definição de um protocolo de estabelecimento de sessão entre pares, pela Internet. Uma definição encontrada no website Wikipedia é suficiente aqui: “**Internet Engineering Task Force (IETF)** é um grupo informal internacional aberto, composto de técnicos, agências, fabricantes, fornecedores e pesquisadores, que se ocupa do desenvolvimento e promoção de standards para Internet, em estreita cooperação com o World Wide Web Consortium e ISO/IEC, em particular TCP/IP e o conjunto de protocolos Internet. O IETF tem como missão identificar e propor soluções a questões/problemas relacionados à utilização da Internet, além de propor padronização das tecnologias e protocolos envolvidos.

As recomendações da IETF são usualmente publicadas em documentos denominados Request for Comments (RFCs), sendo que o próprio SIP é descrito pela RFC 3160”

Mensagens do protocolo SIP

O protocolo de inicialização de sessões – SIP – é um protocolo de sinalização da camada de aplicação, desenvolvido para estabelecer, modificar, e encerrar sessões multimídia pela Internet. O protocolo SIP emergiu em meados da década de 90, a partir de uma pesquisa do professor Henning Schulzrinne, associado ao departamento de Ciência da Computação da Columbia University, e sua equipe de pesquisa, conforme citado em [58]. Posteriormente, este protocolo foi desenvolvido pelo *Internet Engineering Task Force (IETF)* e foi projetado para trabalhar com outros protocolos da Internet, tais como TCP, UDP, IP, DNS e outros, de acordo com [59]. A versão mais atual do SIP pode ser vista em [60].

Como o próprio nome diz, este protocolo permite que 2 pontos da rede estabeleçam sessões de mídia entre si. As principais funções de um protocolo de sinalização são:

- Localização do destinatário pelo remetente;
- Contatar o destinatário para determinar suas capacidades para o estabelecimento da sessão;
- Troca de informações entre remetente e destinatário estabelecendo a sessão;
- Modificação de sessões já estabelecidas;
- Encerramento de sessões existentes.

Há também outros protocolos de camada de aplicação, do IETF, usados com o SIP para proverem outros serviços de multimídia: SDP – *Session Description Protocol*, para descrever os parâmetros das mídias a serem trocadas, RTP – *Real-time Transport Protocol*, para transportar dados em tempo real, RTSP – *Real-time Streaming Protocol*, para controlar os pacotes das mídias e o MEGACO – *Media Gateway Control Protocol*, para controlar os *gateways*. Entretanto, o protocolo SIP é independente desses protocolos de camada de aplicação. Esse documento não irá descrever estes

outros protocolos. O objetivo é focar na função do SIP de localização de pontos finais a serem contatados.

O protocolo SIP provê serviço de mapeamento de nomes e de redirecionamento de chamadas. Dessa forma, o usuário de um sistema SIP tem mobilidade entre terminais da rede. Ou seja, um usuário com ID = 8000 pode estar online num terminal móvel numa rede qualquer, ou num computador em outra rede, sempre com o mesmo ID, porque o servidor SIP terá o mapeamento correto entre usuário e IP corrente, desde que o usuário esteja online no servidor, ou seja, enviando a mensagem SIP REGISTER regularmente. Se o *device* do usuário envia a mensagem SIP REGISTER regularmente, o servidor saberá que o registro do usuário está atualizado e então o IP declarado é confiável, ou seja, o usuário está mesmo no nodo com tal IP.

Este protocolo incorpora elementos de 2 protocolos amplamente utilizados na internet: HTTP e SMTP. Do HTTP, o SIP “tomou emprestado” a arquitetura cliente-servidor e o uso de *uniform resource locators* (URLs). Do SMTP, o SIP “tomou emprestado” um esquema de codificação em texto e um estilo de cabeçalhos para as mensagens. Por exemplo, o SIP reusa os cabeçalhos do SMTP tais como To, From, Date e Subject. O cabeçalho DATE é usado para registrar a data quando a requisição ou resposta foi enviada. Cada cabeçalho tem seu significado específico, sendo que alguns são obrigatórios e outros não. Toda a funcionalidade do SIP está contida nos cabeçalhos e na linha inicial das mensagens. E devido ao fato do SIP ser análogo ao HTTP na forma como as mensagens são construídas, os desenvolvedores podem fácil e rapidamente criar aplicações usando linguagens populares de programação como JAVA e C++, utilizando o SIP.

O SIP usa mensagens textuais para executar a sua sinalização necessária ao estabelecimento de sessões e tais mensagens são divididas em duas classes, como mostrado em [61]:

- *Request messages*: requer alguma ação da parte que inicia a chamada (do remetente),
- *Response messages*: indica o resultado da ação desempenhada pela parte chamada (ação do destinatário).

As *request messages* partem do UAC e as *response messages* partem do UAS.

Mesmo estando divididas nestes dois grupos, as mensagens do SIP apresentam um formato comum como mostrado abaixo:

General message

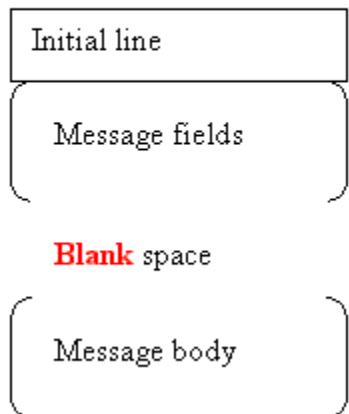


Figura 35

A linha inicial define a classe da mensagem. Os campos da mensagem contém atributos como:

To, From, Date, etc. Por exemplo, um atributo pode ser:

To: R. Pimenta <sip:pimenta@inatel.br>

No corpo da mensagem podem ser vistos os campos do protocolo SDP, por exemplo.

Se um usuário necessita, por exemplo, falar ou trocar algum dado com outro usuário, o remetente tem de enviar uma requisição para o destinatário, convidando-o a uma sessão de mídia. Para isso, é necessário iniciar o diálogo SIP enviando uma mensagem de requisição INVITE primeiramente. Então, a linha inicial da mensagem, neste caso, irá definir uma requisição INVITE. Para entender melhor, o seguinte diagrama de sequência ilustra o estabelecimento de uma sessão de mídia através do uso do SIP:

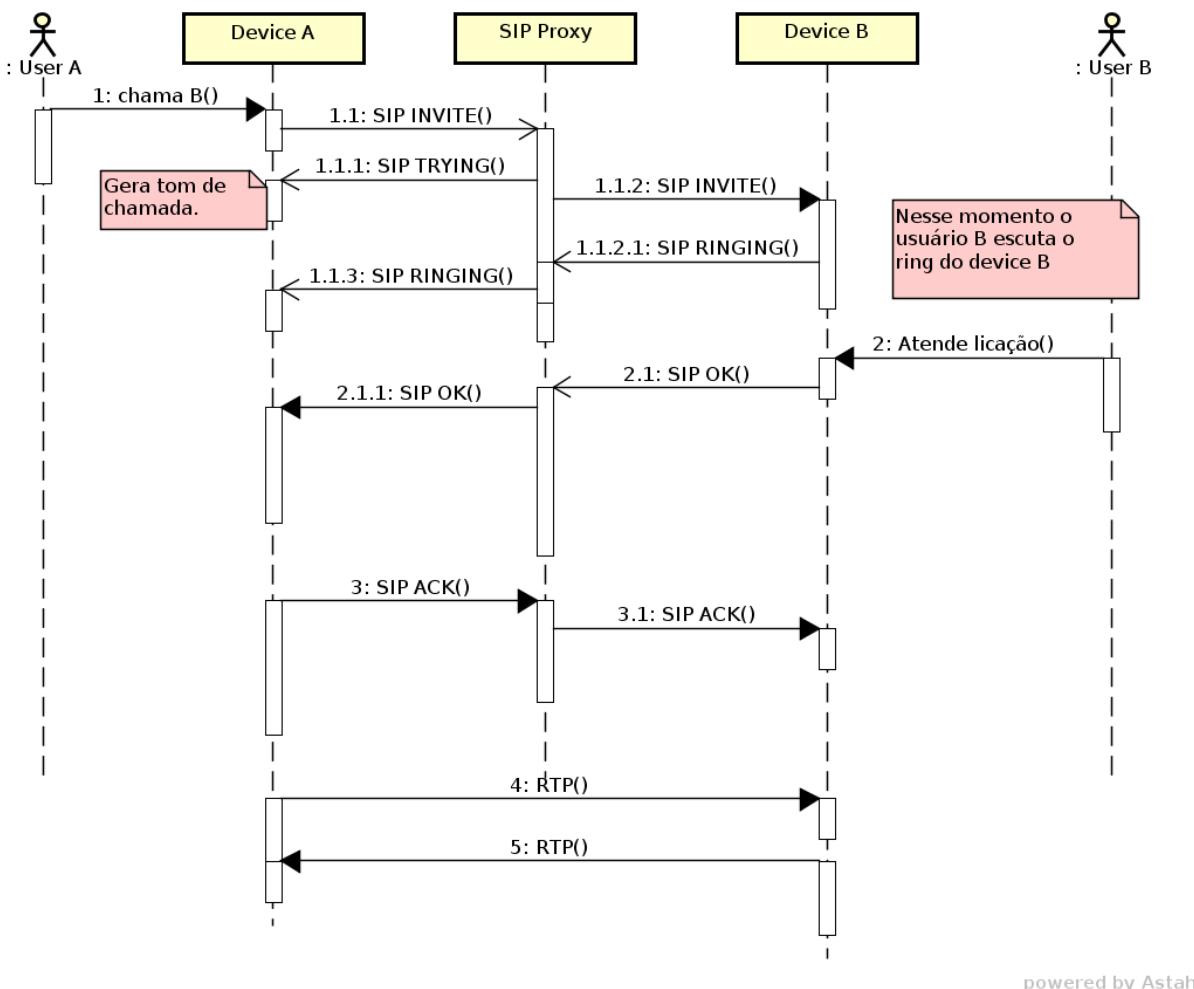


Figura 36

Com o auxílio do diagrama da figura 36, desse ponto em diante detalhes das mensagens SIP estão explicados a seguir.

Na figura 36 está considerada uma simples chamada do *device A* para o *Device B*, que é aceita e atendida. Além disso, está considerado que os *devices A* e *B* estão no mesmo domínio de rede, além do SIP Proxy. Ex: A e B estão conectados num mesmo roteador, bem como a máquina onde roda o *sip server* (o SIP proxy). Os IPs dessas 3 entidades poderiam ser, por exemplo: 192.168.0.101 (A), 192.168.0.102 (proxy) e 192.168.0.103 (B). O IP do próprio roteador poderia ser 192.168.0.1. Quando todos os *peers* estão no mesmo domínio de rede, será simples trocar os pacotes de RTP (*media* de áudio e/ou vídeo), porque os IPs são válidos para todos os *devices*. Ou seja, mesmo sendo IPs privados na rede, todos eles são alcançáveis por cada *peer*. Então, se o *peer A* tem IP 192.168.0.101, o *peer B* não terá problemas em mandar pacotes RTP para 192.168.0.101, porque 192.168.0.101 será visível à B, já que estão todos no mesmo domínio de rede. Portanto, quando as mensagens SIP INVITE e SIP OK (mensagens que carregam o SDP com as informações de IPs, portas e codecs) passam pelo *proxy*, o mesmo não precisa tomar qualquer medida. Ele apenas encaminha as mensagens SIP, porque o conteúdo delas é válido no SDP.

Entretanto, existem situações quando o *proxy* identifica que algum *peer* não está na mesma rede

local e que o *proxy* precisa cuidar de alterar os Ips encontrados no SDP vindo de tal *peer* remoto. Isso é uma correção de IP, sendo que um valor de IP privado é trocado por um valor de IP público, como será visto mais adiante.

Em relação às mensagens vistas na figura 36, vem:

1 – Usuário A chama B. Para tal o *device* A pode ser um *smartphone* ou computador, com *softphone*. Ou o *device* pode ser um *VoIP phone*. Exemplos de *softphone* são: Zoiper, Linphone, Mizudroid, Microsip. Existem vários outros. Cada um deles usa uma implementação da pilha SIP. Mas, todos eles devem usar uma implementação do protocolo SIP que siga a especificação da RFC do IETF. Caso contrário pode haver problemas na sinalização SIP. Por exemplo, o Linphone usa uma biblioteca chamada liblinphone, que tem a implementação da pilha SIP. O Microsip usa uma biblioteca chamada PJSip, que contem a implementação da pilha SIP. Essas duas implementações são condizentes com a RFC 3261 (RFC do SIP). Portanto, é seguro dizer que Microsip e Linphone podem se comunicar sem problemas de sinalização. Quando o usuário A decide chamar B, o seu *device* já deve estar online. Ou seja, o *device* A já deve estar registrado no *proxy*, mas o mesmo é verdadeiro para B.

1.1 - O *device* A envia uma mensagem SIP INVITE para o *proxy*. Aqui o *device* A conhece o *proxy* e nenhum outro dispositivo, em termos de endereço IP. Então obviamente o *proxy* é o único destino viável a enviar a mensagem SIP INVITE, pelo menos do ponto de vista de A. O *device* A conhece o *proxy* apenas porque o próprio usuário sabe qual é o seu IP ou nome de domínio e usa essa informação configurando-a no *softphone*, por exemplo, para deixá-lo *registrar-se* no *proxy*. Um exemplo de domínio onde um *softphone* pode ser registrado é sip.linphone.org. Esse domínio implica no IP da máquina onde roda o *sip server* do Linphone. A mensagem SIP INVITE enviada é uma *sip request*. É enviada do UAC. Essa mensagem carrega no seu corpo o conteúdo do protocolo SDP. O conteúdo do SDP leva o IP = 192.168.0.101, mas porta onde o *device* A irá ler os pacotes RTP que vierem de B. Segue junto a lista de todos os *codecs* que A contém (áudio e vídeo)

1.1.1 – Quando o *proxy* recebe a SIP INVITE ele imediatamente responde com SIP TRYING, independente do status do outro *peer* (*online* ou *offline*). Assim o UAC do *device* A aguarda, em silêncio, pela resposta do *device* B. A sabe então que o *proxy* já está tentando contatar B.

1.1.2 - Então o *proxy* simplesmente repassa o INVITE para o *peer* correto. Quando o *proxy* recebe o SIP INVITE, essa mensagem está direcionada à B. Como B já deve estar registrado e online, o *proxy* tem condição de saber qual é o IP atual de B. Assim, o *proxy* envia o SIP INVITE para o nodo correto na rede. Se o peer B estiver offline ou não registrado no momento da chamada, o *proxy* dá uma resposta com mensagem SIP sinalizando o erro. Ex: *time out* ou *not found*. Se o usuário do *device* B está online de fato em vários dispositivos, todos com o ID B, então o *proxy* repassará o SIP INVITE para todos eles. Todos esses *devices* tocarão (*ring*). O primeiro que atender fica com a chamada e o *proxy* envia um SIP CANCEL para todos os outros. Esse é o *fork* de chamadas.

1.1.2.1 - Quando o UAS do *device* B recebe a mensagem SIP INVITE, através dos dados no cabeçalho da mensagem, ele tem condição de saber que a mensagem é realmente para ele. Nesse momento o UAS envia um SIP RINGING para o *proxy*, como uma resposta. Isso indica que o

device B já está tocando (*ring*) e esperando pelo usuário aceitar e atender a ligação. A mensagem SIP INVITE contem informação suficiente para o *device B* saber para onde ele deve enviar a mensagem SIP RINGING. Ou seja, o UAS sabe em que nodo da rede está o *proxy*, porque ele recebeu, no cabeçalho da mensagem INVITE, a informação sobre qual *proxy* a transmitiu, em termos de endereço IP.

1.1.3 - O *proxy* simplesmente repassa a mensagem SIP RINGING para o *device A*. O *proxy* também tem condição de saber para onde ele deve encaminhar a mensagem SIP RINGING, conforme dados encontrados no cabeçalho da mensagem. Quando o SIP RINGING chega no *device A*, ele sai do silêncio e passa a soar o tom de chamando, para o usuário saber que o *device B* já está tocando.

2 - O usuário B atende no *device B*. Nesse momento o UAS em B monta a mensagem SIP OK. Essa é a resposta para A, que significa que a ligação foi atendida. Quando o UAS monta a mensagem SIP OK, ele coloca o conteúdo do SDP no *body* dessa mensagem SIP. Nesse momento, o conteúdo da mensagem SDP deve conter *codecs* de áudio e vídeo coincidentes com aqueles que estavam indicados no SDP da mensagem SIP INVITE. Nem todos precisam coincidir, mas no mínimo um par de cada um deve, para que possa haver áudio e vídeo durante o diálogo a seguir. Essa mensagem SIP OK é enviada pelo mesmo caminho onde havia sido enviada a mensagem SIP RINGING. Se o UAS de B demorar para enviar a mensagem SIP OK, o *device A* irá enviar uma mensagem SIP para o *proxy* solicitando o cancelamento da ligação.

2.1 - O SIP *proxy* simplesmente repassa a mensagem para o *device A*, assim como havia feito com a mensagem SIP RINGING. Para um exemplo onde todos os *peers* estão na mesma rede, nada é feito pelo *proxy* sobre o SDP. Caso contrário, este seria o momento do *proxy* interferir no conteúdo do SDP, colocando lá dados que façam sentido para o *peer* que irá recebê-los. Ou seja, o *peer* a receber esse conteúdo precisaria obter dados públicos do outro *peer*, não dados privados de outra rede qualquer.

2.1.1 - O UAC do *device A* recebe a mensagem SIP OK. Nesse momento o *device A* tem condições de saber, através do conteúdo do SDP, que o *device B* concorda em dialogar usando certos *codecs* em comum acordo. Imediatamente o tom de chamando no *device A* é cessado. Vale lembrar que o SDP contem os IPs e portas para onde enviar as *medias*, respectivamente de cada *peer* que o montou e enviou. Nesse momento já fica possível deixar que os sistemas montem seus pacotes de RTP (áudio e/ou vídeo) e os disparem em direção ao outro *peer*. Então os pacotes são enviados para o IP e porta recebido no SDP. E cada *peer*, 'escutará' seus pacotes RTPs recebidos em suas portas declaradas.

3 – Mas, antes da sessão ser estabelecida com pacotes RTPs, o *device A* envia uma *mensagem SIP ACK* ao *device B*, novamente através do *proxy*. Se a mensagem SIP ACK não chegar no *device B* em tempo, esse *device* emitirá outro SIP OK ao *device A*. Quando isso ocorre a sinalização falha toda e é fácil perceber na GUI do *softphone* que iniciou a chamada, porque fica difícil encerrá-la aí apenas clicando no botão de cancelar, por exemplo. O sistema passa a não responder bem, dependendo da implementação.

3.1 – O *proxy* repassa a mensagem SIP ACK para o *device B*.

4 - O device A passa a enviar pacotes de RTP diretamente para o device B. Relembre que agora A sabe bem para onde enviar os pacotes de media. Então aí forma-se a *direct media*.

5 - O device B passa a enviar pacotes de RTP diretamente para o device A. Desse ponto em diante os usuários dos dispositivos já estão conversando, pelo menos com áudio.

Uma boa apresentação oral a ser feita por um aluno dessa disciplina seria mostrar 2 *softphones* rodando num computador, ou em computadores diferentes, comunicando entre si usando SIP. Eles poderiam estar registrados em qualquer servidor SIP. E as mensagens SIP trocadas poderiam ser mostradas e explicadas com o Wireshark. Pode-se usar o Linphone para isso. O próprio Linphone tem um servidor SIP disponível na Internet, para uso gratuito.

Segue abaixo um exemplo de mensagem SIP REGISTER, usada para um dispositivo se registrar num SIP Proxy e ficar online. Esse tipo de mensagem não está na figura 36.

```
REGISTER sip:192.168.0.102;transport=tcp SIP/2.0
Via: SIP/2.0/TCP
192.168.0.101:62738;rport;branch=z9hG4bKPj3f5e8298e00d4184a5f93e07
23b6853a;alias
Max-Forwards: 70
From: "Rodrigo Notebook"
<sip:4000@192.168.0.102>;tag=e82d7eee1dc041d6bcba2dae7115afd2
To: "Rodrigo Notebook" <sip:4000@192.168.0.102>
Call-ID: fc6df3a945764135bdc5e1600cf97820
CSeq: 22068 REGISTER
User-Agent: MicroSIP/3.11.0
Supported: outbound, path
Contact: "Rodrigo Notebook"
<sip:4000@192.168.0.101:62738;transport=TCP;ob>;reg-
id=1;+sip.instance=<urn:uuid:00000000-0000-0000-0000-
00000c27160a>"
Expires: 300
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE,
NOTIFY, REFER, MESSAGE, OPTIONS
Authorization: Digest username="4000", realm="localhost",
nonce="000159e0000006e2dba4127f7f8460075d19efafb2579beb",
uri="sip:192.168.0.102;transport=tcp",
response="8f0e37e50af94229a181ea5c6e739950"
Content-Length: 0
```

Veja abaixo os comentários sobre alguns campos dessa mensagem, ou seja o significado do cabeçalho da mensagem;

REGISTER sip:192.168.0.102;transport=tcp SIP/2.0

Essa mensagem REGISTER é enviada ao IP 192.168.0.102, porque esse é o domínio onde o usuário decidiu se registrar. Ou seja, o usuário sabe que há um SIP *server* nesse IP e quer que seu *softphone* fique online no mesmo. Aqui o usuário poderia ter escolhido qualquer domínio disponível. Ex: sip.linphone.org ou outro IP. Na rede onde o usuário se encontra com seu dispositivo móvel, é necessário que haja rota até o nodo com o *sip server*. Nem sempre é um usuário que escolhe domínio de registro. Para sistemas que as máquinas interagem entre si por conta própria, um software com a pilha SIP implementada pode ter em si já definido o domínio onde se registrar. A palavra REGISTER indica ao *sip server* qual é o propósito dessa mensagem. O protocolo escolhido para transporte dessa mensagem é o TCP. Mas, geralmente o protocolo SIP é mais usado sobre UDP. A pilha SIP em uso é do SIP versão 2.0.

**Via: SIP/2.0/TCP
192.168.0.101:62738;rport;branch=z9hG4bKPj3f5e8298e00d4184a5f93e07
23b6853a;alias**

O campo *Via*, registra de onde a mensagem saiu, ou seja IP do nodo por onde ela está partindo. Essa mensagem partiu do IP 192.168.0.101. Os significados de *rport*, *branch* e *alias* poderiam ser explicados durante uma apresentação oral sobre SIP, por um aluno da disciplina. O campo *Via*, adicionado por todos os nodos onde a mensagem SIP passa, registra então a rota que uma mensagem toma do originador até o destinatário. Se um dos nodos, dessa rota, precisa dar uma *response* ao nodo adjacente anterior, ele usará o campo *via* para identificar onde mandar a resposta.

Max-Forwards: 70

Um mensagem SIP pode passar por vários nodos de rede, (proxies que fazem redirecionamento), mas essa REGISTER não pode ser encaminhada mais que 70 vezes. Se passar em 71 nodos, o 71º irá descartá-la.

**From: "Rodrigo Notebook"
<sip:4000@192.168.0.102>;tag=e82d7eee1dc041d6bcba2dae7115afd2**

Essa mensagem REGISTER é enviada em nome do usuário 4000 que está no domínio 192.168.0.102. Isso indica qual é o usuário responsável em pedir esse registro.

To: "Rodrigo Notebook" <sip:4000@192.168.0.102>

O campo acima indica qual usuário está se registrando. ID = 4000. Assim o proxy saberá que ele deve registrar o usuário 4000. O proxy é responsável em determinar de qual IP, na real, essa mensagem parte. Esse será o IP a ser associado ao ID 4000, por exemplo. Um nome para ser mostrado no *display* dos softphones pode acompanhar esse registro.

Call-ID: fc6df3a945764135bdc5e1600cf97820

Todas as mensagens relacionadas com esse REGISTER (Ex: confirmações com SIP OK) têm um identificador para associá-las entre si. Esse identificador é o Call-ID.

CSeq: 22068 REGISTER

As mensagens SIP podem usar número para sequenciamento delas, o que é útil quando o protocolo de transporte é UDP.

User-Agent: MicroSIP/3.11.0

O User-Agent indica o tipo de software usado como *softphone*. Nesse caso foi o Microsip.

Supported: outbound, path

O campo Supported pode ser explicado em apresentação por aluno em sala de aula.

Contact: "Rodrigo Notebook"

```
<sip:4000@192.168.0.101:62738;transport=TCP;ob>;reg-id=1;+sip.instance=<urn:uuid:00000000-0000-0000-0000-00000c27160a>"
```

Quando um UAC solicita o registro num servidor SIP, ele coloca o campo Contact na mensagem SIP REGISTER, para dizer onde que ele está na rede. Ex: dizer que está no IP 192.168.0.101 e alcançável na porta 62738. Mas, essa é a visão do próprio UAC. Quando a mensagem SIP REGISTER chega no *sip server*, ele confere se esses dados de contato estão realmente corretos ou atualizados. Para essa conferência, o *sip server* analisa, via dados da camada de transporte (protocolo TCP ou UDP), de onde que a mensagem realmente veio, em termos de IP e porta. Quando um UAC está atrás de um NAT, do ponto de vista do *sip server*, o endereço de onde a mensagem é recebida não é igual aquele citado em Contact.

Expires: 300

Quando um UAC se registra, ele diz ao *sip server* quanto tempo o registro deve durar. Isso significa que o *sip server* pode expirar o registro do UAC. Mas, é claro que o UAC irá enviar outra mensagem REGISTER a cada tempo igual ao indicado, para renovar seu registro e manter o cliente respectivo online 100% do tempo. Expires 300 vale 300 segundos (5 minutos). Se o UAC sair da rede ou for desligado, então não haverá outra mensagem REGISTER e o *sip server*, depois de 5 minutos, irá retirar o cadastro do UAC do seu Registrar.

Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS

Um UAC pode declarar que ele aceitará receber *sip requests* de certos tipos. Exemplos de tipos: BYE, CANCEL, MESSAGE, PRACK, etc. Nem todos esses tipos de mensagens SIP estão explicadas nesse documento. Portanto, seria de grande valia se um aluno dessa disciplina fizesse uma apresentação oral, em linhas gerais, sobre

as mensagens não explicadas aqui.

Authorization: Digest username="4000", realm="localhost",
nonce="000159e0000006e2dba4127f7f8460075d19efafb2579beb",
uri="sip:192.168.0.102;transport=tcp",
response="8f0e37e50af94229a181ea5c6e739950"

o campo acima está relacionado com a segurança de autenticação.
Esse campo também pode ser alvo de apresentação oral por parte de
um aluno.

Content-Length: 0

Algumas mensagens SIP tem o body. A SIP REGISTER não tem e
portanto o tamanho do seu conteúdo é zero.

A seguir está apresentada a mensagem de resposta que o SIP server envia ao UAC em relação à
mensagem SIP REGISTER processada:

SIP/2.0 200 OK

Via: SIP/2.0/TCP
192.168.0.101:62738;received=192.168.0.101;rport=62738;branch=z9hG
4bKPj3f5e8298e00d4184a5f93e0723b6853a;alias

From: "Rodrigo Notebook"

<sip:4000@192.168.0.102>;tag=e82d7eee1dc041d6bcba2dae7115afd2

To: "Rodrigo Notebook"

<sip:4000@192.168.0.102>;tag=1131bd920ba396ec69ab0fb8f3928a2c.8a82

Call-ID: fc6df3a945764135bdc5e1600cf97820

CSeq: 22068 REGISTER

Contact:

<sip:4000@192.168.0.101:62738;transport=TCP;ob>;expires=300

Server: OpenSIPS (2.2.1 (arm/linux))

Content-Length: 0

O que importa nessa mensagem, para o UAC é a resposta OK, significando que o registro foi bem
sucedido. A mensagem de OK é uma SIP *response*, esperada pelo UAC. Mensagens de *response* não
alteram o campo Via. Se uma mensagem SIP *request* passa por n nodos antes de chegar no destino,
ela terá n campos Via ao chegar no destino. Quando uma resposta volta pelo mesmo caminho
descrito pelos campos Via, em cada nodo no retorno o campo via mais acima é eliminado. Ou seja,
os campos Via funcionam como uma pilha que cresce no sentido da *request*. E nessa pilha os
campos Via são desempilhados no retorno da *response*. Um nodo decide colocar um campo Via na
mensagem quando ele quer participar da rota de retorno da resposta a ser enviada.

Na mensagem acima, no campo Contact, o *sip server* respondeu indicando qual foi o contato do
UAC realmente registrado. Ou seja, o *sip server* indicou que o contato do UAC está no IP

192.168.0.101, porta 62738. Esse é o contato que o SIP server enxerga onde está o UAC se registrando. Pelo visto o contato mantido pelo *sip server* é igual aquele citado pelo próprio UAC. Isso significa que não há um NAT entre UAC e *sip server*. Se o *sip server* receber uma *sip request* direcionada ao *user* 4000, por exemplo, ele irá encaminhá-la exatamente para o local indicado no Contact visto acima. O campo Contact tem utilidade para os *peers* nas pontas do diálogo, quando um deles precisa enviar nova *request* ao outro. É o campo Contact que indicará para onde enviar nova *request*. Então o endereçamento pelo Contact é de *peer* a *peer*, enquanto o endereçamento pelo Campo Via é de nodo a nodo numa trajetória da mensagem SIP.

Voltando à figura 36, um exemplo de mensagem SIP INVITE está visto abaixo. Durante um diálogo SIP, os IDs nos campos *From* e *To* não são modificados nas mensagens seguintes ao SIP INVITE.

```
INVITE sip:1000@192.168.0.102;transport=tcp SIP/2.0
Via: SIP/2.0/TCP
192.168.0.101:62738;rport;branch=z9hG4bKPj6be7a96925fb4d9eba4726b2
7e2bf0f0;alias
Max-Forwards: 70
From: "Rodrigo Notebook"
<sip:4000@192.168.0.102>;tag=298fbf9a539346ac80d669929aa85eea
To: <sip:1000@192.168.0.102>
Contact: "Rodrigo Notebook"
<sip:4000@192.168.0.101:62738;transport=TCP;ob>
Call-ID: 6e1bd0e23df447f0b03a051d2447cbf2
CSeq: 30460 INVITE
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE,
NOTIFY, REFER, MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800
Min-SE: 90
User-Agent: MicroSIP/3.11.0
Proxy-Authorization: Digest username="4000", realm="localhost",
nonce="000159ee000006e4a7998c1d89b0df5cbf7435eb241e1elf",
uri="sip:1000@192.168.0.102;transport=tcp",
response="7106a8370e3068d6e8b5c3e8e3f28b5b"
Content-Type: application/sdp
Content-Length: 306
v=0
o=- 3684076498 3684076498 IN IP4 192.168.0.101
s=pjmedia
```

```
b=AS:84
t=0 0
a=X-nat:0
m=audio 4000 RTP/AVP 8 110 101
c=IN IP4 192.168.0.101
b=TIAS:64000
a=rtcp:4001 IN IP4 192.168.0.101
a=sendrecv
a=rtpmap:8 PCMA/8000
a=rtpmap:110 speex/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-16
```

Abaixo está a explicação de alguns dos campos dessa mensagem:

INVITE sip:1000@192.168.0.102;transport=tcp SIP/2.0

O INVITE está direcionado ao usuário com ID 1000. O UAC que cria essa mensagem considera que o usuário 1000 está no mesmo domínio conhecido (único domínio conhecido). Portanto o uso de “@192.168.0.102”. Essa mensagem chegará ao *sip server* onde o usuário 4000 se registrou.

From: "Rodrigo Notebook"

<sip:4000@192.168.0.102>;tag=298fbf9a539346ac80d669929aa85eea

Na mensagem SIP INVITE o campo From do cabeçalho significa o remetente da mensagem. Portanto, o remetente é o *user* 4000.

To: <sip:1000@192.168.0.102>

O campo To implica em quem está sendo convidado a estabelecer sessão. Nesse caso, o *user* 1000.

Contact: "Rodrigo Notebook"

<sip:4000@192.168.0.101:62738;transport=TCP;ob>

Quando um UAC envia uma mensagem SIP INVITE ele acrescenta o campo Contact nela. O valor nesse campo será o mesmo recebido no campo Contact da mensagem SIP OK após envio da mensagem SIP REGISTER. Ou seja, é o contato que o *sip server* conhece de fato. Esse mesmo Contact será repassado ao destinatário quando a mensagem SIP INVITE for encaminhada pelo *sip server*.

Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS

O campo allow conta ao destinatário quais são as mensagens SIP que

o remetente aceita receber como *sip request*. Com a mensagem SIP OK é um *sip response*, ela não está nessa lista.

Session-Expires: 1800

O significado do campo *Sesion-Expires* pode ser analisado por um aluno dessa disciplina e então apresentado no momento de discorrer sobre o protocolo SIP.

Content-Type: application/sdp

Nessa mensagem o conteúdo no *body* da mensagem é do tipo SDP. Isso significa que o protocolo SDP está sendo usado e os valores de seus próprios campos estão contidos nesse INVITE. Não é obrigatório usar o protocolo SDP. O SIP pode carregar qualquer conjunto de *characteres* em seu *body*. Portanto, pode carregar JSON, por exemplo.

Content-Length: 306

Conforme o campo acima, o *body* dessa mensagem tem tamanho igual a 306 *characteres*. Segue abaixo o *payload* em SDP. O SDP não será discutido nessa disciplina em detalhes. Caso algum aluno se interesse por esse assunto, pode estudá-lo e fazer uma apresentação oral nessa disciplina apenas sobre o SDP. Abaixo está visível que o UAC deseja receber media no IP 192.168.0.101. A media deve estar codificada com o *codec* Speex, ou telephone-event, ou PCMA.

v=0

o=- 3684076498 3684076498 IN IP4 192.168.0.101

s=pjmedia

b=AS:84

t=0 0

a=X-nat:0

m=audio 4000 RTP/AVP 8 110 101

c=IN IP4 192.168.0.101

b=TIAS:64000

a=rtcp:4001 IN IP4 192.168.0.101

a=sendrecv

a=rtpmap:8 PCMA/8000

a=rtpmap:110 speex/8000

a=rtpmap:101 telephone-event/8000

a=fmtp:101 0-16

Abaixo está o exemplo da mesma mensagem SIP INVITE que chega no UAS B. Ou seja, depois que essa mensagem passa pelo *sip server (proxy)*. Assim tem-se a chance de verificar agora se houve alguma modificação produzida pelo *proxy*. Como visto, algumas modificações foram feitas e alguns atributos foram acrescentados à mensagem SIP. Nem tudo sobre SIP está explicado nesse documento. Portanto, a quem vá investigar mais detalhes sobre o SIP, para uma posterior apresentação oral em sala de aula, é recomendável dar uma olhada na RFC 3261. Essa RFC é grande e não compensa estudá-la inteira. Se alguém tentar fazer isso, verá que ao chegar no meio dela já estará esquecendo o que leu no início. É recomendável usar essa RFC procurando somente pelos termos a pesquisar. Por exemplo, o *proxy* acrescentou o atributo *rinstance* na primeira linha da mensagem. Para saber o significado disso, pode-se procurar pelas ocorrências desse termo na RFC do SIP. Algumas modificações estão destacadas na mensagem abaixo.

INVITE

sip:1000@192.168.0.104:37784;rinstance=9ae52270cb7d8415;transport=TCP SIP/2.0

Record-Route: <sip:192.168.0.102;transport=tcp;lr;did=32.7b671d45>

Via: SIP/2.0/TCP

192.168.0.102:5060;branch=z9hG4bKbec1.bf7a0852.0;i=17

Via: SIP/2.0/TCP

192.168.0.101:62738;received=192.168.0.101;rport=62738;branch=z9hG4bKPj6be7a96925fb4d9eba4726b27e2bf0f0;alias

Max-Forwards: 70

From: "Rodrigo Notebook"

<sip:4000@192.168.0.102>;tag=298fbf9a539346ac80d669929aa85eea

To: <sip:1000@192.168.0.102>

Contact: "Rodrigo Notebook"

<sip:4000@192.168.0.101:62738;transport=TCP;ob>

Call-ID: 6e1bd0e23df447f0b03a051d2447cbf2

CSeq: 30460 INVITE

Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS

Supported: replaces, 100rel, timer, norefersub

Session-Expires: 1800

Min-SE: 90

User-Agent: MicroSIP/3.11.0

Content-Type: application/sdp

Content-Length: 306

v=0

```
o=- 3684076498 3684076498 IN IP4 192.168.0.101
s=pjmedia
b=AS:84
t=0 0
a=X-nat:0
m=audio 4000 RTP/AVP 8 110 101
c=IN IP4 192.168.0.101
b=TIAS:64000
a=rtcp:4001 IN IP4 192.168.0.101
a=sendrecv
a=rtpmap:8 PCMA/8000
a=rtpmap:110 speex/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-16
```

A linha do INVITE passou a apresentar o IP verdadeiro do *user* 1000. Como o *proxy* sabe em que IP tal *user* está, ele tem condição de atualizar essa linha. Assim, quando a mensagem chegar no UAS B, ele saberá que a mensagem é para ele mesmo. Um novo campo Via foi acrescentado. O mais recente fica em cima do mais antigo, como numa pilha. No campo via mais antigo, foi acrescentado o atributo *received* e *rport*. Quando os valores desse campo são iguais aqueles já presentes nesse campo desde quando ele foi acrescentado pelo UAC, isso significa que não há um NAT entre UAC e *proxy*.

Antes mesmo que o UAS B interaja com o *softphone*, fazendo-o tocar (*ring*), uma mensagem SIP Trying é enviada ao proxy. Essa mensagem pode ser vista abaixo. Tal mensagem não está representada na figura 36.

SIP/2.0 100 Trying

```
Via: SIP/2.0/TCP
192.168.0.102:5060;branch=z9hG4bKbec1.bf7a0852.0;i=17
Via: SIP/2.0/TCP
192.168.0.101:62738;received=192.168.0.101;rport=62738;branch=z9hG
4bKPj6be7a96925fb4d9eba4726b27e2bf0f0;alias
To: <sip:1000@192.168.0.102>
From: "Rodrigo Notebook"
<sip:4000@192.168.0.102>;tag=298fbf9a539346ac80d669929aa85eea
Call-ID: 6e1bd0e23df447f0b03a051d2447cbf2
CSeq: 30460 INVITE
```

Content-Length: 0

Mas, quando o UAS B detecta que o *softphone* está tocando, ele imediatamente envia ao *proxy* a mensagem SIP RINGING, vista a seguir:

SIP/2.0 180 Ringing
Via: SIP/2.0/TCP
192.168.0.102:5060;branch=z9hG4bKbec1.bf7a0852.0;i=17
Via: SIP/2.0/TCP
192.168.0.101:62738;received=192.168.0.101;rport=62738;branch=z9hG
4bKPj6be7a96925fb4d9eba4726b27e2bf0f0;alias
Record-Route: <sip:192.168.0.102;transport=tcp;lr;did=32.7b671d45>
Contact:
<sip:1000@192.168.0.104:37784;rinstance=9ae52270cb7d8415;transport
=TCP>
To: <sip:1000@192.168.0.102>;tag=69f27306
From: "Rodrigo
Notebook"<sip:4000@192.168.0.102>;tag=298fbf9a539346ac80d669929aa8
5eea
Call-ID: 6e1bd0e23df447f0b03a051d2447cbf2
CSeq: 30460 INVITE
User-Agent: Z 3.3.25608 r25552
Content-Length: 0

Um UAS não acrescenta campos Via ao enviar mensagens SIP.

Assim que o usuário aceita e atende a ligação, o UAS envia um *sip response* como a mensagem abaixo (SIP OK):

SIP/2.0 200 OK
Via: SIP/2.0/TCP
192.168.0.102:5060;branch=z9hG4bKbec1.bf7a0852.0;i=17
Via: SIP/2.0/TCP
192.168.0.101:62738;received=192.168.0.101;rport=62738;branch=z9hG
4bKPj6be7a96925fb4d9eba4726b27e2bf0f0;alias
Record-Route: <sip:192.168.0.102;transport=tcp;lr;did=32.7b671d45>
Require: timer
Contact:
<sip:1000@192.168.0.104:37784;rinstance=9ae52270cb7d8415;transport
=TCP>
To: <sip:1000@192.168.0.102>;tag=69f27306

From: "Rodrigo
Notebook"<sip:4000@192.168.0.102>;tag=298fbf9a539346ac80d669929aa8
5eea
Call-ID: 6e1bd0e23df447f0b03a051d2447cbf2
CSeq: 30460 INVITE
Session-Expires: 1800;refresher=uac
Min-SE: 90
Allow: INVITE, ACK, CANCEL, BYE, NOTIFY, REFER, MESSAGE, OPTIONS,
INFO, SUBSCRIBE
Content-Type: application/sdp
Supported: replaces, norefersub, extended-refer, timer, X-cisco-
serviceuri
User-Agent: Z 3.3.25608 r25552
Allow-Events: presence, kpml
Content-Length: 190

v=0
o=Z 0 2 IN IP4 192.168.0.104
s=Z
c=IN IP4 192.168.0.104
t=0 0
m=audio 8000 RTP/AVP 110 101
a=rtpmap:110 speex/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=sendrecv

Reparar que o Call-ID permanece ainda o mesmo. O Campo Contact enviado daqui tem os dados de contato do *device* B, do ponto de vista do próprio *proxy*. O campo User-Agente diz que o agente (UAS) usado é o Zoiper versão 3.3. Portanto, o *device* B está rodando um *softphone* Zoiper. No SDP, está visível que o *device* B irá receber media no IP 192.168.0.104. E os *codecs* que ele topa usar são Speex e telephone-event. Sendo que o primeiro tem mais prioridade. Os IDs nos campos To e From permanecem os mesmos. Essa mensagem não sofre modificações relevantes ao passar pelo *proxy*.

O Call-ID idêntico em todas as mensagens significa que elas pertencem a um mesmo diálogo. Ou seja, correspondem a uma sinalização inteira, até que a sessão seja estabelecida. Já o atributo *branch* associa as mensagens que correspondem a um *sip request* inicial, mas que estão todas do

mesmo lado do *proxy*. Por exemplo, na figura 36 há mensagens do lado esquerdo e do lado direito do *proxy*. As mensagens que estão do mesmo lado e que pertence a uma mesma transação têm o mesmo valor de *branch*. As mensagens iniciadas com INVITE no *device A* e encerradas com OK recebido no *device A* pertencem à mesma transação e aquelas do mesmo lado do *proxy*, de uma mesma transação, contêm o mesmo valor de *branch*. Quando o UAC A recebe o SIP OK ele inicia outra transação, enviando um SIP ACK. Portanto, a nova mensagem SIP ACK tem novo valor de *branch*. Mas, o Call-ID ainda é o mesmo. Ou seja, uma nova transação num mesmo diálogo. Os atributos **tag** vistos em alguns campos do cabeçalho das mensagens SIP poderiam ser explicados durante uma apresentação sobre o SIP, por parte do aluno. A mensagem SIP ACK está vista abaixo:

```
ACK
sip:1000@192.168.0.104:37784;transport=TCP;rinstance=9ae52270cb7d8
415 SIP/2.0

Via: SIP/2.0/TCP
192.168.0.101:62738;rport;branch=z9hG4bKPj8c865b6906aa4a1190c24735
d54fc1eb;alias

Max-Forwards: 70

From: "Rodrigo Notebook"
<sip:4000@192.168.0.102>;tag=298fbf9a539346ac80d669929aa85eea

To: <sip:1000@192.168.0.102>;tag=69f27306

Call-ID: 6e1bd0e23df447f0b03a051d2447cbf2

CSeq: 30460 ACK

Route: <sip:192.168.0.102;transport=tcp;lr;did=32.7b671d45>

Content-Length: 0
```

É interessante notar que na primeira linha da mensagem acima já aparece o IP do destinatário *device B*. Isso é possível porque nesse ponto da sinalização o *device A* já recebeu informação para contato do *device B*.

O ACK visto acima também é reenviado pelo *proxy*, para o *device B*. Ao passar pelo *proxy* receberá mais um campo Via, como de praxe.

Quando o usuário no *device B* decide encerrar a ligação, o UAC B envia outra *sip request* ao *device A*. Dessa vez a mensagem SIP BYE, que pode ser vista abaixo:

```
BYE sip:4000@192.168.0.101:62738;transport=TCP;ob SIP/2.0

Via: SIP/2.0/TCP 192.168.0.104:37784;branch=z9hG4bK-d8754z-
779bdbb76f54266c-1---d8754z-

Max-Forwards: 70

Route: <sip:192.168.0.102;transport=tcp;lr;did=32.7b671d45>

Contact:
<sip:1000@192.168.0.104:37784;rinstance=9ae52270cb7d8415;transport
=TCP>
```

To: "Rodrigo Notebook"<sip:4000@192.168.0.102>;tag=298fbf9a539346ac80d669929aa85eea
From: <sip:1000@192.168.0.102>;tag=69f27306
Call-ID: 6e1bd0e23df447f0b03a051d2447cbf2
CSeq: 2 BYE
User-Agent: Z 3.3.25608 r25552
Content-Length: 0

Como essa mensagem SIP BYE é um *request* na direção device B->device A, então o primeiro campo Via adicionado entra já no device B. Mas essa mensagem chegará com dois campos Via no device A.

Essa mensagem também passa pelo proxy e é encaminhada ao device A. O device A deve então responder com um SIP OK. A resposta está vista abaixo:

SIP/2.0 200 OK
Via: SIP/2.0/TCP
192.168.0.104:37784;rport=49607;received=192.168.0.104;branch=z9hG
4bK-d8754z-779bdbb76f54266c-1---d8754z-
Call-ID: 6e1bd0e23df447f0b03a051d2447cbf2
From: <sip:1000@192.168.0.102>;tag=69f27306
To: "Rodrigo Notebook"
<sip:4000@192.168.0.102>;tag=298fbf9a539346ac80d669929aa85eea
CSeq: 2 BYE
Content-Length: 0

Como essa mensagem passa pelo proxy antes de chegar no device B, nesse sentido de retorno, o proxy retira ("desempilha") o campo Via mais recente e portanto essa mensagem se apresenta com apenas um campo Via aqui, que significa exatamente o nodo onde ela está sendo entregue.

A figura 36 não está representando essas mensagens de SIP BYE e SIP OK de resposta. Quando as mensagens SIP BYE e SIP OK são trocadas os devices encerram suas transmissões de pacotes de media. Isso encerra a sessão e é o fim do diálogo.

Toda a análise descrita acima considerou que os peers estão no mesmo domínio de rede. Com um dos peers em uma rede remota, haveria diferenças em certos campos das mensagens SIP.

O domínio de rede não é a mesma coisa que domínio SIP. O domínio SIP, que pode ser representado por um IP ou nome (Ex: sip.inatel.br), define um servidor SIP onde UACs podem se registrar. Se X clientes estão registrados num domínio SIP, então o *sip server* respectivo é responsável em manter o cadastro e dados de localização atualizados de X devices, por exemplo. Se um device online num domínio SIP sai da rede local (mesma rede definida pelo roteador onde está

conectado o nodo do *sip server*) e vai para uma rede remota (qualquer outra onde o *device* se conecta a outro roteador, obtendo outro endereço IP privado), então esse *device* muda de domínio de rede, mas ainda pode estar no mesmo domínio SIP se ainda continuar online no mesmo *sip server*. Quando um *device* está numa rede remota, o IP que ele obtém é privado na mesma rede (se IPv4 está em uso, e há um NAT) e não faz sentido na rede local do *sip server*. Da mesma forma, o IP local do *sip server* não faz sentido para o *device* que se posicionou numa rede remota. Nesse caso, se o *device* na rede remota deseja se registrar ainda no mesmo domínio que era usado na rede local, então o *sip server* deverá conter um IP público. Através do IP público do *sip server* fica possível se registrar nele. Além do IP público, também seria possível fazer registro nele através do nome do domínio, desde que o IP público estivesse mapeado ao nome do domínio num DDNS, por exemplo. Então, deve haver alguma empresa na Internet que suporte o registro de nomes e IPs, fazendo o papel de um DDNS. No Brasil existe a empresa Winco, que fornece serviço de DDNS. Uma explicação sobre DDNS poderia ser dada juntamente com a apresentação do protocolo SIP, por quem vá apresentá-lo oralmente em sala de aula. No site [62] há uma explicação inicial sobre DDNS e pode servir como ponto de partida para uma pesquisa.

Mais adiante serão demonstrados alguns detalhes com um *peer* em uma rede remota.

Detalhes técnicos sobre o SIP que não foram ditos até aqui podem ser obtidos através dos vídeos em [63], que são muito bem explicados, com didática melhor que esse próprio documento. São sete vídeos. Vale a pena ver esses vídeos para esclarecer mais dúvidas sobre SIP ou como referência à preparação de uma apresentação sobre o protocolo, oral a ser dada em sala de aula por aluno dessa disciplina. Muito do que não foi dito nesse documento, está explicado no vídeo em [63].

SIP em Telecomunicações

Porque relevar SIP? Conforme o 3GPP, em [64] está dito que o “*Session Initiation Protocol was selected as the signalling mechanism for IMS, thereby allowing voice, text and multimedia services to traverse all connected networks*”. Mas, o que é 3GPP e o que é IMS?

O **3rd Generation Partnership Project (3GPP)** é uma colaboração entre grupos de associações relacionadas com Telecomunicações. Por exemplo, associações de empresas de vários países. O escopo inicial do 3GPP era criar especificações para um sistema de telefonia móvel aplicável globalmente, de terceira geração (3G), baseado em GSM, dentro do escopo do ITU-T. Esse escopo foi aumentado posteriormente para incluir o desenvolvimento e manutenção de tecnologias tais como 4G e IMS. O projeto do 3GPP iniciou em 1998 e agora o time que o suporta fica localizado no ETSI (European Telecommunications Standards Institute), em Sophia-Antipolis, na França. Uma das pretensões do 3GPP era definir uma geração de rede *wireless* que deveria ser fundamentalmente capaz de suportar comunicações *wireless* baseadas no Internet Protocol.

O **IP Multimedia Subsystem (IMS)** é um *framework* arquitetural para a entrega de serviços *multimedia* baseados em redes *all IP*. Ou seja, o IMS especifica várias entidades lógicas numa rede IP (“*collection of different functions, linked by standardized interfaces*”), que facilitam ou provêm serviços de cobrança, transporte de *media*, VoIP, *stream* de video, etc. Historicamente os telefones

móveis têm provido chamadas de voz sobre uma rede no estilo switched-circuit, ao invés de um modelo estritamente baseado em pacotes comutados em redes IP. Por outro lado, métodos alternativos de entrega de voz e outros serviços de *multimedia* sobre redes IP têm se tornado disponíveis em *smartphones* (EX: VoIp e Skype), mas sem uma padronização pela indústria. O IMS é um *framework* arquitetural para prover tal padronização. O IMS foi originalmente desenhado pelo corpo de padronizações *wireless* do 3GPP, como uma parte da visão de evoluir as redes móveis para além do GSM. Para facilitar a integração com a Internet, o IMS usa protocolos do IETF, sempre que possível, por exemplo o SIP. De fato, no IMS, existem várias entidades, elementos e usuários que abrirão sessões entre si, através do protocolo SIP. Ou seja, o SIP é realmente muito empregado no IMS e o conhecimento de tal protocolo poderá abrir portas a quem vá trabalhar com IMS. Pesquisas sobre o uso de SIP em IMS, mostrando a importância, utilidade e vantagens desse protocolo nesse contexto, seria um bom trabalho a agregar valor à apresentação que poderá ser feita sobre SIP em sala de aula, por aluno dessa disciplina. Contudo, tem-se que pesquisar também sobre o futuro uso de IMS em 5G ou uso atual em 4G, para mostrar quais são as perspectivas sobre o IMS. Talvez o artigo do IEEE [65] dê uma boa visão ou proposta de como o IMS continuará sendo usado. Um usuário pode se conectar à arquitetura IMS de várias formas, mas geralmente usando o padrão IP. Terminais IMS (tais como telefones móveis ou computadores) podem se registrar diretamente no IMS, mesmo que estejam em outros países, ou seja, fora de rede, desde que estejam usando IP e executando SIP *user agents*.

Detalhes com peers em redes remotas

Essa seção mostra alguns detalhes no protocolo SIP, quando um *peer* em um diálogo está em outro domínio de rede, diferente daquele onde se encontra o SIP Proxy no qual ele fez registro e está online. Algumas informações abaixo podem se parecer com outras já dadas nesse capítulo, mas agora os exemplos abaixo usam outros endereços IPs e outros *user IDs*. Algumas semelhanças nas explicações acima e abaixo servirão para reforço do entendimento sobre o protocolo SIP, já que o mesmo contém muitos detalhes em sua especificação. Explicações adicionais estão presentes abaixo. Obs: para a leitura das explicações seguintes considere que existe um domínio onde é possível registrar clientes SIP. Esse domínio pode ser passado num *softphone*, por exemplo, e é público, ou seja, alcançável de qualquer ponto na Internet. Tal domínio tem nome = icchw.acmeddns.com.br e o IP público correspondente é 131.221.240.71.

O exemplo abaixo é composto por um *proxy* num nodo com IP público 131.221.240.71. O mesmo nodo tem IP privado = 192.168.0.101. Há também um *softphone* rodando numa rede remota (*user* 7000) (WLL-Inatel) e um *softphone* rodando num PC na rede local (*user* 8000). Tal PC não é o mesmo nodo do *proxy*. Dois Wiresharks foram usados: um em cada nodo onde estão os *softphones*. Assim, as mensagens SIPs abaixo mostram o conteúdo das mesmas ao partir dos *peers* ou ao chegar neles. Então, as mensagens abaixo permitem aferir quais modificações o Sip Proxy faz nas mesmas, antes de encaminhá-las aos *peers* destinatários.

Encontra-se abaixo um exemplo de mensagem SIP do registro do usuário 8000. Quando

o usuário 8000 usa seu *softphone*, ele define onde quer ficar online. Nesse exemplo, ele definiu ficar online no proxy que está no nodo com IP 192.168.0.101. E isso é possível, já que tal *softphone* está na mesma rede onde roda o SIP Proxy.

```
REGISTER sip:192.168.0.101;transport=TCP SIP/2.0
Via: SIP/2.0/TCP 192.168.0.104:47140;branch=z9hG4bK-d8754z-68e31a43281de2ca-1--d8754z-
Max-Forwards: 70
Contact: <sip:8000@192.168.0.104:47140;rinstance=e4e38ab5b8a71477;transport=TCP>
To: "8000-PC" <sip:8000@192.168.0.101;transport=TCP>
From: "8000-PC" <sip:8000@192.168.0.101;transport=TCP>;tag=d6681434
Call-ID: OWZhNGE2N2M3ZGZhZjRjZTlkMzk1YjRmYWl5ZTI0OWU.
CSeq: 2 REGISTER
Expires: 60
Allow: INVITE, ACK, CANCEL, BYE, NOTIFY, REFER, MESSAGE, OPTIONS, INFO, SUBSCRIBE
Supported: replaces, norefsub, extended-refer, timer, X-cisco-serviceuri
User-Agent: Z 3.3.25608 r25552
Authorization: Digest
username="8000",realm="localhost",nonce="00006abf000001c562eafc6cbc322d83c140c9b4d8657ccd",uri="sip:192.168.0.101;trans
port=TCP",response="49fe09d6b56b86eba400fdcd3b64e36",algorithm=MD5
Allow-Events: presence, kpml
Content-Length: 0
```

A primeira linha da mensagem mostra que o registro é para ser feito no proxy que está no IP 192.168.0.101.

O campo *Via* indica por onde a mensagem passou até chegar no destinatário. Nesse caso, como ela foi originada numa máquina com IP 192.168.0.104, esse é o IP que vai nesse primeiro campo *Via*. Se a mensagem passar por vários *proxies* sendo redirecionada até chegar no *proxy* alvo, vários outros campos *Via* serão empilhados no cabeçalho da mensagem. Caso o proxy precise dar uma resposta a essa mensagem, tal resposta será direcionada ao nodo indicado no campo *Via*, na porta 47140.

O campo *Contact* registra qual é o contato do usuário 8000. Ou seja, caso uma nova transação seja iniciada enviando mensagem para 8000, esse será o contato considerado.

O campo *From* indica quem é o originador da mensagem SIP REGISTER.

O campo *To* indica em nome de quem o *proxy* deve criar o registro. Isso implica que o *proxy* irá registrar o usuário 8000 online.

O campo *Expires* indica que o usuário 8000 ficará online 60 segundos. Depois disso o registro expira, se o agente SIP do *softphone* respectivo não enviar nova requisição de registro. Ou seja, a mensagem SIP REGISTER deve ser enviada de tempo em tempo, para manter o usuário online.

A mensagem abaixo representa a confirmação do registro dada pelo proxy:

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP 192.168.0.104:47140;received=192.168.0.104;rport=46494;branch=z9hG4bK-d8754z-68e31a43281de2ca-1--d8754z-
To: "8000-PC" <sip:8000@192.168.0.101;transport=TCP>;tag=eceb1bcf84571fc39d3d39a20adc6c5a.fb7e
From: "8000-PC" <sip:8000@192.168.0.101;transport=TCP>;tag=d6681434
Call-ID: OWZhNGE2N2M3ZGZhZjRjZTlkMzk1YjRmYWl5ZTI0OWU.
CSeq: 2 REGISTER
Contact: <sip:8000@192.168.0.104:47140;rinstance=e4e38ab5b8a71477;transport=TCP>;expires=60
Server: OpenSIPS (2.2.1 (arm/linux))
Content-Length: 0
```

Na mensagem acima está visível que o *proxy* declara que o contato do usuário 8000 é no IP 192.168.0.104 e na porta 47140. Ou seja, esse é o endereço que o *proxy* considerará futuramente, caso precise verificar onde está o usuário 8000.

É interessante notar que no campo *Via* o *proxy* acrescentou “`received=192.168.0.104`”. Isso implica que o IP declarado pelo próprio UAC 8000 é igual ao IP de onde a mensagem SIP Register realmente partiu. Então, do ponto de vista do *proxy*, o usuário 8000 não está atrás de um NAT, por exemplo.

Assim que o UAC 8000 recebe tal mensagem de OK, ele sabe que já está online e pode deixar que a GUI do softphone respectivo mostre esse status.

Caso o usuário 8000 troque de máquina, ou de rede, ou a máquina dele seja “*resetada*,” outro IP pode ser definido e nesse caso o *softphone* enviará um novo pedido de registro (SIP Register) já considerando o novo IP. Nesse caso, o *proxy* atualizará o registro de 8000 e passará a considerar então outro ponto na rede ou outro IP como contato ao usuário 8000. Veja abaixo uma nova configuração de rede após 'resetar' os equipamentos e terminais, agora com IPs modificados:

Servidor SIP Proxy = IP 192.168.0.103 e 131.221.240.71

PC com Zoiper usuário 8000 = IP 192.168.0.102 (está na mesma rede que o SIP Proxy)

Notebook com Microsip usuário 7000 = IP 10.0.216.12 (está em rede remota)

Com esta nova configuração, veja como ficam as mensagens de registro do usuário 8000 que está no PC na rede local:

```
REGISTER sip:192.168.0.103;transport=TCP SIP/2.0
Via: SIP/2.0/TCP 192.168.0.102:47140;branch=z9hG4bK-d8754z-61334a35647b3477-1---d8754z-
Max-Forwards: 70
Contact: <sip:8000@192.168.0.102:47140;rinstance=4db6c692ad82fc22;transport=TCP>
To: "8000-PC"<sip:8000@192.168.0.103;transport=TCP>
From: "8000-PC"<sip:8000@192.168.0.103;transport=TCP>;tag=24cd6447
Call-ID: YWEyNGJiMTI0ODBiZTNhOGUwZjg1Y2QwN2Y2N2RiZWY.
CSeq: 2 REGISTER
Expires: 60
Allow: INVITE, ACK, CANCEL, BYE, NOTIFY, REFER, MESSAGE, OPTIONS, INFO, SUBSCRIBE
Supported: replaces, norefersub, extended-refer, timer, X-cisco-serviceuri
User-Agent: Z 3.3.25608 r25552
Authorization: Digest
username="8000",realm="localhost",nonce="0002852b000000ab74a67ab80ab15e30f50515751607b18d",uri="sip:192.168.0.103;tra
nsport=TCP",response="d7080f22efea613d52a4e1dbc13a3d7",algorithm=MD5
Allow-Events: presence, kpml
Content-Length: 0
```

Importante: mesmo que o UAC do user 8000 declare que ele está alcançável pela porta 47140, como visto no campo *Via* da mensagem acima, tal mensagem pode ter sido enviada por outra porta até o *proxy*, na real, usando TCP. Isso pode ser visto na janela do *Wireshark*:

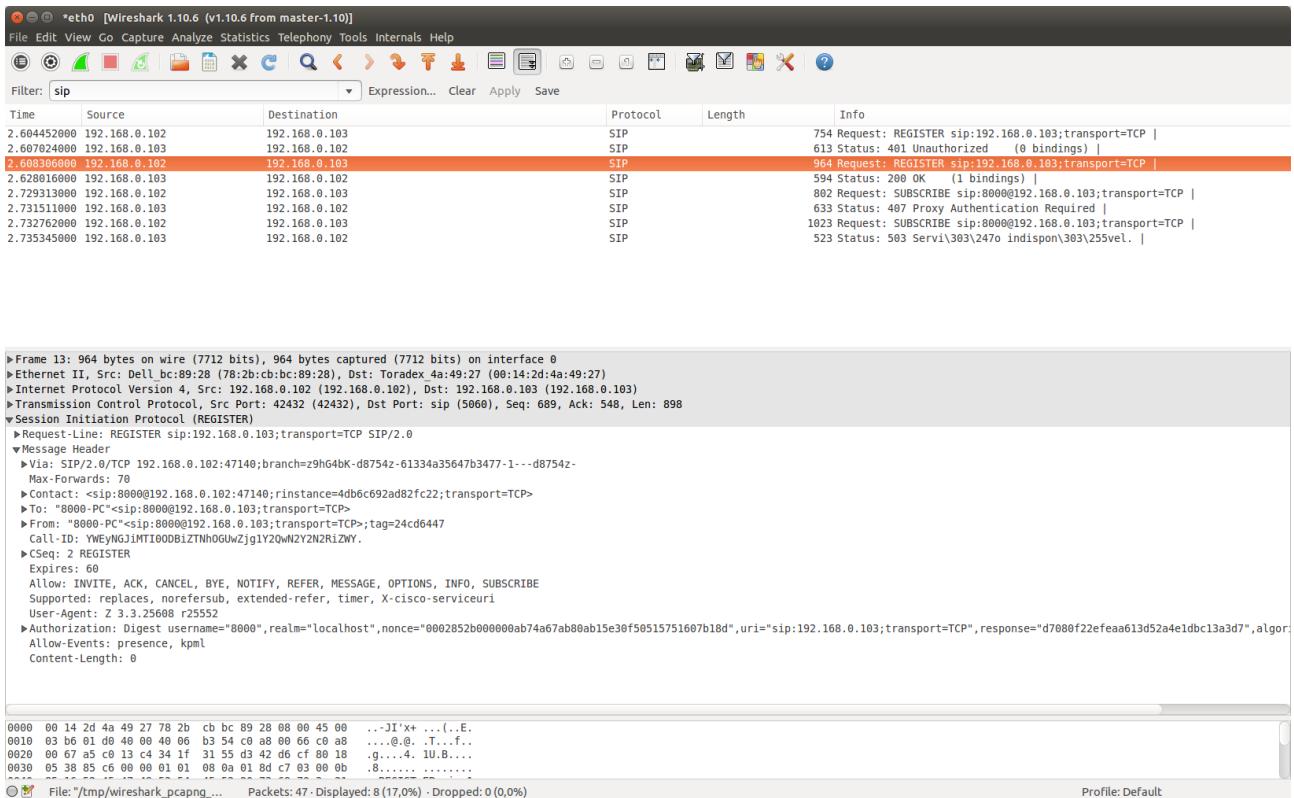


Figura 37 Fonte: projeto do ICC em 2016.

Na figura acima está visível que a mensagem SIP Register foi enviada, na real, pela porta 42432. Nesse caso, o SIP proxy acrescenta essa informação atualizada no campo Via, ao dar uma resposta SIP OK para o UAC. Vê-se abaixo na resposta, que no campo Via agora há o novo parâmetro rport. Isso implica que , quando o proxy precisar enviar mensagem ao UAC, ele enviará a mensagem para a porta atualizada 42432.

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP 192.168.0.102:47140;received=192.168.0.102;rport=42432;branch=z9hG4bK-d8754z-61334a35647b3477-1--d8754z-
To: "8000-PC" <sip:8000@192.168.0.103;transport=TCP>;tag=4ec2e9176cdb769f4b9ecc950c7ec67.f2bf
From: "8000-PC" <sip:8000@192.168.0.103;transport=TCP>;tag=24cd6447
Call-ID: YWEyNGJiMTI0ODBiZTNhOGUwZjg1Y2QwN2Y2RiZWY.
CSeq: 2 REGISTER
Contact: <sip:8000@192.168.0.102:47140;rinstance=4db6c692ad82fc22;transport=TCP>;expires=60
Server: OpenSIPS (2.2.1 (arm/linux))
Content-Length: 0
```

Pela figura abaixo pode-se ver que realmente a mensagem SIP OK foi enviada para a porta 42432 :

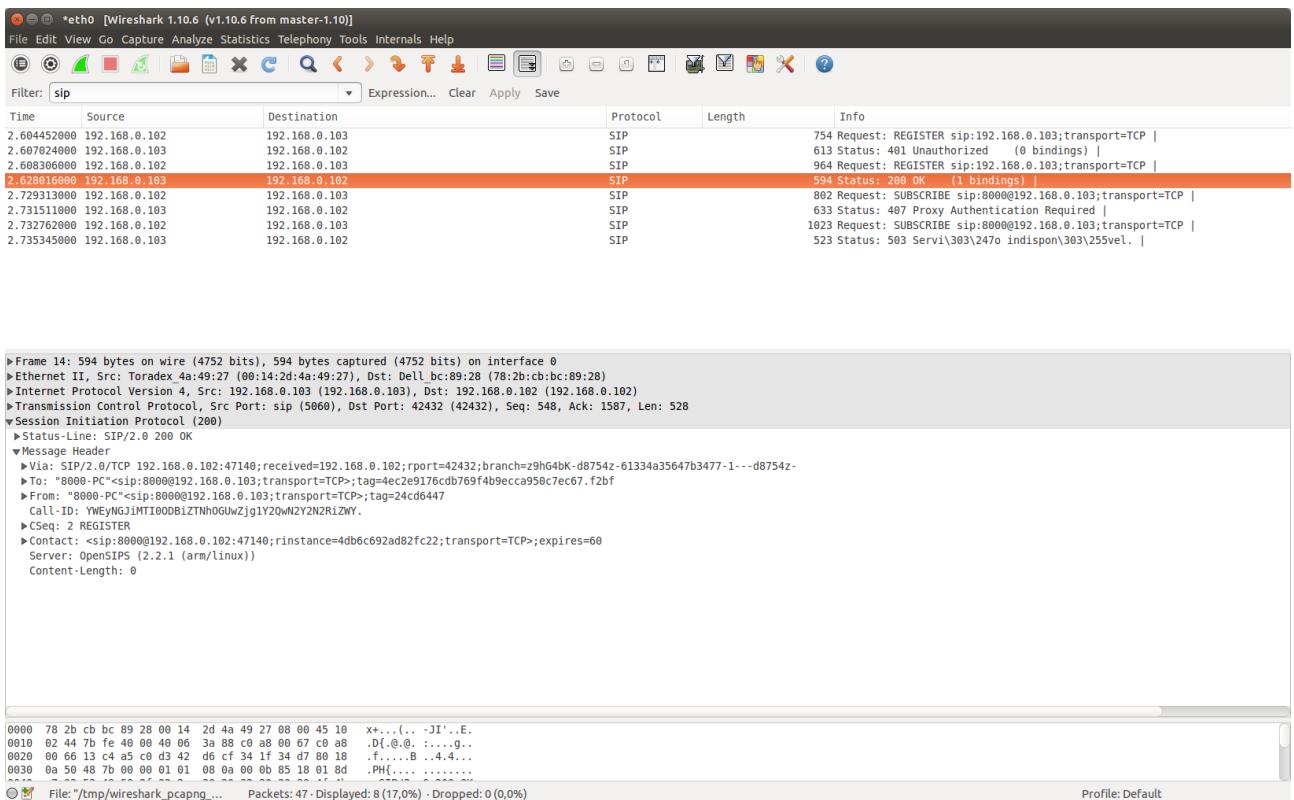


Figura 38 Fonte: projeto do ICC em 2016.

A mensagem abaixo mostra o registro do usuário 7000, que vem de uma rede remota:

```

REGISTER sip:icchhw.acmeddns.com.br;transport=tcp SIP/2.0
Via: SIP/2.0/TCP 10.0.216.12:62581;rport;branch=z9hG4bKPjd1942832a91b458ab7ed134155226eaa;alias
Max-Forwards: 70
From: "7000-notebook" <sip:7000@icchhw.acmeddns.com.br>;tag=0b35de3136de466b87a49536211445b9
To: "7000-notebook" <sip:7000@icchhw.acmeddns.com.br>
Call-ID: 954dc55e433c40a7a9546e98b2f9cf43
CSeq: 39034 REGISTER
User-Agent: MicroSIP/3.11.0
Supported: outbound, path
Contact: "7000-notebook" <sip:7000@10.0.216.12:62581;transport=TCP;ob>;+sip.ice;reg-id=1;+sip.instance=<urn:uuid:00000000-0000-0000-0000-00000c27160a>""
Expires: 300
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Authorization: Digest username="7000", realm="localhost", nonce="00028d4b0000019870a2470a10efafa259fafc30dfc9b3d8", uri="sip:icchhw.acmeddns.com.br;transport=tcp", response="44ea2d279786f60edbfb0af9cf60c0965"
Content-Length: 0
    
```

Para o caso da mensagem acima, o usuário 7000 está numa rede remota e não conhece o IP privado do SIP Proxy, mas conhece um domínio que implica no mesmo: icchhw.acmeddns.com.br . O domínio já tem que estar mapeado num DDNS apontando então para o IP 131.221.240.71. Nesse caso, ele registra o seu softphone Microsip no domínio conhecido.

Na imagem abaixo está visível que a mensagem SIP Register foi enviada realmente para o IP 131.221.240.71 e porta 5060, que é a porta default para SIP.

Mesmo que o UAC do usuário 7000 conheça o domínio, ele ainda declara em seu contato o seu IP privado, como pode ser visto em “[Contact: "7000-notebook" <sip:7000@10.0.216.12:62581...>](mailto:sip:7000@10.0.216.12:62581...)”. Já os campos From e To são criados com o nome do domínio e não IP.

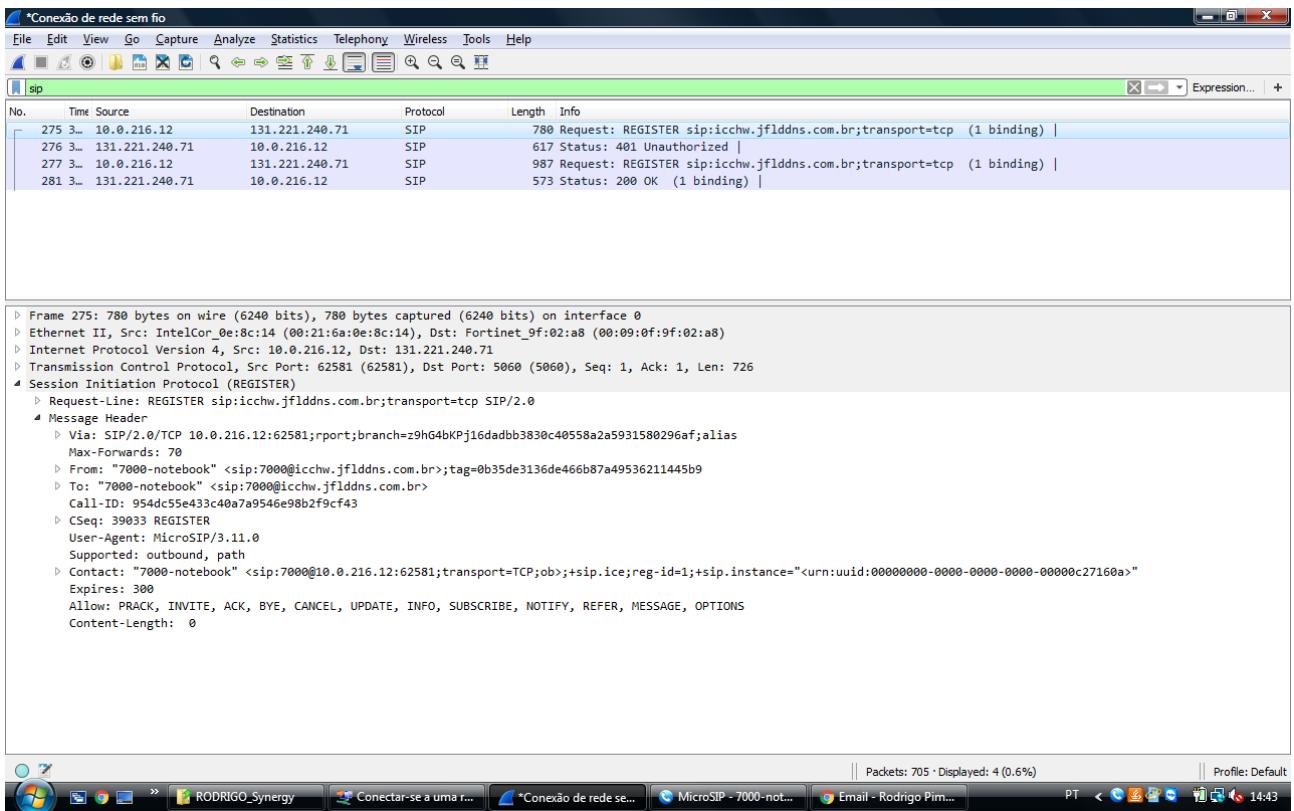


Figura 39 Fonte: projeto do ICC em 2016.

Veja abaixo a resposta à mensagem acima, dada pelo SIP Proxy:

```

SIP/2.0 200 OK
Via: SIP/2.0/TCP 10.0.216.12:62581;received=10.0.216.12;rport=62581;branch=z9hG4bKPj1942832a91b458ab7ed134155226eaa;alias
From: "7000-notebook" <sip:7000@icchw.acmeddns.com.br>;tag=0b35de3136de466b87a49536211445b9
To: "7000-notebook" <sip:7000@icchw.acmeddns.com.br>;tag=4ec2e9176cdb769f4b9ecc950c7ec67.10ad
Call-ID: 954dc55e433c40a7a9546e98b2f9cf43
CSeq: 39034 REGISTER
Contact: <sip:7000@10.0.216.12:62581;transport=TCP;ob>;expires=300
Server: OpenSIPS (2.2.1 (arm/linux))
Content-Length: 0

```

Como visto no campo *Via* acima, o IP declarado pelo UAC é igual ao IP reportado pelo proxy, no parâmetro *received*. Isso implica que o proxy vê o UAC 7000 no mesmo IP de onde o próprio UAC declarou estar. Portanto, do ponto de vista do proxy, o IP do UAC é um IP público, e não privado. Isso implica que não há um NAT entre proxy e UAC 7000.

Até aqui foi visto sobre o registro de softphones num SIP Proxy Registrar. Agora serão vistas as transações SIP executadas durante uma dialog. Nesse exemplo abaixo, o usuário 8000 (rede local) liga para o usuário 7000 (rede remota), que aceita a ligação, mas desliga após alguns segundos.

```

INVITE sip:7000@192.168.0.103;transport=TCP SIP/2.0
Via: SIP/2.0/TCP 192.168.0.102:47140;branch=z9hG4bK-d8754z-725b562ed2aabadd-1---d8754z-
Max-Forwards: 70
Contact: <sip:8000@192.168.0.102:47140;transport=TCP>
To: <sip:7000@192.168.0.103;transport=TCP>
From: "8000-PC" <sip:8000@192.168.0.103;transport=TCP>;tag=18bb061a
Call-ID: NGYyZmYwZDdjYjY0MGViMmRkZDI5MTk2OTM4MGJmNDc.
CSeq: 2 INVITE
Allow: INVITE, ACK, CANCEL, BYE, NOTIFY, REFER, MESSAGE, OPTIONS, INFO, SUBSCRIBE
Content-Type: application/sdp
Proxy-Authorization: Digest

```

```

username="8000",realm="localhost",nonce="000296a9000002a916b04f5c23ac85fb7bc2545ad44634f2",uri="sip:7000@192.168.0.10
3;transport=TCP",response="972cc7abf75491d63d18fb485ba4d81",algorithm=MD5
Supported: replaces, norefersub, extended-refer, timer, X-cisco-serviceuri
User-Agent: Z 3.3.25608 r25552
Allow-Events: presence, kpml
Content-Length: 241

```

A mensagem acima é um SIP INVITE. Parte do usuário 8000, para o usuário 7000, como pode ser visto nos campos From e To, mas passa pelo *proxy* no meio do caminho. A mensagem INVITE é enviada para o mesmo *proxy* onde o usuário 7000 se registrou, obviamente. Então, a primeira linha da mensagem aponta para o IP 192.168.0.103. O *user agent client* (UAC) coloca o campo *Via* no cabeçalho da mensagem, para indicar de onde ela originou.

Através do campo *Contact* o UAC diz ao *proxy* onde encontrá-lo, caso surja uma nova transação SIP com mensagem a ser enviada a esse UAC. O valor desse campo *Contact* chegará no destinatário 7000, mas com o IP privado de 8000 e isso não será útil, já que o agente SIP do usuário 7000 não conhece o IP privado de seu *peer*.

O campo *User-Agent* diz qual é o agente SIP que iniciou esse SIP INVITE. Nesse caso é um agente Z (= Zoiper).

O campo *Content-Length* mostra que o *body* da mensagem tem 241 *characteres*, visto abaixo. Esse conteúdo é o protocolo SDP (Session Description Protocol). No protocolo SDP, o campo "o" (o=Z 0 0 IN IP4 192.168.0.102) indica que o UAC aceitará pacote de RTP de mídia no IP 192.168.0.102, que é o seu IP obviamente. Entretanto, esse IP não será útil ao usuário 7000 numa rede remota. Portanto, uma das funções do *proxy* é corrigir esse IP e substituí-lo por um outro que faça sentido para o agente SIP do usuário 7000, como será visto adiante.

```

v=0
o=Z 0 0 IN IP4 192.168.0.102
s=Z
c=IN IP4 192.168.0.102
t=0 0
m=audio 8000 RTP/AVP 3 110 8 0 98 101
a=rtpmap:110 speex/8000
a=rtpmap:98 iLBC/8000
a=fmtp:98 mode=20
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=sendrecv

```

A mensagem abaixo mostra como essa mensagem SIP INVITE chega finalmente no usuário 7000 na rede externa. Ou seja, como ela fica modificada após passar pelo SIP Proxy:

```

INVITE sip:7000@10.0.216.12:62581;transport=TCP;ob SIP/2.0
Record-Route: <sip:icchhw.acmeddns.com.br;transport=tcp;lr;did=3a1.f41290b1>
Via: SIP/2.0/TCP icchhw.acmeddns.com.br:5060;bran...
Via: SIP/2.0/TCP 192.168.0.102:47140;rport=42432;received=192.168.0.102;bran...
Max-Forwards: 70
Contact: <sip:8000@192.168.0.102:47140;transport=TCP>
To: <sip:7000@192.168.0.103;transport=TCP>
From: "8000-PC" <sip:8000@192.168.0.103;transport=TCP>;tag=18bb061a
Call-ID: NGYyZmYwZDdjYjY0MGViMmRkZDI5MTk2OTM4MGJmNDc.
CSeq: 2 INVITE
Allow: INVITE, ACK, CANCEL, BYE, NOTIFY, REFER, MESSAGE, OPTIONS, INFO, SUBSCRIBE
Content-Type: application/sdp
Supported: replaces, norefersub, extended-refer, timer, X-cisco-serviceuri

```

User-Agent: Z 3.3.25608 r25552
Allow-Events: presence, kpml
Content-Length: 291

Algumas informações são importantes reparar:

A primeira linha do INVITE chega com o mesmo IP do *user agent server* (UAS). Assim o UAS pode ter a certeza de que esse INVITE é para ele mesmo tratar. Além disso, os campos *from* e *to* não têm seus valores principais alterados. Eles permanecem assim até o final dessa transação de *request* de chamada.

Quando essa mensagem passou pelo *proxy*, ele acrescentou mais um campo *Via* no cabeçalho da mensagem. Os campos *Via* costuram a rota da mensagem, na direção remetente para destinatário. Eles formam uma pilha, e são desempilhados quando uma resposta vem no sentido contrário, como será visto mais adiante. A utilidade deles é mostrar por qual caminho deverá passar uma resposta a essa mensagem SIP INVITE. O último nodo por onde passou a mensagem foi o *proxy* e então ele coloca o último campo *Via* com a localização dele mesmo. O *script* de configuração do *proxy* está programado para que, ao usar um endereço num campo *Via*, seja usado o nome do domínio, se a mensagem está sendo reencaminhada a um *peer* numa rede remota. Isso faz sentido, já que o *agent* SIP do usuário 7000 não poderia utilizar um campo *Via* com um IP privado de 8000.

O campo “[Contact: <sip:8000@192.168.0.102:47140;transport=TCP>](#)” permanece com o mesmo endereçamento. Então não pode ser usado pelo UAS 7000 para dar uma resposta ao INVITE. Nesse caso, o único campo que tem informação útil ao UAS, na hora de enviar uma resposta é mesmo o campo “[Via: SIP/2.0/TCP icchw.acmeddns.com.br:5060...](#)”. Não há outro campo no SIP INVITE que indique ao UAS onde está o *proxy* para onde a resposta deve ser enviada.

Abaixo está o SDP “corrigido” pelo *proxy*, sendo que agora o campo “o” contém um IP que realmente indica para onde o usuário 7000 poderá enviar pacote de RTP a fim de alcançar o usuário 8000. Esse é o IP 131.221.240.71. Como o usuário 8000 está debaixo do mesmo roteador onde está ligado o *proxy*, tanto *proxy* quanto *softphone* do usuário 8000 apresentam o mesmo IP público, do ponto de vista do *peer* 7000. Então, o *proxy* pode mesmo corrigir o IP no SDP, escrevendo lá o seu próprio IP público.

```
v=0
o=Z 0 0 IN IP4 131.221.240.71
s=Z
c=IN IP4 131.221.240.71
t=0 0
m=audio 8000 RTP/AVP 3 110 8 0 98 101
a=rtpmap:110 speex/8000
a=rtpmap:98 iLBC/8000
a=fmtp:98 mode=20
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=sendrecv
a=oldip:192.168.0.102
a=oldcip:192.168.0.102
```

Assim que o UAS 7000 recebe o SIP INVITE, ele imediatamente diz ao *proxy* que já está tentando interpretar e manipular tal mensagem. Ele faz isso enviando um SIP 100 Trying ao *proxy*, como visto na mensagem abaixo:

SIP/2.0 100 Trying
Via: SIP/2.0/TCP icchw.acmeddns.com.br:5060;received=131.221.240.71;branch=z9hG4bK7d9a.ca701d03.0;i=9

```

Via: SIP/2.0/TCP 192.168.0.102:47140;rport=42432;received=192.168.0.102;bran
ch=z9hG4bK-d8754z-725b562ed2aabadd-1---d8754z-
Record-Route: <sip:icchw.acmeddns.com.br;transport=tcp;lr;did=3a1.f41290b1>
Call-ID: NGYyZmYwZDdjYjY0MGViMmRkZDI5MTk2OTM4MGJmNDc.
From: "8000-PC" <sip:8000@192.168.0.103>;tag=18bb061a
To: <sip:7000@192.168.0.103>
CSeq: 2 INVITE
Content-Length: 0

```

Essa mensagem de Trying é enviada para icchw.acmeddns.com.br:5060, que implica o SIP Proxy. É interessante notar que o UAS 7000 acrescenta “`received=131.221.240.71`” no campo *Via top most*, o que indica que tal agente SIP recebeu a mensagem do proxy vinda do IP 131.221.240.71, que é realmente o IP público do proxy nesse exemplo. O SIP proxy também manda uma mensagem de SIP Trying para o UAC imediatamente após receber o SIP INVITE, para que o UAC saiba que o proxy já está tratando a requisição. Veja tal mensagem abaixo:

```

SIP/2.0 100 Giving a try
Via: SIP/2.0/TCP 192.168.0.102:47140;received=192.168.0.102;rport=42432;bran
ch=z9hG4bK-d8754z-725b562ed2aabadd-1---d8754z-
To: <sip:7000@192.168.0.103;transport=TCP>
From: "8000-PC"<sip:8000@192.168.0.103;transport=TCP>;tag=18bb061a
Call-ID: NGYyZmYwZDdjYjY0MGViMmRkZDI5MTk2OTM4MGJmNDc.
CSeq: 2 INVITE
Server: OpenSIPS (2.2.1 (arm/linux))
Content-Length: 0

```

A mensagem acima é enviada à porta 42432 do IP 192.168.0.102. Essa é a porta vista pelo proxy, quando ele recebeu o SIP INVITE do UAC 8000. A informação de *received* com tal IP e porta segue acrescentada novamente no campo *Via* do cabeçalho.

Em seguida, o UAS 7000 envia uma mensagem ao proxy dizendo que o *softphone* respectivo já está “*ringando*” (soando o ring). Para tal, a seguinte mensagem é enviada do UAS 7000 ao proxy:

```

SIP/2.0 180 Ringing
Via: SIP/2.0/TCP icchw.acmeddns.com.br:5060;received=131.221.240.71;bran
ch=z9hG4bK7d9a.ca701d03.0;i=9
Via: SIP/2.0/TCP 192.168.0.102:47140;rport=42432;received=192.168.0.102;bran
ch=z9hG4bK-d8754z-725b562ed2aabadd-1---d8754z-
Record-Route: <sip:icchw.acmeddns.com.br;transport=tcp;lr;did=3a1.f41290b1>
Call-ID: NGYyZmYwZDdjYjY0MGViMmRkZDI5MTk2OTM4MGJmNDc.
From: "8000-PC" <sip:8000@192.168.0.103>;tag=18bb061a
To: <sip:7000@192.168.0.103>;tag=aa67abed1e4e409b83cc77ba2f2d4f14
CSeq: 2 INVITE
Contact: "7000-notebook" <sip:7000@10.0.216.12:62581;transport=TCP;ob>;+sip.ice
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Content-Length: 0

```

A mensagem acima é recebida no proxy e repassada ao UAC 8000. Essa mensagem contém o contato do UAS, no campo *Contact*. Veja como fica a mensagem SIP 180 Ringing recebida pelo UAC:

```

SIP/2.0 180 Ringing
Via: SIP/2.0/TCP 192.168.0.102:47140;rport=42432;received=192.168.0.102;bran
ch=z9hG4bK-d8754z-725b562ed2aabadd-1---d8754z-
Record-Route: <sip:icchw.acmeddns.com.br;transport=tcp;lr;did=3a1.f41290b1>
Call-ID: NGYyZmYwZDdjYjY0MGViMmRkZDI5MTk2OTM4MGJmNDc.
From: "8000-PC" <sip:8000@192.168.0.103>;tag=18bb061a
To: <sip:7000@192.168.0.103>;tag=aa67abed1e4e409b83cc77ba2f2d4f14
CSeq: 2 INVITE
Contact: "7000-notebook" <sip:7000@10.0.216.12:62581;transport=TCP;ob>;+sip.ice
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Content-Length: 0

```

O campo *Via top most* é desempilhado no proxy, antes da mensagem ser repassada ao UAC 8000.

Mais adiante no tempo, o usuário 7000 atende a chamada no seu softphone Microsip e o UAS respectivo gera então um SIP 200 Ok e envia para o outro peer. Mas, antes a mensagem passa pelo SIP Proxy também. Veja abaixo tal mensagem que sai do UAS 7000:

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP icchw.acmeddns.com.br:5060;received=131.221.240.71;branch=z9hG4bK7d9a.ca701d03.0;i=9
Via: SIP/2.0/TCP 192.168.0.102:47140;rport=42432;received=192.168.0.102;branch=z9hG4bK-d8754z-725b562ed2aabadd-1---d8754z-
Record-Route: <sip:icchw.acmeddns.com.br;transport=tcp;lr;did=3a1.f41290b1>
Call-ID: NGYyZmYwZDdjYjY0MGViMmRkZDI5MTk2OTM4MGJmNDc.
From: "8000-PC" <sip:8000@192.168.0.103>;tag=18bb061a
To: <sip:7000@192.168.0.103>;tag=aa67abed1e4e409b83cc77ba2f2d4f14
CSeq: 2 INVITE
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Contact: "7000-notebook" <sip:7000@10.0.216.12:62581;transport=TCP;ob>;+sip.ice
Supported: replaces, 100rel, timer, norefersub
Content-Type: application/sdp
Content-Length: 278
```

Com visto acima, a mensagem é enviada para icchw.acmeddns.com.br na porta 5060, que é onde está o SIP proxy escutando mensagens SIP. Reparar que os IPs nos campos From e To permanecem inalterados durante toda a transação de inicialização de sessão. O SDP dessa mensagem contem 278 *characteres*. E o campo “o” já indica o IP correto para onde os pacotes de *media* (RTP) deverão ser enviados ao usuário 7000, já que esse IP se comporta como um IP público do *softphone* com *user* 7000.

```
v=0
o=- 3679830271 3679830272 IN IP4 10.0.216.12
s=pjmedia
b=AS:84
t=0 0
a=X-nat:0
m=audio 63763 RTP/AVP 110 101
c=IN IP4 10.0.216.12
b=TIAS:64000
a=rtpcp:63765 IN IP4 10.0.216.12
a=sendrecv
a=rtpmap:110 speex/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-16
```

Essa mensagem acima é repassada ao UAC 8000, da seguinte forma:

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP 192.168.0.102:47140;rport=42432;received=192.168.0.102;branch=z9hG4bK-d8754z-725b562ed2aabadd-1---d8754z-
Record-Route: <sip:icchw.acmeddns.com.br;transport=tcp;lr;did=3a1.f41290b1>
Call-ID: NGYyZmYwZDdjYjY0MGViMmRkZDI5MTk2OTM4MGJmNDc.
From: "8000-PC" <sip:8000@192.168.0.103>;tag=18bb061a
To: <sip:7000@192.168.0.103>;tag=aa67abed1e4e409b83cc77ba2f2d4f14
CSeq: 2 INVITE
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Contact: "7000-notebook" <sip:7000@10.0.216.12:62581;transport=TCP;ob>;+sip.ice
Supported: replaces, 100rel, timer, norefersub
Content-Type: application/sdp
Content-Length: 278

v=0
o=- 3679830271 3679830272 IN IP4 10.0.216.12
s=pjmedia
b=AS:84
t=0 0
a=X-nat:0
m=audio 63763 RTP/AVP 110 101
c=IN IP4 10.0.216.12
b=TIAS:64000
a=rtpcp:63765 IN IP4 10.0.216.12
a=sendrecv
a=rtpmap:110 speex/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-16
```

Mais uma vez, o campo *Via top most* é desempilhado antes de ser repassado ao UAC 8000. Nenhuma correção é necessária no IP declarado no SDP. Porque o IP do *device* do user 7000 é público do ponto de vista do proxy.

Quando o UAC recebe uma mensagem SIP OK, ele deve enviar um SIP ACK ao UAS. Caso contrário, o SIP UAS ainda enviará outras mensagens SIP OK e depois cancelará o envio de mensagens, sendo que assim a sessão não será estabelecida. Reparar que essa mensagem ACK passa obrigatoriamente pelo proxy no local 131.221.240.71, devido ao fato que o cabeçalho em mensagens SIP anteriores continha o campo Record-Route. O campo *Contact* na mensagem acima é usado para endereçar o SIP ACK. Veja isso abaixo na primeira linha da mensagem SIP ACK. Aqui seria interessante um aluno dessa disciplina fazer um estudo sobre o cabeçalho *Record-Route* e explicá-lo durante apresentação oral. Explicar, por exemplo, o que ocorre se ele está presente, em termos de endereçamento de respostas SIP.

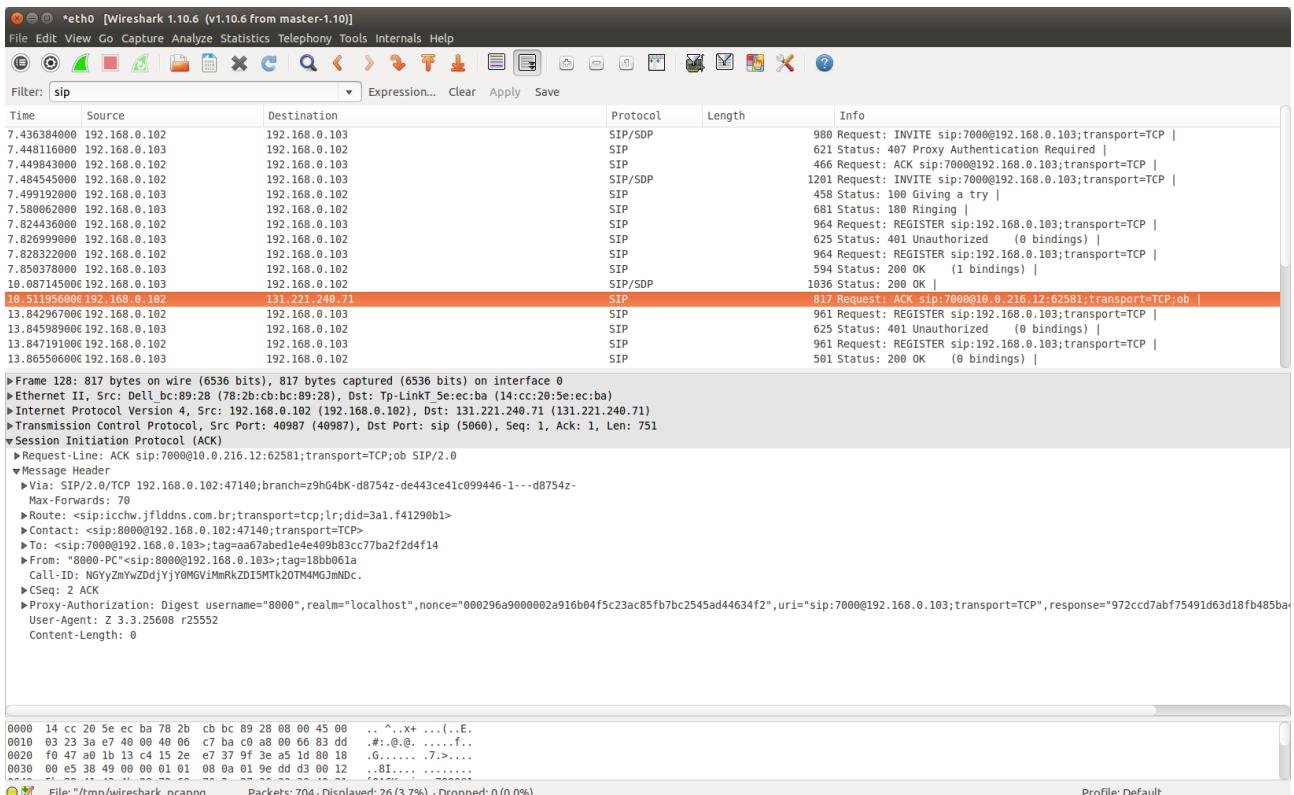


Figura 40 Fonte: projeto do ICC em 2016.

```

ACK sip:7000@10.0.216.12:62581;transport=TCP;ob SIP/2.0
Via: SIP/2.0/TCP 192.168.0.102:47140;branch=z9hG4bK-d8754z-de443ce41c099446-1---d8754z-
Max-Forwards: 70
Route: <sip:icchw.acmeddns.com.br;transport=tcp;lr;did=3a1.f41290b1>
Contact: <sip:8000@192.168.0.102:47140;transport=TCP>
To: <sip:7000@192.168.0.103>;tag=aa67abed1e4e409b83cc77ba2f2d4f14
From: "8000-PC"<sip:8000@192.168.0.103>;tag=18bb061a
Call-ID: NGYyZmYwZDdjYjY0MGViMmRkZDI5MTk2OTM4MGJmNDc.
CSeq: 2 ACK
Proxy-Authorization: Digest username="8000",realm="localhost",nonce="000296a9000002a916b04f5c23ac85fb7bc2545ad44634f2",uri="sip:7000@192.168.0.103;transport=TCP",response="972ccd7abf75491d63d18fb485ba"
User-Agent: Z 3.3.25608 r25552
Content-Length: 0

```

A mensagem acima é enviada do UAC 8000 para o UAS 7000. Mas, ela tem que passar pelo SIP Proxy obrigatoriamente. Nesse caso, mesmo que a mensagem tenha a linha “[ACK](#) sip:7000@10.0.216.12:62581...”, que implica em enviar para o IP 10.0.216.12, ela é enviada para 131.221.240.71, como visto na imagem do Wireshark acima. Isso é devido mesmo à presença do campo Record-Route no cabeçalho da mensagem.

Enquanto os campos *Via*, que costuram a rota de uma requisição SIP, mostram por onde uma mensagem SIP de resposta deve retornar (*sip response*), o *Record-Route* é usado tanto por mensagem num sentido, quanto no outro (*sip requests*), obrigando todas a passarem pelo endereço indicado por ele. Os SIP Proxies são os responsáveis em acrescentar esses campos *Record-Route* no cabeçalho das mensagens SIP.

Finalmente, quando o UAS recebe esse SIP ACK, a sessão estará estabelecida e daí em diante o envio de pacotes RTP poderá fluir.

```
ACK sip:7000@10.0.216.12:62581;transport=TCP;ob SIP/2.0
Via: SIP/2.0/TCP 192.168.0.103:5060;branch=z9hG4bK7d9a.ca701d03.2;i=d
Via: SIP/2.0/TCP 192.168.0.102:47140;rport=40987;received=131.221.240.71;branch=z9hG4bK-d8754z-de443ce41c099446-1---d8754z-
Max-Forwards: 70
Contact: <sip:8000@131.221.240.71:40987;transport=TCP>
To: <sip:7000@192.168.0.103>;tag=aa67abed1e4e409b83cc77ba2f2d4f14
From: "8000-PC" <sip:8000@192.168.0.103>;tag=18bb061a
Call-ID: NGYyZmYwZDdjYjY0MGViMmRkZDI5MTk2OTM4MGJmNDc.
CSeq: 2 ACK
Proxy-Authorization: Digest
username="8000",realm="localhost",nonce="000296a9000002a916b04f5c23ac85fb7bc2545ad44634f2",uri="sip:7000@192.168.0.10
3;transport=TCP",response="972ccd7abf75491d63d18fb485ba4d81",algorithm=MD5
User-Agent: Z 3.3.25608 r25552
Content-Length: 0
```

A mensagem acima mostra como o UAS 7000 recebe o SIP ACK. Não é mais necessário manter o campo *Record-Route* e o campo *Contact* é atualizado no proxy, antes de enviar a mensagem adiante.

Mais adiante o usuário 7000 resolve desligar e então o UAC 7000 respectivo envia uma mensagem de SIP BYE para o UAS 8000. Veja abaixo como essa mensagem sai do UAS:

```
BYE sip:8000@192.168.0.102:47140;transport=TCP SIP/2.0
Via: SIP/2.0/TCP 10.0.216.12:62581;rport;branch=z9hG4bKPjb4b052a4ddd04e3c8ed5da7ad7b94985;alias
Max-Forwards: 70
From: <sip:7000@192.168.0.103>;tag=aa67abed1e4e409b83cc77ba2f2d4f14
To: "8000-PC" <sip:8000@192.168.0.103>;tag=18bb061a
Call-ID: NGYyZmYwZDdjYjY0MGViMmRkZDI5MTk2OTM4MGJmNDc.
CSeq: 25760 BYE
Route: <sip:icchhw.acmeddns.com.br;transport=tcp;lr;did=3a1.f41290b1>
User-Agent: MicroSIP/3.11.0
Content-Length: 0
```

Importante reparar que a mensagem BYE inicia outra transação SIP: desligamento de sessão. Isso faz com que o remetente dessa mensagem passe a ser o *user agent client* e o seu *peer* o *agent user server*. Nesse caso, o UAC 7000 acrescenta o campo *Via* na mensagem, e o proxy adicionará outro. Como aqui temos uma nova *transaction*, os campos From e To invertem seus valores de endereçamentos.

Veja abaixo como essa mensagem chega no UAS 8000:

```
BYE sip:8000@192.168.0.102:47140;transport=TCP SIP/2.0
Via: SIP/2.0/TCP 192.168.0.103:5060;branch=z9hG4bK12e6.f24b6563.0;i=c
Via: SIP/2.0/TCP 10.0.216.12:62581;received=10.0.216.12;rport=62581;branch=z9hG4bKPjb4b052a4ddd04e3c8ed5da7ad7b94985;alias
```

Max-Forwards: 70
From: <sip:7000@192.168.0.103>;tag=aa67abed1e4e409b83cc77ba2f2d4f14
To: "8000-PC" <sip:8000@192.168.0.103>;tag=18bb061a
Call-ID: NGYyZmYwZDdjYjY0MGViMmRkZDI5MTk2OTM4MGJmNDc.
CSeq: 25760 BYE
User-Agent: MicroSIP/3.11.0
Content-Length: 0

Quando o UAS 8000 recebe esse BYE, ele envia um SIP OK para o UAC 7000 (usando o campo *Via top most* da mensagem acima, como endereço de destino), da seguinte forma:

SIP/2.0 200 OK
Via: SIP/2.0/TCP 192.168.0.103:5060;branch=z9hG4bK12e6.f24b6563.0;i=c
Via: SIP/2.0/TCP 10.0.216.12:62581;received=10.0.216.12;rport=62581;branch=z9hG4bKPjb4b052a4ddd04e3c8ed5da7ad7b94985;alias
Contact: <sip:8000@192.168.0.102:47140;transport=TCP>
To: "8000-PC" <sip:8000@192.168.0.103>;tag=18bb061a
From: <sip:7000@192.168.0.103>;tag=aa67abed1e4e409b83cc77ba2f2d4f14
Call-ID: NGYyZmYwZDdjYjY0MGViMmRkZDI5MTk2OTM4MGJmNDc.
CSeq: 25760 BYE
User-Agent: Z 3.3.25608 r25552
Content-Length: 0

Essa mensagem vai direto para o nodo 192.168.0.103, como visto no campo *Via top most*. Mas esse é exatamente o nodo onde está o SIP Proxy. Como o UAS 8000 está na mesma rede que o proxy, ele pode utilizar o IP privado do proxy. A mensagem abaixo mostra como esse SIP OK chega finalmente no UAC 7000 após passar pelo proxy.

SIP/2.0 200 OK
Via: SIP/2.0/TCP 10.0.216.12:62581;received=10.0.216.12;rport=62581;branch=z9hG4bKPjb4b052a4ddd04e3c8ed5da7ad7b94985;alias
Contact: <sip:8000@192.168.0.102:47140;transport=TCP>
To: "8000-PC" <sip:8000@192.168.0.103>;tag=18bb061a
From: <sip:7000@192.168.0.103>;tag=aa67abed1e4e409b83cc77ba2f2d4f14
Call-ID: NGYyZmYwZDdjYjY0MGViMmRkZDI5MTk2OTM4MGJmNDc.
CSeq: 25760 BYE
User-Agent: Z 3.3.25608 r25552
Content-Length: 0

Após o UAC 7000 receber essa mensagem acima, a sessão SIP está definitivamente encerrada.

Esses foram 2 exemplos de mensagens SIP para servir de apoio inicial a quem vá pesquisar mais sobre SIP, antes de fazer uma apresentação oral em sala de aula nessa disciplina. O primeiro exemplo contou com 2 peers na mesma rede e o segundo exemplo contou com um peer na rede local e outro em rede remota. Nos 2 exemplos todos os peers estão no mesmo domínio SIP. Para peers em domínios SIP diferentes, tem-se que usar uma outra entidade chamada SIP Redirect, mas isso está fora do escopo desse material. Entretanto, caso algum aluno queira explicar sobre o SIP Redirect, numa apresentação, as informações serão de grande valia ao aprendizado mais detalhado ainda sobre SIP.

Pilhas SIP

Ao desenvolver software que precise usar o protocolo SIP, pode-se fazer uso de pilhas SIP já disponíveis no mercado. Por exemplo, alguém que vá desenvolver um *softphone*, talvez atendendo requisitos de algum projeto particular de empresa da área de vigilância (segurança residencial) por câmeras IP, poderá usar alguma biblioteca que já implemente a pilha SIP. Nesse caso, existe a

Liblinphone, que apresenta formas de usar a pilha SIP já implementada por ela. O *softphone* Linphone usa a Liblinphone, que é escrita em C. Outras pilhas SIP estão disponíveis no mercado. Por exemplo, existe a PJSIP, que é usada pelo *softphone* Microsip. Estudar a liblinphone e apresentar formas de utilizá-la em projetos de software resulta em um trabalho muito valioso para ser demonstrado oralmente, em locais como o Inatel Competence Center. Por exemplo, como usar essa pilha para atravessar NATs? Tem-se que usar ICE e STUN? Como fazer isso?

Outro assunto interessante que poderia ser visto nessa disciplina, relacionado com SIP, é o Asterisk. O Asterisk é um PABx virtual, não um SIP proxy. Com o Asterisk é possível conectar ramais VoIP e programar planos de discagens para tais ramais. Não é possível interagir com as mensagens SIP e modificá-las, como será visto no capítulo seguinte, mas é possível criar planos que surtem efeito em ligações a serem realizadas, por exemplo. Um ramal A que deseja ligar para um ramal B pode ter a ligação desviada para um ramal C, por exemplo, se o plano de discagem estiver configurado para isso. Os planos de discagens podem ser feitos com a linguagem específica, que é interpretada pelo Asterisk. O Asterisk é o PABx virtual muito usado no mundo e tem grande 'poder' de manipulação de ligações. Em 2007 o ICC desenvolveu uma URA (unidade de resposta audível) 100% suportada pelo Asterisk, para a empresa Leucotron. As mensagens trocadas entre os ramais e o Asterisk são em SIP. Além da possibilidade de escrever os planos de discagem em linguagem proprietária do Asterisk, em 2007 era possível escrever sistemas em Java, com biblioteca do Asterisk, de tal forma que o programa gerado podia criar planos de discagens úteis ao Asterisk. Uma pesquisa sobre o Asterisk atualmente, mostrando suas capacidades atuais e integrações com outras linguagens de programação, seria um bom trabalho a ser apresentado por aluno dessa disciplina. O Asterisk realmente vale a pena ser aprendido, devido às várias possibilidades de controlar ligações VoIP com ele.

Finalmente, um assunto muito importante com a criação de sessões, para fins de comunicação como no uso de VoIP, é a travessia de NATs. O documento [126] dá conta desse assunto e é uma boa referência de estudo. Esse documento poderia ser estudado por um aluno dessa disciplina e o conteúdo poderia ser todo apresentado oralmente. Isso seria um ótimo seminário!

Capítulo IV - OpenSIPS

Em 2016-2018 o Inatel Competence Center desenvolveu um projeto em telecomunicações que utilizou um SIP *proxy*. Isso foi necessário porque tal projeto deveria conter um *sip server* e clientes (*peers*) conectados no mesmo. Os clientes podem ser *smartphones* com *softphones*, SIP *phones*, *softphones* em PCs, etc. Vários dos requisitos deste projeto, aplicáveis como regras sobre as ligações telefônicas feitas pelos *peers* podiam ser implementadas facilmente num SIP *proxy*, principalmente se fosse o OpenSIPS.

No projeto, o software usado pelos usuários, para telecomunicação, como os *softphones*, utiliza trocas de mensagens com o protocolo SIP. O protocolo SIP, usado para a sinalização e estabelecimento de sessão entre os *peers*, é baseado em texto, como o HTTP e isso facilita executar modificações no conteúdo de suas mensagens. De fato, para o projeto foi muito necessário executar vários tipos de modificações ou manipulações com as mensagens do SIP. Nesse caso, seria inviável usar um PABx virtual, como o Asterisk para o gerenciamento das mensagens SIP. Somente um *proxy SIP* permite o controle adequado sobre as mensagens como foi necessário no projeto. Um ótimo SIP Proxy disponível no mercado de telecomunicações se chama **OpenSIPS**. Esse SIP *proxy* é de código aberto e muito fácil de usar (depois que é bem entendido e isso requer boa quantidade de estudo sobre ele). A documentação sobre o mesmo é excelente e existe uma lista de discussões muito bem movimentada a respeito do mesmo.

Todas as mensagens SIP geradas pelos *softphones* interagindo entre si, através do projeto, passam pelo OpenSIPS. E esse *proxy* contém um *script* no qual pode-se programar ações a serem realizadas sobre o SIP. As ações são tomadas pelo OpenSIPS então, que passa a obedecer o que estiver programado em tal *script*. Dessa forma, dizemos que existe um *script* para programar o que o OpenSIPS deve fazer com o SIP. Tal *script* é tão flexível que ele pode conter funções prontas para manipular partes específicas das mensagens SIP e também qualquer *string* contida nas mesmas. Para a criação perfeita de tal *script*, faz-se necessário o estudo da documentação de certos módulos usados pelo OpenSIPS.

Para que o OpenSIPS consiga interagir com as mensagens SIP, da forma como desejamos e programamos, o mesmo faz uso de certos módulos auxiliares. Ou seja, o próprio OpenSIPS contém módulos específicos para funções específicas e tais módulos podem ser incluídos (utilizados) no *script* de configuração, sob demanda. Por exemplo, se é necessário manipular dados de localização de um usuário que se registra nesse SIP Proxy, então pode-se usar um módulo específico para manipular informações de *user location*. Tal módulo se chama **usrloc**. Um módulo que não seja útil para o *script* atual não precisa ser incluído pelo mesmo e então não ocupará memória desnecessariamente.

Resumo sobre o OpenSIPS

“**OpenSIPS** is an Open Source SIP proxy/server for voice, video, IM, presence and any other SIP extensions.” Para um entendimento prévio sobre o que é o protocolo SIP, tem-se que ler o capítulo anterior nessa apostila.

O web site sobre o OpenSIPS está aqui: <http://www.opensips.org/> .

Nesse site pode ser encontrada a documentação desse *proxy*, organizada da seguinte forma:

Manuais para desenvolvedores com o OpenSIPS. Esses manuais são divididos em :

- Documentos de instalação.
- Documentos de configuração (mostram como configurar características do *proxy*. Ex: uso de quantidade de memória, etc.)
- Documentos de **scripting**. Esses documentos mostram como escrever um arquivo .cfg que configura como o OpenSIPS deve agir sobre as mensagens SIP. Ex: documentos para uso correto de variáveis, funções e operadores.

Dentre os documentos de *scripting* existem os Modules documentation. Esses são os documentos mais frequentemente acessados pelos desenvolvedores que desejam fazer uso do OpenSIPS. É aí exatamente onde estão as informações sobre como incluir o uso dos módulos auxiliares no *script* do OpenSIPS. Ex: documento para o módulo **usrloc**.

- Documentos de Interfaces. Esses documentos explicam como um sistema externo ao OpenSIPS pode mandar comandos a ele. Ou seja, para alguns casos específicos de uso do *proxy*, é possível invocar funções nele a partir de um terceiro software, independente do conteúdo do *script* de configuração, por exemplo.

Todo este conteúdo de documentação encontra-se aqui:

<https://www.opensips.org/Documentation/Manuals>

Resumidamente, o OpenSIPS é um software formado por vários módulos capazes de manipularem mensagens SIP de várias formas. Cada módulo tem uma função diferente, em termos do que pode ser feito com mensagens SIP. Ao usar o OpenSIPS, pode-se escolher quais módulos serão usados, dependendo do que for necessário fazer com as mensagens. A escolha de qual módulo do openSIPS usar é definida em arquivo de configuração. Em tal arquivo, além de declarar quais módulos estarão em uso, pode-se fazer chamadas de funções constantes nos mesmos módulos. A sintaxe desse arquivo de configuração permite programar lógica *booleana*, ao mesmo tempo que as funções são chamadas. Isso permite uma enorme flexibilidade sobre o que fazer com as mensagens SIP. Para saber bem quais módulo usar e o que exatamente pode ser feito com eles, há a documentação do OpenSIPS, que é muito rica e não deixa a desejar. Por exemplo, a documentação sobre módulos mostra o que pode ser feito e é a mais acessada por quem estiver programando o *script* (arquivo de configuração) do OpenSIPS. Quando o OpenSIPS trabalha como um SIP *proxy* ele recebe todas as

mensagens SIP trocadas entre os *peers* de seu domínio.

Como o OpenSIPS lida com mensagens SIP de *request* e *responses*, ele então lida com mensagens de criação de chamada, como o SIP INVITE, e outras de respostas, como a SIP OK. O *script* do OpenSIPS é dividido em blocos bem definidos, para o tratamento de mensagens de *request* e *response*. Por exemplo, existem os seguintes blocos no *script*:

```
##### Global Parameters #####
```

Nesse bloco declara-se o uso de algumas variáveis globais ao script. Ex:

```
listen=tcp:eth0:5060  
listen= tls:eth0:5061  
listen= tls:eth0:5071
```

```
##### Modules Section #####
```

Nesse bloco, que deve ser obrigatoriamente o inicial, declara-se os módulos auxiliares a serem carregados na execução do OpenSIPS. Ex:

```
#set module path  
mpath="/usr/local/opensips_proxy//lib64/opensips/modules/" (onde estão instalados os módulos )  
loadmodule "exec.so" (módulo com capacidade de executar um executável fora do OpenSIPS)  
loadmodule "cfgutils.so"  
loadmodule "signaling.so"
```

```
##### Router Section #####
```

O OpenSIPS contém a capacidade de rotear as mensagens SIP, ou seja, encaminhá-las a quem é de interesse, como um INVITE que vem do *caller* e deve ser repassado ao *callee*. Nesse roteamento, uma mensagem pode ser manipulada antes de ser encaminhada (antes do *relay*). Existe um bloco genérico, no arquivo de configuração do OpenSIPS, que é o espaço para se manipular qualquer tipo de mensagem, como um INVITE ou um CANCEL. Esse bloco se chama *route*.

```
route{  
}
```

A cada momento que o código dentro desse bloco *route* é executado, isso significa que o OpenSIPS está manipulando uma mensagem SIP. Essa mensagem pode ser de vários tipos. O que fazer com a

mensagem depende do tipo dela e do que se quer fazer. Ex: depende se a mensagem é um INVITE, um REGISTER, um CANCEL, etc. E depende de requisitos do projeto pelo qual algum sistema de comunicação esteja sendo feito com SIP. Para se saber qual é a mensagem sendo executada no momento (mensagem recebida no *proxy* no momento corrente), tem-se que usar métodos como : “`is_method("REGISTER")`”, por exemplo.

Quando um *relay* de mensagem SIP está prestes a ser executado (despacho da mensagem ao próximo nodo), o OpenSIPS muda a lógica de sua execução para o seguinte bloco:

```
route[relay] {  
}
```

No bloco de *relay* (visto acima), no caso particular de requisitos de um projeto, pode haver alguns sub blocos auxiliares às mensagens do tipo INVITE. Esses blocos auxiliares lidam com respostas ao INVITE, com casos de falhas no estabelecimento da chamada e com a mensagem INVITE em si. Veja o código abaixo de exemplo para isso:

```
# for INVITES enable some additional helper routes  
if (is_method("INVITE")) {  
    lookup("location");  
    t_on_branch("per_branch_ops");  
    t_on_reply("handle_nat");  
    t_on_failure("missed_call");  
}
```

Cada vez que o OpenSIPS recebe um SIP INVITE é criado um novo *branch*, e a partir daí o que ainda resta a ser feito está no sub bloco :

```
branch_route[per_branch_ops] {  
    // uma mensagem INVITE recebida implica em executar esse escopo.  
    // esse escopo pode conter código para manipular o SDP.  
}
```

Esse sub bloco acima pode ter a responsabilidade de alterar o conteúdo do SDP do INVITE respectivo, por exemplo. Ou seja, o *proxy* recebe um SIP INVITE com SDP e o código programado para fazer qualquer alteração no SDP pode estar contido no sub bloco mostrado acima. Por exemplo, a alteração pode envolver troca de IP, o que estará relacionado com endereçamento mais correto para o envio de mídia de áudio.

Para cada resposta recebida, por exemplo de um INVITE, como um SIP OK, o sub bloco abaixo é executado:

```
onreply_route[handle_nat] {  
}
```

No caso específico de um dado projeto, esse sub bloco acima pode ter a responsabilidade de alterar

o conteúdo do SDP do SIP OK, por exemplo. A alteração também pode envolver troca de IP, o que está relacionado com endereçamento mais correto para o envio de mídia de áudio.

Esses sub-blocos mostrados acima contêm nomes: *handle_nat* e *per_branch_ops*. Esses nomes são apenas convenções. Poderiam ser quaisquer nomes.

Caso ocorra uma mensagem de falha ao estabelecer uma sessão SIP, o bloco abaixo terá a chance de ser executado:

```
failure_route[missed_call] {  
}
```

No bloco acima temos a chance de tratar as mensagens SIP de código 400 a 499. Ex: 408: pessoa chamada não encontrada. Ocorre quando a pessoa está registrada na tabela *subscriber* do banco de dados do OpenSIPS, mas ainda não efetuou um login nesse *proxy*. O código 420 significa que o número não existe. Ou seja, o OpenSIPS não contém tal dado em sua tabela *subscriber* e portanto não será responsável em fazer o *relay*. O Registrar desse *proxy* é contido no banco de dados e é a tabela *subscriber* que contém as definições de logins e senhas de quem pode se tornar online junto a esse servidor.

Dentro de cada bloco estarão as chamadas às funções dos módulos em uso. Ou seja, estarão as chamadas às funções que farão trabalho sobre as mensagens SIP. Tais funções podem requerer dados do banco de dados. Para tal, tem-se que executar *queries SQL* no *script*, para a obtenção dos dados. Uma *query SQL* pode ser executada diretamente do *script*, com o uso da função **avp_db_query()**.

Ex:

```
avp_db_query("select name from subscriber where id = '10'",  
"$avp(I_A)");
```

O código acima executa uma *query SQL* e obtém um nome. O valor obtido é transferido para o AVP *I_A*. Esse AVP pode então ser lido posteriormente. É importante saber que um AVP não é como uma variável. Um AVP pode acumular vários pares de atributo/valor. Funciona como uma pilha. Então, para evitar o empilhamento de valores no AVP, tem-se que atribuir NULL a ele sempre que desejável 'limpá-lo'. O módulo para tratamento com AVPs é o **avpops.so**. Esse módulo contém a função **avp_db_query()**.

Uma característica importante do OpenSIPS é a sua capacidade em lidar com situações de NATs entre *callers* e *callees*. Num dado projeto de um servidor SIP residencial, por exemplo para interconectar *smartphones* e um video portador, um usuário do terminal principal (servidor SIP) pode estar, por exemplo, em sua residência. Nesse caso, ele estará usando o terminal principal na mesma rede em casa, ou seja, na sua rede local. Ao mesmo tempo se outro usuário estiver com um *softphone* em outra rede (Ex: *smartphone* na Internet), ele estará numa rede remota. Com esses usuários conversando entre si, o usuário residencial estará atrás de um NAT do ponto de vista do

usuário na Internet, por exemplo. Então, o conteúdo do SDP das mensagens que partem da residência para o outro usuário precisa ter certos IPs alterados em seu texto. Felizmente, esse tipo de alteração pode ser feito no script do OpenSIPS, por exemplo com a função **fix_nated_SDP()**. Essa função é do módulo **nathelper**.

Contudo, o uso apenas do OpenSIPS não é suficiente para lidar com NATs. Com o uso do OpenSIPS, pode-se fazer correções nas mensagens SIP (cabeçalhos dessas mensagens) que atravessam NATs. Além disso, pode-se alterar algumas partes do SDP também. Mas, algumas informações do SDP precisam ser alteradas/construídas com a ajuda do protocolo ICE, por exemplo. Esse protocolo define no SDP linhas representando rotas candidatas pelas quais o *peer* correspondente poderá tentar enviar a sua *media*. A rota mais acima na lista de rotas prováveis é a que o *peer* deve tomar como mais provável de funcionar bem. O protocolo ICE troca mensagens com um servidor STUN, tanto a partir de um *peer*, quanto do outro. Com tal troca de mensagens, o ICE pode determinar portas e endereços a definir no SDP como rotas candidatas. Para que isso funcione, um servidor STUN (EX: *stunserver*) deve estar rodando em algum nodo público na rede. Como a questão de atravessar NATs durante o estabelecimento de sessões e envio de medias é de grande importância em telecomunicações e aplicativos móveis, uma pesquisa sobre ICE e STUN seria de grande valia para esse curso, mas uma apresentação oral em sala de aula, por aluno. Por exemplo, existem vários tipos de NATs e explicar cada um deles seria muito útil aos alunos dessa disciplina. Caso um projeto não usará ICE e STUN, pode-se fazer uso de módulos do OpenSIPS que facilitam a criação de um nodo *Media Relay*. Um nodo desse tipo é construído para ficar público na Internet e ser usado como auxílio na travessia de NATs. Nesse caso, a *media* enviada passa pelo *media relay*. Entretanto, caso seja desejado implementar *direct media* (media direta *peer to peer* sem passar num nodo terceiro e nem no proxy), pode ser usado o ICE e o STUN. Estudar o OpenSIPS e entender como escrever o seu *script* para fazer uso de *media relay* também seria um adicional apreciável para essa disciplina, o que poderia ser feito pelo aluno e posteriormente apresentado oralmente em sala de aula. As pesquisas sobre OpenSIPS podem ser bem facilitadas através de discussões via a lista de discussão respectiva, disponível publicamente.

Principais partes no arquivo de configuração do OpenSIPS

Como explicado antes, no início do arquivo de configuração do OpenSIPS tem-se que declarar os módulos a serem usados. Segue abaixo alguns dos módulos carregados especificamente sob requisitos de um projeto realizado pelo Inatel Competence Center em 2016-2018. Veja o resumo sobre a utilidade de cada módulo abaixo:

[loadmodule "exec.so"](#)

Nesse módulo estão as funções que podem ser usadas no *script* de configuração do OpenSIPS com o objetivo de executar arquivos binários que estão fora do *proxy*. Por exemplo, pode-se executar um programa feito em C. Ou seja, de dentro do *script* é possível invocar um binário fora do OpenSIPS.

[loadmodule "sl.so"](#)

Nesse módulo estão as funções que podem ser usadas no script de configuração do OpenSIPS com o objetivo de permitir o OpenSIPS agir como um *stateless UA server* e gerar respostas para requisições SIP sem manter qualquer estado. Ou seja, o *proxy* trabalha sem manter qualquer informação sobre o que tenha feito para mensagens SIP passadas. Isto é benéfico em muitos senários, nos quais é desejável economizar memória do servidor e escalar bem.

[loadmodule "domain.so"](#)

Nesse módulo estão as funções que podem ser usadas no *script* de configuração do OpenSIPS com o objetivo de deixar o OpenSIPS definir e manipular domínios. Cada SIP Proxy é responsável pelos seus assinantes, no sentido de saber localizá-los e contatá-los com mensagens SIP. Esse conjunto de responsabilidades define o domínio do *sip proxy*. Quando um assinante (*softphone* com usuário e senha já definidos) fica *online* no *sip proxy*, isso significa que tal assinante está conectado via um domínio específico.

[loadmodule "tm.so"](#)

Nesse módulo estão as funções que podem ser usadas no *script* de configuração do OpenSIPS com o objetivo de deixar o *proxy* lidar com as transações SIP. Esse módulo é indispensável.

[loadmodule "mi_fifo.so"](#)

Nesse módulo estão as funções que podem ser usadas no *script* de configuração do OpenSIPS com o objetivo de permitir o uso de arquivos texto a serem lidos pelo *proxy*. Tais arquivos podem conter comandos a serem executados no próprio *proxy*. Por exemplo, pode ser necessário usar tais arquivos para passar ao *proxy* os comandos de encerramentos de certas chamadas.

[loadmodule "db_sqlite.so"](#)

Nesse módulo estão as funções que podem ser usadas no *script* de configuração do OpenSIPS com o objetivo de permitir o *proxy* utilizar o banco de dados Sqlite.

[loadmodule "avpops.so"](#)

Nesse módulo estão as funções que podem ser usadas no *script* de configuração do OpenSIPS com o objetivo de permitir o uso de AVPs dentro de tal *script*.

[loadmodule "usrloc.so"](#)

Nesse módulo estão as funções que podem ser usadas no *script* de configuração do OpenSIPS com o objetivo de deixar o *proxy* lidar com informações de localização de usuários. Ex: o OpenSIPS pode extrair informações de localização do usuário diretamente das mensagens SIP e armazená-las na tabela **location** do banco de dados.

[loadmodule "registrar.so"](#)

Nesse módulo estão as funções que podem ser usadas no *script* de configuração do OpenSIPS com o objetivo de permitir o *proxy* conter e usar um SIP REGISTRAR. Na verdade aqui

o *proxy* se comporta também como um REGISTRAR. O REGISTRAR é a entidade que permite o armazenamento das informações de registro dos usuários do *proxy*. Portanto, esse REGISTRAR contem o banco de dados Sqlite com tabelas para **subscribers** e **locations**.

[loadmodule "dialog.so"](#)

Nesse módulo estão as funções que podem ser usadas no *script* de configuração do OpenSIPS com o objetivo de permitir o *proxy* registrar informações de diálogos em andamento. Cada vez que uma sessão é estabelecida entre *caller* e *callee* o *proxy* registra no banco de dados as informações desse diálogo. Quando a sessão é encerrada, as mesmas informações são imediatamente removidas do banco de dados. Assim, através da tabela de *dialogs* é possível saber quantas ligações em andamento existem sob responsabilidade do *proxy*.

[loadmodule "acc.so"](#)

Nesse módulo estão as funções que podem ser usadas no *script* de configuração do OpenSIPS com o objetivo de permitir o *proxy* fazer a contabilidade das ligações realizadas. Assim, o *proxy* registra no banco de dados os valores tais como número do chamador, número do chamado, momento da ligação, duração da mesma, etc. É possível registrar então as ligações atendidas e também as não atendidas.

[loadmodule "nathelper.so"](#)

Nesse módulo estão as funções que podem ser usadas no *script* de configuração do OpenSIPS com o objetivo de permitir o *proxy* lidar com as mensagens SIP, analisando certos dados nas mesmas, para determinar se um usuário (*softphone* já online) encontra-se atrás de um NAT, do ponto de vista do próprio *proxy*. Esse módulo contem funções tais como **fix_nated_SDP()**, etc. **Todas os módulos e funções que o OpenSIPS contem e que são auxiliares para determinar se um peer está atrás de um NAT poderiam ser bem estudados e entendidos para serem apresentados oralmente por aluno dessa disciplina. O aluno poderia explicar como usar os módulos respectivos, fazendo uso das funções dos mesmos. Isso deixaria claro em detalhes todas as possibilidades de determinar se um peer está atrás de um NAT.**

[loadmodule "proto_tcp.so"](#)

Nesse módulo estão as funções que podem ser usadas no *script* de configuração do OpenSIPS com o objetivo de permitir o *proxy* receber mensagens SIP transportadas via TCP.

[loadmodule "proto_tls.so"](#)

Nesse módulo estão as funções que podem ser usadas no *script* de configuração do OpenSIPS com o objetivo de permitir o *proxy* receber mensagens SIP transportadas via TCP e criptografadas com TLS. Quando as mensagens SIP são criptografadas com TLS é necessário usar certificados privados, públicos e autoridades certificadoras. Os certificados podem ser gerados com o OpenSSL e depois incluídos no arquivo de configuração do OpenSIPS. Se TLS é usado, o OpenSIPS poderá exigir que o *peer* que está entrando em contato com ele se identifique fornecendo o seu arquivo de certificado. Antes disso, no *peer* deverá ser instalado o arquivo de certificado correspondente. Além disso, o *peer* também poderá exigir a identificação do OpenSIPS, com

arquivo de certificado. Tudo isso garante que *peer* e OpenSIPS estarão se comunicando seguramente, sem um intruso no meio do caminho se passando por um deles.

Ainda no início do arquivo de configuração do OpenSIPS, tem-se que declarar, com as variáveis globais, quais são as interfaces de rede, IPs ou *hostnames* por onde esse *proxy* escutará o recebimento de mensagens SIP. Aqui pode-se declarar que o OpenSIPS irá escutar as interfaces de rede eth0 ou wlan0, por exemplo. Existe também a possibilidade de usar lo para *loop back*. Ou seja, para uma situação onde *proxy* e *softphone* cliente precisam executar no mesmo hardware numa situação tontamente sem link de rede, interface de rede, cabo de rede, número de IP, etc.

Importante: pode-se declarar tipos diferentes de interfaces, IPs, etc. Tudo ao mesmo tempo no mesmo arquivo. Entretanto, caso uma dessas interfaces não esteja fisicamente disponível, o OpenSIPS não executará. Ex:

```
listen=tcp:eth0:5060
listen=tlx:eth0:5061
listen=tlx:eth0:5071
listen=tcp:wlan0:5060
listen=tlx:wlan0:5061
listen=tlx:wlan0:5071
listen=tcp:lo:5060
listen=tlx:lo:5061
listen=tlx:lo:5071
```

Se qualquer uma das interfaces de rede declarada acima não estiver disponível de fato, o OpenSIPS não executará.

Pode-se declarar *listeners* para TCP, UDP e TLS ao mesmo tempo no mesmo arquivo de configuração do OpenSIPS.

Além desse conteúdo inicial de configuração, citado acima nesse capítulo, o resto do arquivo de configuração é feito obedecendo os requisitos do projeto de telecomunicação em questão. Como cada projeto tem seus requisitos particulares, o que é escrito a seguir no arquivo varia muito de projeto para projeto, não havendo uma regra fixa para isso. A única coisa que é considerada padrão são os blocos usados na configuração, como explicado anteriormente. Veja então na próxima seção como certos requisitos de projeto podem ser implementados com a ajuda do SIP proxy OpenSIPS.

Exemplos de requisitos de projetos que podem ser implementados com a ajuda do OpenSIPS

Abaixo estão explicadas algumas passagens vistas em arquivo de configuração do OpenSIPS, as quais implementam certos requisitos de um projeto. As passagens estão mostradas de forma isolada, com comentários sobre as mesmas.

Inicialmente é útil saber que as manipulações de mensagens SIP no OpenSIPS podem analisar de qual *peer* vêm tais mensagens, ou para qual *peer* elas devem ser encaminhadas. Ou seja, é possível saber qual é o usuário respectivo *caller* ou *calle*. Para tal a variável **\$rU** contém o *username* (*login*) na URI da requisição SIP INVITE em tratamento pelo proxy (usuário destinatário de uma chamada). Ou seja, o próprio proxy já fornece uma variável que pode ser usada no *script* e que sempre contém o *username* para quem vai a *sip request*. Muitas outras variáveis estão disponíveis para esse *script*. Para conhecer quais são todas as variáveis e que valores elas carregam, pode-se acessar a página [67].

Ex:

```
INVITE sip:crdphmacl\_SPnuV5xqtnSX@131.221.240.204:36137;transport=TCP;ob SIP/2.0
Record-Route: <sip:icchw.acmeddns.com.br;transport=tcp;lr;did=17.7b63b2d4>
Via: SIP/2.0/TCP icchw.acmeddns.com.br:5060;branch=z9hG4bKc555.156cbd61.0;i=2
Via: SIP/2.0/TCP 192.168.100.241:42253;received=192.168.100.241;rport=42253;branch=z9hG4bKPje1f91e8c-9634-4dfa-be28-4ec75946b6a3;alias
Max-Forwards: 70
From: "6002EXT" <sip:6002@192.168.100.177>;tag=7f16b000-632a-4dbc-b893-dccf47d253e7
To: sip:6000@192.168.100.177
Contact: "6002EXT" <sip:6002@192.168.100.241:42253;transport=TCP;ob>;+sip.ice
Call-ID: 69535656-663f-4949-b09a-f26d21419ffb
CSeq: 6748 INVITE
Route: <sip:6002@192.168.100.177;transport=tcp;lr>
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800
Min-SE: 90
Content-Type: application/sdp
Content-Length: 766
```

Quando o OpenSIPS recebe a mensagem acima, a variável **\$rU** conterá o valor [crdphmacl_SPnuV5xqtnSX](#) que é o identificador (*username/ login*) do usuário respectivo para onde a mensagem deve ser encaminhada, ou seja, quem deve ser chamado. O usuário pode ser uma pessoa real ou um sistema de software. Tanto faz.

A variável **\$td** contém o domínio na URI do cabeçalho “To” da mensagem SIP. Esse é o domínio no qual pertence a pessoa chamada. Pode ser um IP na própria rede local do OpenSIPS.

Ex:

```
INVITE sip:crdphmacl_SPnuV5xqtnSX@131.221.240.204:36137;transport=TCP;ob SIP/2.0
Record-Route: <sip:icchw.acmeddns.com.br;transport=tcp;lr;did=17.7b63b2d4>
Via: SIP/2.0/TCP icchw.acmeddns.com.br:5060;branch=z9hG4bKc555.156cbd61.0;i=2
Via: SIP/2.0/TCP 192.168.100.241:42253;received=192.168.100.241;rport=42253;branch=z9hG4bKPje1f91e8c-9634-4dfa-be28-4ec75946b6a3;alias
Max-Forwards: 70
From: "6002EXT" <sip:6002@192.168.100.177>;tag=7f16b000-632a-4dbc-b893-dccf47d253e7
To: sip:6000@192.168.100.177
Contact: "6002EXT" <sip:6002@192.168.100.241:42253;transport=TCP;ob>;+sip.ice
Call-ID: 69535656-663f-4949-b09a-f26d21419ffb
CSeq: 6748 INVITE
Route: <sip:6002@192.168.100.177;transport=tcp;lr>
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800
Min-SE: 90
```

```
Content-Type: application/sdp
Content-Length: 766
```

A variável **\$fU** contem o *username* na URI do cabeçalho “From” da mensagem SIP. Isso implica no identificador (na pessoa) que está originando a chamada, via SIP INVITE. É o *caller* do SIP INVITE.

Ex:

```
INVITE sip:crdphmacl_SPnuV5xqtnSX@131.221.240.204:36137;transport=TCP;ob SIP/2.0
Record-Route: <sip:icchw.acmeddns.com.br;transport=tcp;lr;did=17.7b63b2d4>
Via: SIP/2.0/TCP icchw.acmeddns.com.br:5060;branch=z9hG4bKc555.156cbd61.0;i=2
Via: SIP/2.0/TCP 192.168.100.241:42253;received=192.168.100.241;rport=42253;branch=z9hG4bKPje1f91e8c-9634-4dfa-be28-4ec75946b6a3;alias
Max-Forwards: 70
From: "6002EXT" <sip:6002@192.168.100.177>;tag=7f16b000-632a-4dbc-b893-dccf47d253e7
To: sip:6000@192.168.100.177
Contact: "6002EXT" <sip:6002@192.168.100.241:42253;transport=TCP;ob>;+sip.ice
Call-ID: 69535656-663f-4949-b09a-f26d21419ffb
CSeq: 6748 INVITE
Route: <sip:6002@192.168.100.177;transport=tcp;lr>
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800
Min-SE: 90
Content-Type: application/sdp
Content-Length: 766
```

A variável **\$DLG_status** contem o status de um diálogo. (Ex: encerrado, ativo).

A variável **\$DLG_count** contem o número de diálogos correntemente ativos.

A variável **\$DLG_timeout** configura o *lifetime* máximo de um diálogo.

A variável **\$fn** contem o *display name* visto no cabeçalho “From” da mensagem SIP. Pode ser o nome completo de uma pessoa.

Ex:

```
INVITE sip:crdphmacl_SPnuV5xqtnSX@131.221.240.204:36137;transport=TCP;ob SIP/2.0
Record-Route: <sip:icchw.acmeddns.com.br;transport=tcp;lr;did=17.7b63b2d4>
Via: SIP/2.0/TCP icchw.acmeddns.com.br:5060;branch=z9hG4bKc555.156cbd61.0;i=2
Via: SIP/2.0/TCP 192.168.100.241:42253;received=192.168.100.241;rport=42253;branch=z9hG4bKPje1f91e8c-9634-4dfa-be28-4ec75946b6a3;alias
Max-Forwards: 70
From: "6002EXT" <sip:6002@192.168.100.177>;tag=7f16b000-632a-4dbc-b893-dccf47d253e7
To: sip:6000@192.168.100.177
Contact: "6002EXT" <sip:6002@192.168.100.241:42253;transport=TCP;ob>;+sip.ice
Call-ID: 69535656-663f-4949-b09a-f26d21419ffb
CSeq: 6748 INVITE
Route: <sip:6002@192.168.100.177;transport=tcp;lr>
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800
Min-SE: 90
Content-Type: application/sdp
Content-Length: 766
```

A variável **\$tU** contem o *username* na URI do cabeçalho “To”.

Ex:

```
INVITE sip:crdphmacl_SPnuV5xqtnSX@131.221.240.204:36137;transport=TCP;ob SIP/2.0
```

```

Record-Route: <sip:icchw.acmeddns.com.br;transport=tcp;lr;did=17.7b63b2d4>
Via: SIP/2.0/TCP icchw.acmeddns.com.br:5060;branch=z9hG4bKc555.156cbd61.0;i=2
Via: SIP/2.0/TCP 192.168.100.241:42253;received=192.168.100.241;rport=42253;branch=z9hG4bKPje1f91e8c-9634-4dfa-be28-4ec75946b6a3;alias
Max-Forwards: 70
From: "6002EXT" <sip:6002@192.168.100.177>;tag=7f16b000-632a-4dbc-b893-dccf47d253e7
To: sip:6000@192.168.100.177
Contact: "6002EXT" <sip:6002@192.168.100.241:42253;transport=TCP;ob>;+sip.ice
Call-ID: 69535656-663f-4949-b09a-f26d21419ffb
CSeq: 6748 INVITE
Route: <sip:6002@192.168.100.177;transport=tcp;lr>
Allow: PRACK, INVITE, ACK, BYE, CANCEL, UPDATE, INFO, SUBSCRIBE, NOTIFY, REFER, MESSAGE, OPTIONS
Supported: replaces, 100rel, timer, norefersub
Session-Expires: 1800
Min-SE: 90
Content-Type: application/sdp
Content-Length: 766

```

Requisito: 2 ligações simultâneas no máximo

Imagine que apenas duas ligações simultâneas podem existir. Uma terceira ligação não pode ser permitida. O código abaixo mostra como implementar esse exemplo de requisito:

```

if ($DLG_count==2){
    #Já existem 2 ligações. Retorne sinal de ocupado (486) ou outro aviso SIP coerente (Ex: 603 = decline). Ex:
    # 600 = ocupado em todo lugar.
    t_reply("600", "Sistema já ocupado com 2 ligações. Tente mais tarde.");
    exit;
}

```

Para esse requisito bastou verificar se já existe 2 diálogos correntes ativos. Em caso positivo, o “exit” faz o proxy abandonar o INVITE recebido e portanto a sessão não é estabelecida para uma nova ligação.

Requisito: proibido ligar para usuários específicos

```

if (is_method("INVITE")) {

    if ((($rU==$avp(I_A))|| ($rU==$avp(I_B))) {
        t_reply("486", "Chamada impossível."); # Está proibido ligar para o usuário A ou B..
        exit;
    }
}

```

O código acima simplesmente analisa se o *username* corresponde a um de 2 valores. Ou seja, se o *username* presente no INVITE corresponde ao valor em *avp(I_A)* ou *avp(I_B)*, então a ligação não pode ser completada. Evidentemente, os *avps* já devem estar inicializados com os *usernames* dos usuários que não podem receber a ligação.

Requisito: uso de certos usuários prioritários que podem ligar

```

if ((($rU==$avp(M_T))||($fU==$avp(M_T))) {
    #ligação permitida. Nada a ser feito aqui.
}
else{
    if ((($fU==$avp(I_A))|| ($fU==$avp(I_B))) {
        #ligação permitida. Nada a fazer aqui.
    }
    else{
}
}

```

```

        t_reply("486", "Chamada proibida.");
        Exit; #Esse exit elimina a mensagem SIP INVITE e a ligação não é estabelecida.
    }
}

```

O código acima mostra que a ligação será permitida somente se o usuário definido em *avp(M_T)* é um dos *peers* (*caller* ou *callee*) ou se o originador é o usuário contido em *avp(I_A)* ou no *avp(I_B)*. Em outras palavras, se um dos *peers* é o usuário M (Ex: diretor de uma empresa), então não pode haver qualquer restrição na ligação, quer M esteja recebendo a chamada ou originando-a. Além disso, se o originador é o usuário A ou B (Ex: cozinheiro do restaurante de uma empresa e seu ajudante), a ligação poderá ser completada. Em qualquer outro caso, a ligação será proibida e evitada. Então, se algum colaborador da empresa, que não é o diretor, tentar ligar para o usuário A ou B, a ligação não será completada. Se A tentar ligar para B, ou vice-versa, idem.

Requisito: derrubar ligações quando ocorrer chamada de usuários prioritários

```

if (($fU==$avp(I_A))||($fU==$avp(I_B))){  

    # executar um binario externo que se encarrega de derrubar ligações em andamento.  

    exec("/usr/bin/FifoCmd $ci"); # comando fifo para derrubar outras ligações agora. $ci contem o callid da  

    # unica ligação que não pode ser derrubada: a do próprio usuário chamador.  

}

```

Nesse requisito, é analisando se o usuário *caller* é A ou B. Se é um deles, todas as ligações em andamento devem ser derrubadas imediatamente (até mesmo qualquer uma do diretor da empresa). Por exemplo, se A e B são usuários do departamento de bombeiros da empresa, ou são sistemas de alarmes contra incêndios, então as ligações em andamento devem ser encerradas automaticamente pelo OpenSIPS. Nesse caso, o OpenSIPS envia mensagem SIP CANCEL a todos os *peers* em ligações correntes. Isso é muito útil quando A ou B deve ter a chamada (seu SIP INVITE) encaminhada de forma *broadcast* para todos os *peers* do domínio do proxy. Então, antes de encaminhar um SIP INVITE a um destinatário, o mesmo não pode estar numa ligação corrente, caso contrário o *caller* receberia sinal de ocupado. É devido a isso que deve-se derrubar as ligações em andamento, quando a ligação de A ou B é muito mais prioritária.

Nesse requisito, o OpenSIPS executa um arquivo binário externo (Ex: programa feito em C++). Esse arquivo binário (FifoCmd) recebe como parâmetro um *call ID* (**\$ci**). Esse *call ID* identifica a ligação originada pelo próprio usuário A ou B. Cada ligação SIP tem um *call ID*, que é um identificador universal da chamada. Duas chamadas diferentes nunca têm o mesmo *Call ID*. O executável C++ solicita ao OpenSIPS a lista de todas as ligações em andamento. Essa é uma lista das ligações que serão derrubadas antes de completar a nova ligação originada pelo usuário A ou B. O OpenSIPS retorna então a lista de todos os *call IDs* de todas as ligações, até mesmo da nova ligação sendo montada. Em seguida, o executável FifoCmd solicita ao OpenSIPS que ele derrube todas as ligações de determinados *call IDs*. Tais *call IDs* identificam todas as ligações, mas o executável obviamente não solicita para derrubar a ligação do próprio usuário A ou B, porque ele conhece o *call ID* (passado como o parâmetro **\$ci**) respectivo e não inclui na lista de ligações a serem derrubadas. Resumidamente, o OpenSIPS invoca um executável externo. Tal executável

interaja com o OpenSIPS solicitando a ele uma lista de chamadas em andamento e depois soliciando a ele para derrubá-las.

Há duas formas de criar chamadas em *broadcast* no OpenSIPS. Por exemplo, se o usuário A precisa chamar todos os *peers* do domínio do OpenSIPS e todos eles estão registrados com o mesmo *username* U, então basta A chamar U (EX: SIP INVITE para U). Nesse caso, o próprio OpenSIPS enviará um SIP INVITE de forma *broadcast* para todos os *peers* online. E assim que um dos *softphones* (*peers*) atende, todos os outros recebem automaticamente um SIP CANCEL e param de 'ringar'. Com o Asterisk isso seria mais trabalhoso de implementar. Mas, essa não é a forma mais elegante de fazer o *broadcast* das chamadas, porque ela exige que cada *peer* esteja online com um mesmo *username*. O ideal é programar o script do OpenSIPS, para que esse *proxy* envie um SIP INVITE para uma lista de *peers*, mesmo que eles estejam online cada um com *username* diferente. Para tal, um aluno dessa disciplina poderia pesquisar como montar *broadcast* de ligações, com a forma mais elegante, e apresentar a pesquisa feita oralmente em sala de aula.

Requisito: registro de *dialogs* no banco de dados

```
setflag(ACC_DO); # do accounting  
# Registra o identificador do chamador e do chamado em variáveis relacionadas com a dialog atual.  
$dlg_val(caller) = $fU;  
$dlg_val(callee) = $rU;
```

Esse código acima registra no banco de dados que há ligação corrente envolvendo 2 *peers*. Na tabela *dialog*.

Requisito: registro de *peer* fora de qualquer ligação

Quando a ligação é encerrada, o seguinte código tem a chance de ser executado :

```
if (is_method("BYE") ) {  
    if( ($fU==$avp(l_A)) || ($tU==$avp(l_A)) ){  
        # Uma ligação com usuário A está sendo encerrada nesse momento.  
        # Aqui deve ser executada uma query SQL para atualizar um registro na tabela GeneralConfigurations.  
        avp_db_query("UPDATE GeneralConfigurations SET Value='no' WHERE Attribute = 'EXISTS_SPECIAL_CALL'");  
    }  
}
```

Imagine que é necessário saber que o usuário A não está envolvido em qualquer ligação a qualquer momento. Nesse caso, sempre que o usuário A sair de uma ligação, deve ser registrado em alguma tabela do banco de dados tal informação, para consulta posterior. O OpenSIPS, quando instalado, traz seu banco de dados com suas tabelas originais. Mas, novas tabelas podem ser criadas para qualquer projeto. Por exemplo, uma nova tabela chamada GeneralConfigurations poderia ser criada, com uma coluna chamada EXISTS_SPECIAL_CALL. Essa coluna poderia conter um valor 'booleano' e deveria conter 'no' sempre que o usuário A não estivesse envolvido em qualquer ligação corrente. O código visto acima serve bem para isso.

Requisito: Duração máxima de ligação

```

# create dialog with timeout
if ( !create_dialog("B") ) {
    send_reply("500","Erro interno no sip server");
    exit;
}

avp_db_query("select Value from GeneralConfigurations where Attribute = 'CONFIGURATION_CALL_DURATION'";
"$avp(CallMaxDuration)");
$DLG_timeout = $avp(CallMaxDuration);      #Estabelece quanto tempo pode durar uma ligação no máximo. Minutos
representados em segundos.
$DLG_timeout = $DLG_timeout * 60;
$avp(CallMaxDuration) = NULL;

```

O código acima lê o valor contido na coluna CONFIGURATION_CALL_DURATION, da tabela GeneralConfigurations do banco de dados. O valor lido, multiplicado por 60, é atribuído à *core variable* **\$DLG_timeout**. Quando isso é feito, nenhuma ligação durará mais que o tempo estipulado. Se uma ligação em andamento não for encerrada, pelos usuários, antes desse tempo, o próprio OpenSIPS enviará uma SIP CANCEL aos *peers* participantes da *dialog*.

Para que esse esquema funcione, as *dialogs* devem ser de um dado tipo. Esse mesmo tipo é definido na função *create_dialog()* com o parâmetro “B”.

BÔNUS. Requisito: ligações com áudio entre peers em redes diferentes

Estas ligações devem passar corretamente através de NATs. Isso requer modificação no conteúdo de cabeçalhos de mensagens SIP + modificações no conteúdo do SDP. Essas modificações são garantidas através das passagens abaixo:

```

if(nat_uac_test("34")){
    if(is_method("REGISTER")){
        fix_nated_register();
        setbflag(NAT);
        $avp(attr) = "in_another_network";
    } else{
        fix_nated_contact();
        setflag(NAT);
    }
}
else{
    if(is_method("REGISTER")){
        if($avp(DOMAIN) == NULL){
            avp_db_query("select domain from domain where attrs = 'Public'", "$avp(DOMAIN)");
        }

        if($avp(DOMAIN)==$td){
            $avp(attr) = "in_another_network";
        }
        else{
            $avp(attr) = "in_same_network";
        }
    }
}

```

A função **nat_uac_test()** verifica se a mensagem SIP REGISTER vem de um nodo na rede atrás de um NAT, do ponto de vista do *proxy*. Ou seja, se o usuário não está na mesma rede local que o

proxy. Essa função consegue fazer a verificação baseado em dados contidos no cabeçalho da mensagem SIP REGISTER e com comparação com o IP real de onde a mensagem SIP REGISTER realmente vem. Quando a verificação dá um resultado positivo, o registro do mesmo usuário na tabela *location* recebe o *flag* “*in_another_network*”. Basta atribuir esse valor ao *avp attr*. Esse *flag* permanece na tabela *location* enquanto o usuário estiver online. Isso permite, a qualquer momento, saber se um *softphone* online está trabalhando atrás de um NAT. Quando um *softphone* se registra no *proxy* através do nome do domínio e não do IP da rede local, isso significa que ele está atrás de um NAT (em outra rede). Nesse caso, ele também é marcado com o flag “*in_another_network*”. A tabela *domain*, na coluna *domain*, contém o valor do domínio pelo qual o *proxy* é responsável. Ex: icchw.acmeddns.com.br. O artifício acima de verificar qual o domínio estava definido na mensagem SIP REGISTER, para determinar se ela vem de outra rede, não é o mais adequado. Na verdade, deve-se aferir se o *peer* está se registrando presente atrás de um NAT usando apenas funções do OpenSIPS para tal. Sem verificar qual domínio o usuário definiu no seu *softphone* no momento do registro. Mas, quando esse documento foi escrito, o código acima foi baseado num dos projetos feitos no ICC e um estudo mais apurado sobre as funções do OpenSIPS não foi desenvolvido. Portanto, vale a pena um aluno do Inatel investigar como garantir que um peer está realmente atrás de um NAT, usando apenas funções específicas para isso, como a **nat_uac_test()** e depois apresentar as conclusões de forma oral em sala de aula.

Quando ocorre uma ligação, a mensagem SIP INVITE é examinada, para a determinação de sua origem, no sentido de saber se há um NAT entre o originador e o *proxy*. Para tal, basta verificar quem é o chamador e então olhar na tabela *location* se tal chamador está “*in_another_network*” ou “*in_same_network*”. Como no código abaixo:

```

if($avp(DOMAIN) == NULL){
    avp_db_query("select domain from domain where attrs = 'Public'", "$avp(DOMAIN)");
}
$avp(ContatoChamador) = NULL;
$avp(ContatoChamador) = ${ct.fields(uri){s.select,0,:}};
$avp(ReceivedChamador) = NULL;
$avp(ReceivedChamador) = ${ct.fields(uri){s.select,1,@}{s.select,0,:}};

#caller is in another network
if (nat_uac_test("34") || ($avp(DOMAIN)==$fd))
{
    set_advertised_address ("$avp(DOMAIN)");
}
else
{
    #caller is in local network

    #If callee is behind a NAT from the point of view of opensips
    if (${avp(attr)[$T_branch_idx]} == "in_another_network")
    {
        set_advertised_address ("$avp(DOMAIN)");

        #Se chamador está na rede local e o chamado não, então o SDP do chamador deve ser corrigido com um IP
        #publico.
        $avp(PUBLICIP_I) = null;
        avp_db_query("select Value from GeneralConfigurations where Attribute = 'CONFIGURATION_PUBLIC_IP'",
"$avp(PUBLICIP_I)");
        fix_nated_sdp("10","$avp(PUBLICIP_I)");
    }
}

```

```

#flags - the value may be a bitwise OR of the following flags:
#0x02 - rewrite media IP address (c=) with source address of the message or the provided IP address (the provide IP address take precedence over the source address).
#0x08 - rewrite IP from origin description (o=) with source address of the message or the provided IP address (the provide IP address take precedence over the source address).

}

else
{

    # nada a fazer.
}
}

```

Se o chamador está em outra rede que não a mesma do OpenSIPS, a função **set_advertised_address()** troca alguns dados no cabeçalho de mensagens futuras enviadas do *callee* para o *caller*. A mesma alteração em cabeçalho ocorre também quando o *caller* está na rede local, mas o *callee* não. Nesse caso, há um NAT do ponto de vista do *callee* e ele precisa da rota correta que leva até o *caller*, para quando precisar responder a mensagem SIP INVITE. Então, a função **set_advertised_address()** quando chamada, se encarrega de corrigir certos IPs nos cabeçalhos das mensagens SIP atravessando NATs, para que quem está na frente de um NAT, saiba como enviar *media* a quem está atrás dele.

Quando o *caller* está na rede local, mas o *callee* não, então o conteúdo do SDP que vai junto com o SIP INVITE também precisa ser alterado, para que o *callee* saiba corretamente para onde mandar a sua mídia (áudio por exemplo.). Para tal, é usada a função **fix_nated_sdp()**. Entretanto, o ICE e STUN ainda podem ser necessários, como explicado anteriormente nesse documento.

Além das correções em relação ao SIP INVITE, existem também aquelas em relação ao SIP OK. Por exemplo:

```

$avp(Contato) = ${ct.fields(uri){s.select,0,;));
if (nat_uac_test("1")){
    fix_nated_contact();
    $avp(FixedContato) = ${ct.fields(uri){s.select,0,;});
}

if (t_check_status("2[0-9][0-9]")||t_check_status("1[8-8][0-9]")) {
    #se a resposta eh um OK ou um Ring, se os peers estao em rede diferente e se a resposta vem do peer que estah na rede local, deve-se corrigir o IP do SDP.
    #1 - Verifica, atraves da tabela location, se chamador e chamado estao na mesma rede.
    #2 - Se estao em redes diferentes, verifica se a resposta em questao vem de um peer local
    #3 - Se sim, entao corrige o IP no SDP.

$avp(NET_Chamado) = NULL;
$avp(NET_Chamador) = NULL;
# rede do chamador:
    avp_db_query("select attr from location where contact like '$avp(ContatoChamador)%' and username = '$fU'", "$avp(NET_Chamador)");
    if($avp(NET_Chamador) == NULL){
        $var(received) = "sip:" + $avp(ReceivedChamador);
        avp_db_query("select attr from location where received like '$var(received)%' and username = '$fU'", "$avp(NET_Chamador)");
    }
# rede do chamado:

```

```

    avp_db_query("select attr from location where contact like '$avp(Contato)%' and username = '$tU",
    "$avp(NET_Chamado)");
    if($avp(NET_Chamado) == NULL){
        avp_db_query("select attr from location where contact like '$avp(FixedContato)%' and username = '$tU",
        "$avp(NET_Chamado)");
    }
    if($avp(NET_Chamado) == NULL){
        avp_db_query("select attr from location where contact like '$avp(Contato)%' and username = '$avp(M_T)"',
        "$avp(NET_Chamado)");
    }

if($avp(NET_Chamado) != $avp(NET_Chamador)){
    #chamador e chamado estão mesmo em redes diferentes.
    if($avp(NET_Chamado) == "in_same_network"){
        # o chamado estah realmente na rede local, então essa resposta do chamado deve ter o IP no SDP alterado.
        $avp(PUBLICIP_II) = null;
        avp_db_query("select Value from GeneralConfigurations where Attribute = 'CONFIGURATION_PUBLIC_IP"',
        "$avp(PUBLICIP_II)");
        fix_nated_sdp("10","$avp(PUBLICIP_II)");
    }
}
$avp(NET_Chamado) = NULL;
$avp(NET_Chamador) = NULL;
}
$avp(Contato) = NULL;
$avp(FixedContato) = NULL;

```

O código acima verifica se chamador e chamado estão em redes diferentes. Se sim e se o chamado está na rede local, a resposta do mesmo tem o conteúdo do SDP alterado. Corrigido. Para esse algoritmo, a primeira coisa a ser feita é detectar onde está cada *peer*. Para isso, é pesquisado o atributo *attr* da tabela *location*, para cada assinante. Esse atributo indica se um *peer* está “*in_same_network*” ou “*in_another_network*” do ponto de vista do OpenSIPS. A pesquisa na tabela *location* depende do *username* do *peer* obviamente, mas também do contato real do mesmo. Um mesmo assinante pode estar online em vários dispositivos móveis. Daí a necessidade de fazer tal pesquisa não somente pelo *username*. Tem-se que saber exatamente a localização do usuário via contato. O contato tem informação de IP e porta de onde vem a mensagem do usuário. E quando o usuário está atrás de um NAT, o seu contato real fica na coluna *received* e não na coluna *contact*, na tabela *location*.

O requisito explicado acima, em linhas gerais, consiste em BÔNUS nesse material, porque ele serve de ponto de partida a quem irá trabalhar com OpenSIPS e NATs, mas é um tópico mais avançado que o desejado para essa disciplina e portanto foi comentado rapidamente. Contudo, além do exemplo acima, pode-se encontrar na Internet muitas discussões e exemplos de como atravessar NATs, em termos de correções de mensagens SIP e SDP. Vale muito a pena pesquisar outros exemplos de soluções para atravessar NATs e trazê-las para a sala de aula em forma de apresentação oral feita por alunos dessa disciplina. A solução apresentada aqui não é a melhor obrigatoriamente e é possível que haja códigos mais simples que esse.

Além disso, a página [68] apresenta todos os módulos do OpenSIPS e muitos deles não discutidos aqui poderiam ser estudados e apresentados oralmente na sala de aula, o que seria um trabalho mais fácil que propor outro código para atravessar NATs. E muitas das *core functions* (funções não relacionadas aos módulos), podem ser vistas na página [69].

Como instalar e configurar o OpenSIPS para iniciar o uso dele

O Primeiro passo para a instalação do OpenSIPS é obter o código dele, via git. O segundo passo é realizar a compilação do mesmo código. Finalmente, pode-se configurar esse *proxy* através do arquivo ***opensips.cfg***. Esse é o arquivo onde é possível escrever o *script* do *proxy*.

Quando o OpenSIPS foi obtido para testes pela primeira vez em 2016, o código da versão 2.2.1 (*stable LTS release*) foi baixado. Contudo, essa seção considera a obtenção e compilação da versão 2.3 do OpenSIPS, que é uma das mais recentes em 2018. E por *default*, a instalação do OpenSIPS e sua configuração usam o MySQL como banco de dados. Mas, os passos abaixo consideram que o MySQL será substituído pelo SQLite.

Siga os passos abaixo para obter o código fonte, compilar e configurar o OpenSIPS no seu computador. Use Linux.

“Para saber sobre o consumo de memória do OpenSIPS, consultar:

<http://www.opensips.org/Documentation/TroubleShooting-IncreaseMem> “

Abaixo seguem instruções resumidas de como fazer a instalação, sem maiores detalhes e considerando o Linux Ubuntu 14.06 LTS. No exemplo abaixo é feito um clone do projeto OpenSIPS, de tal forma que fica mais fácil contribuir com o código para o mesmo, se desejado.

INSTALAÇÃO E CONFIGURAÇÃO

```
sudo apt-get install sqlite3 libsqlite3-dev
```

```
sudo apt-get install libssl-dev
```

```
mkdir OPENSIPS_SOURCE
```

```
cd OPENSIPS_SOURCE
```

```
git clone -b 2.4 https://github.com/OpenSIPS/opensips.git opensips_2_4
```

```
sudo apt-get install flex bison libncurses5-dev
```

```
cd opensips/
```

```
sudo make menuconfig
```

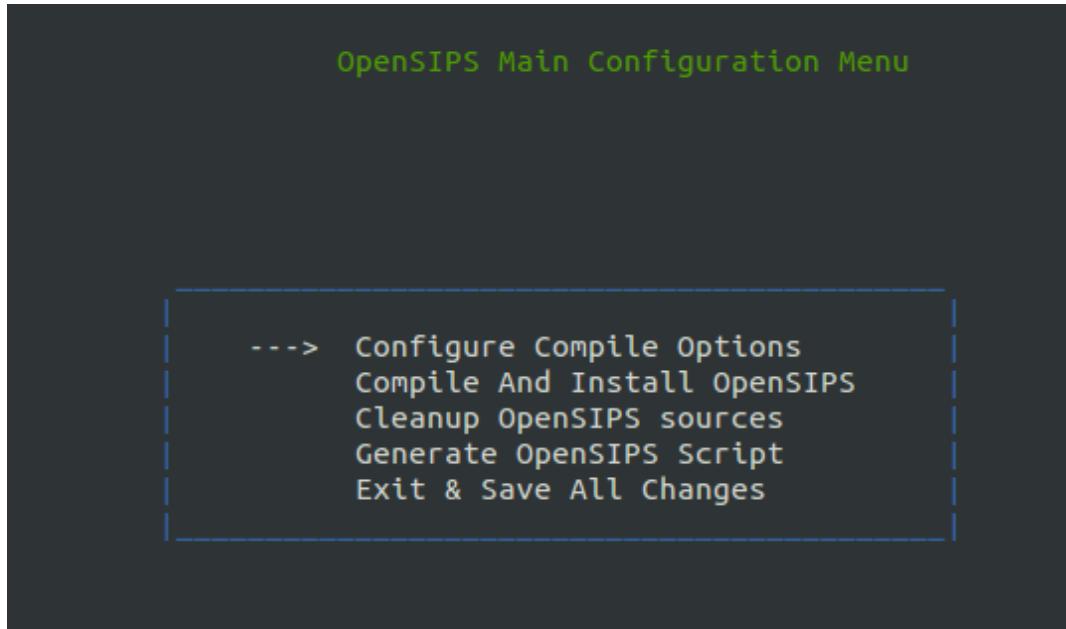


Figura 41

Para navegar no menu visto na figura 41, usa-se as setas do teclado = { ← , → }.

---> configure copile options; ----> configure copile flags

marcar as opções desejadas (ou deixe como está).

----> Configure excluded modules

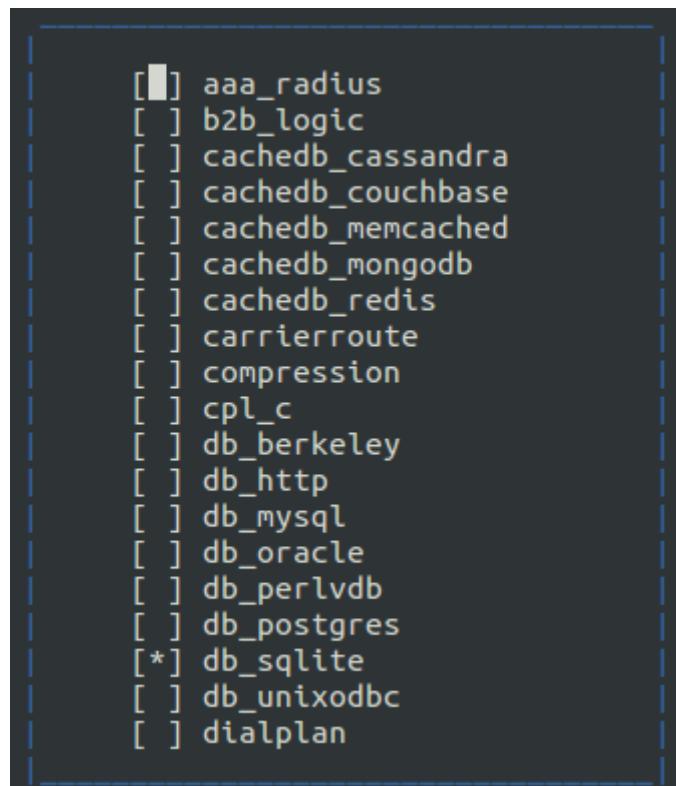


Figura 42

Como visto na figura 42, escolher **db_sqlite**. O SQLite é um banco de dados mais 'leve' que o MySQL. O SQLite deverá ser instalado também, além do OpenSIPS. A marcação acima diz que o OpenSIPS deve usar o seu próprio módulo db_sqlite, não que o banco em si vá ser instalado. Marcar também o **proto-tls**, para que o OpenSIPS tenha suporte à TLS, se for desejado futuramente.

Configure Install Prefix (para configurar o diretório de instalação do OpenSIPS)

EX: /usr/local/ (Valor já default. Basta confirmar esse)

---> Save changes

Ao avisar sobre libs do sqlite e tls, apenas pressione qualquer tecla para continuar.

---> Compile and Install OpenSIPS

Aguarde compilar tudo!

---> Exit and save all changes.

cd /usr/local/etc/

sudo chmod 755 -Rf opensips

cd opensips

sudo vi opensipsctlrc

A seguinte configuração deve ser feita no arquivo opensipsctlrc:

**## your SIP domain
SIP_DOMAIN=192.168.37.104 Use aqui o IP corrente da sua máquina.**

**##== database type: MYSQL, PGSQL, ORACLE, DB_BERKELEY, DBTEXT, or SQLITE
by default none is loaded
If you want to setup a database with opensipsdbctl, you must at least specify
this parameter.
DBENGINE=SQLITE**

**##== database host
DBHOST=localhost**

**##== database name (for ORACLE this is TNS name)
DBNAME=opensips**

**#== database path used by dbtext, db_berkeley, or sqlite
DB_PATH="/usr/local/etc/opensips/sqlite"**

**##== database read/write user
DBRWUSER=opensips**

**##== password for database read/write user
DBRPW="opensipsrw"**

**##== database super user (for ORACLE this is 'scheme-creator' user)
DBROOTUSER="root"**

Salve o arquivo e saia dele. Continuar com as instruções abaixo:

```
sudo apt-get install sqlite3-pcre
```

```
cd /usr/local/sbin
```

```
sudo ./opensipsdbctl create
```

Não deixe instalar tabelas extras. (EX: relacionadas com presence e outras). Estão fora do escopo da primeira aula dessa disciplina.

O OpenSIPS nesse momento permite instalar várias outras tabelas, além das default. Uma delas é a de presence. Uma pesquisa sobre a utilidade dessas tabelas e uma explicação sobre elas poderiam ser feitas para apresentação de mais informação em sala de aula, por parte de um aluno.

As tabelas default são:

acc	dispatcher	grp	speed_dial
address	<u>domain</u>	load_balancer	<u>subscriber</u>
aliases	dr_carriers	<u>location</u>	tls_mgm
clusterer	dr_gateways	missed_calls	uri
dbaliases	dr_groups	re_grp	usr_preferences
<u>dialog</u>	dr_partitions	rtpproxy_sockets	version
dialplan	dr_rules	silo	

```
sudo apt-get install m4
```

Até aqui o OpenSIPS foi obtido, compilado e instalado. Agora os passos seguintes mostrarão como gerar o seu arquivo inicial de configuração, o arquivo **opensips.cfg**. Na verdade um arquivo *standard* mínimo já está pronto aqui: /usr/local/etc/opensips. As instruções abaixo criam outro arquivo opensips.cfg com mais algumas funções em uso. Mais adiante, os 2 arquivos poderão ser comparados.

```
sudo /usr/local/sbin/osipscfg
```

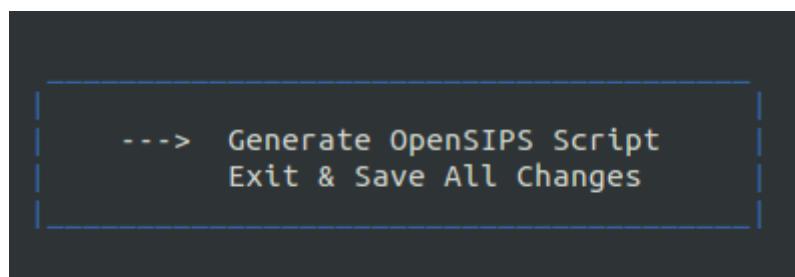


Figura 43.

```
----> Generate OpenSIPS Script
```

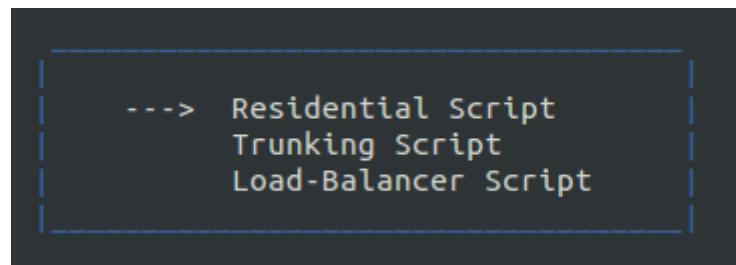


Figura 44

-----> **Residential Scripts** -----> **Configure Residential Scripts**

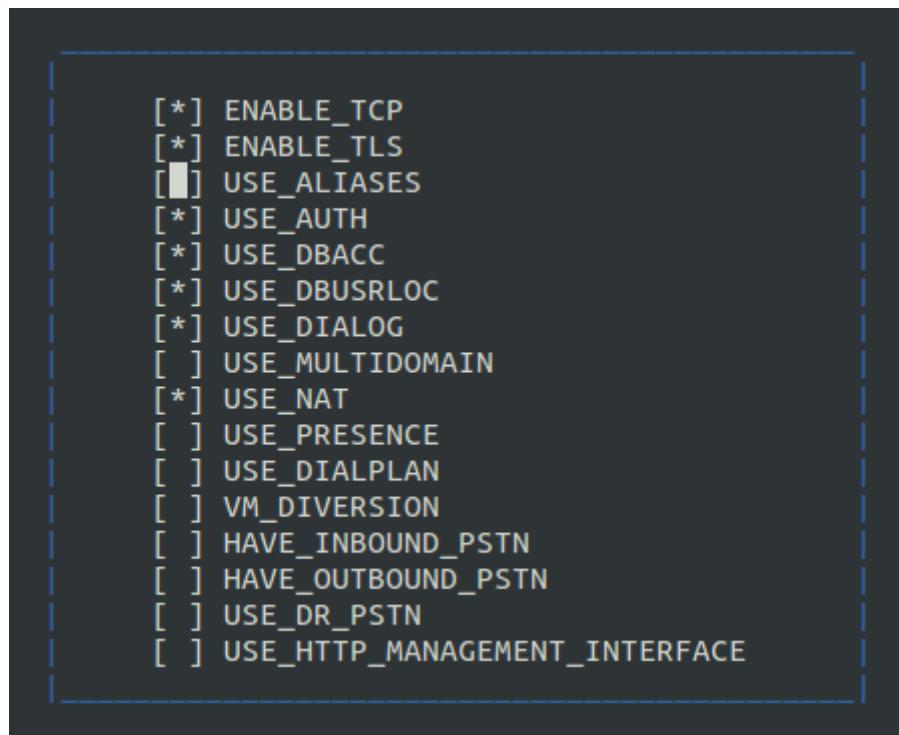


Figura 45

As opções vistas com * acima devem ser marcadas.

Voltar no menu anterior.

----> **Generate Residential Script** (Tecle <enter> aqui, para gerar os scripts)

Saia completamente da aplicação de configuração e volte no prompt do Ubuntu.

Um novo script foi gerado em /usr/local/etc/opensips. Esse script é do tipo residencial. Mas, poderia ser do tipo Trunking ou Load-balancer. Caso algum aluno pesquise a utilidade desses 2 outros tipos de script e apresente o resultado da pesquisa em sala de aula, oralmente, isso será interessante.

cd /usr/local/etc/opensips/

sudo mv opensips.cfg opensips_ORIGINAL.cfg

```
sudo mv opensips_residential_*.cfg opensips.cfg
sudo chmod 755 -f opensips.cfg
sudo apt-get install meld
meld opensips.cfg opensips_ORIGINAL.cfg (comando para mostrar o que foi criado de novo no arquivo de script)
Editar o arquivo opensips.cfg e trocar as ocorrências de '127.0.0.1' por 'eth0' ou 'wlan0', conforme o link de rede usado atualmente.
```

Ainda nesse arquivo de configuração, procure as linhas com CUSTOMIZE ME e veja o que mais mudar...

Por exemplo, linhas com "mysql://opensips:opensipsrw@localhost/opensips" devem ser mudadas para "sqlite:///usr/local/etc/opensips/sqlite"
Trocá-lo loadmodule "db_mysql.so" por loadmodule "db_sqlite.so".

Ainda nesse arquivo, comentar todas as linhas relacionadas com 'tls' até que os certificados de segurança sejam gerados para uso de TLS.

Acrescentar as 2 linhas seguintes, nesse arquivo:

```
# ----- db_sqlite params -----
modparam("db_sqlite", "load_extension", "/usr/lib/sqlite3/pcre.so")
```

Essas 2 linhas acrescentadas podem vir antes de “### DIALOG module”.

Salve e saia do arquivo opensips.cfg.

O executável do OpenSIPS e outros scripts auxiliares estão em /usr/local/sbin. No Anexo I dessa apostila existe o código para o arquivo opensips.cfg. Esse código é mais elaborado que o código default gerado pelo instalador. Portanto, se desejado, pode-se substituir um código pelo outro. O código a ser substituído, se desejado, estará em /usr/local/etc/opensips .

Preparar para executar o OpenSIPS:

Numa shell qualquer, execute os comandos:

```
cd /var/log
tail -f syslog
```

Esses comandos acima permitirão ver o log da execução do OpenSIPS. Em outra shell qualquer, executar os seguintes comandos:

```
cd /usr/local/etc/opensips
sudo ../../sbin/opensipsctl start
```

No log poderá aparecer mensagens de erros relacionadas com *rtpproxy*. Elas podem ser desprezadas no escopo da aula sobre OpenSIPS. Entretanto, o uso de *rtpproxy* pode ser feito para tráfego de pacotes de *media*. Mas, nesse caso a máquina tem que se comportar como um *rtp proxy* também e a *media* passa por ela. O assunto de uso de *rtp proxy* não foi pesquisado. Caso algum aluno decida

estudar sobre isso, poderá fazê-lo com ajuda do próprio site do OpenSIPS, que tem alguma documentação sobre isso na parte que descreve os módulos desse *proxy*. O entendimento do assunto seria apropriado apresentar em sala de aula oralmente.

Pronto! O OpenSIPS já está em execução e já pode ser usado como um SIP *proxy* executando em seu computador.

Comandos opensipsctl

Vários comandos úteis do OpenSIPS estão registrados abaixo.

[Sudo opensipsctl online](#) (mostra quais usuários estão registrados e online no OpenSIPS)

[sudo opensipsctl ul show 6000](#) (mostra detalhes do registro e localização do usuário 6000)

[sudo opensipsctl add 8000 1234](#) (cria novo assinante 8000 com senha 1234)

[sudo opensipsctl rm 8000](#) (remove assinante 8000)

[sudo opensipsctl ul rm 8000](#) (coloca assinante 8000 em offline, até que ele se registre novamente)

[sudo /opensipsctl start](#) (inicia o OpenSIPS)

[sudo /opensipsctl stop](#) (encerra o OpenSIPS)

Esses comandos são executados a partir da pasta **/usr/local/sbin**.

Tabelas do banco de dados do OpenSIPS

Como explicado anteriormente, o SIP *proxy* contém um SIP REGISTRAR, que é um banco de dados no mesmo hardware onde está o OpenSIPS. Esse banco de dados está implementado em Sqlite e inicialmente foi montado durante a instalação do SIP *proxy*. Isto é, o SIP *proxy* escolhido nesse projeto já vem com um REGISTRAR e tabelas *default*. Por exemplo, ao instalar o SIP *proxy* (da forma como ele vem de “fábrica”), o banco de dados é instalado também. E esse banco já contém inicialmente tabelas para registrar dados de localização de usuários online, dados de assinantes de contas SIP, registro de chamadas não atendidas por um usuário, etc.

Algumas dessas tabelas originais e seus significados estão abaixo:

domain

essa tabela contém o nome do domínio do SIP *proxy*. Cada SIP *proxy* cuida de um domínio, que é o nome do conjunto de seus próprios assinantes. Um domínio pode ter qualquer nome. O nome do domínio geralmente substitui o IP do mesmo nodo onde roda o SIP *Proxy*. Depois que um domínio é definido, um assinante pode se registrar no SIP *proxy* usando o mesmo domínio, não somente via IP. Ou seja, um domínio definido representa o IP público onde o SIP *proxy* é

alcançável. A definição do nome do domínio na tabela domain pode ser feita manualmente, com query SQL, por exemplo.

location

essa tabela contem dados de localização de um usuário correntemente online no SIP *proxy*. Um usuário online pode estar conectado no *proxy* a partir da mesma rede ou a partir de uma outra rede. Do ponto de vista do SIP *proxy*, um usuário online pode estar *in_same_network* ou *in_another_network*. Esses dois atributos ficam na coluna *attr* da tabela *location*, mas somente se o script do OpenSIPS estiver programado para guardar tal informação, como explicado anteriormente nesse documento. Tal coluna pode ser adicionada à tabela *location*, ela não é criada automaticamente com a instalação do OpenSIPS. Essa tabela contem as seguintes colunas e seus significados:

- *id* → contem a chave primária do registro.

- *username* → identificador (login) de um assinante.

- *callerName* → essa coluna não é original do banco de dados do SIP proxy. Ela pode ser criada especialmente para algum projeto. Pode conter o nome do usuário a ser mostrado na GUI de um *softphone*. Ex: ao receber uma chamada num *softphone*, aparece o callerName de quem está chamando. Dependendo do *softphone* usado, o usuário pode cadastrar o seu callerName num atributo chamado *display name* ou *Caller ID* (caso do ZOIPER), por exemplo.

- *contact* → Quando um usuário SIP se registra no SIP *proxy*, tal *proxy* recebe um SIP REGISTER. E tal mensagem SIP é montada no próprio *user agent client* e contem o contato do usuário. Esse contato vai então para a coluna contact. Ou seja, o agente SIP do próprio usuário (cliente) gera um dado de contato e entrega para o SIP *proxy*, para esse *proxy* saber 'teoricamente' onde contatar o cliente. Ex de contato: *sip:6001@192.168.21.136:5062;transport=TCP*. Esse contato significa que o agente SIP do cliente declara estar alcançável no IP 192.168.21.136, pela porta 5062. Ou seja, quem quiser contatar tal usuário, deve mandar mensagem para esse IP. Mas, nem sempre isso é correto. Ou melhor, se o agente SIP está atrás de um NAT, ele não saberá disso e informará o seu IP privado, não um público. Dessa forma, se o agente SIP do cliente está atrás de um NAT (*softphone* conectado na rede atrás de um NAT do ponto de vista do SIP *proxy*), ele informará um IP e porta inválidos para contato com ele mesmo.

- *received* → Quando um agente SIP cliente se registrar no SIP *Proxy* a partir de um ponto atrás de um NAT, o *proxy* perceberá que o contato declarado por tal agente não é válido. Nesse caso, o SIP *proxy* tem a capacidade de identificar qual é o IP e porta por onde veio a mensagem SIP REGISTER na real e considerar esse nodo como sendo o local de onde recebeu (ou onde está o agente cliente). Tal informação composta por IP e porta vai para a coluna *received*. Então, basicamente, para cada usuário online via a mesma rede do *proxy*, não haverá valor nessa coluna. O contrário também é verdadeiro.

- *expires* → apenas contem o momento quando o registro de um usuário online irá expirar. Nesse

mesmo momento o agente SIP cliente (*softphone* do usuário) irá mandar outra mensagem SIP REGISTER. Se isso não ocorrer, o SIP proxy passa a considerar o registro como expirado. Um registro expirado é automaticamente eliminado do banco de dados depois de certo tempo.

- user agent → informa qual é o dispositivo que está executando o *user agent client*, do qual partem as mensagens SIP REGISTERs. Exs:

Yealink SIP-T22P 7.42.0.30 00:15:65:15:81:6e (esse é um Voip Phone)

Zoiper r36944 (esse é um softphone)

Linphone/3.6.1 (eXosip2/4.0.0) (esse é outro softphone)

subscriber

essa tabela contém quais são os assinantes do SIP proxy. Ou seja, quais *usernames* podem se registrar e ficarem online no *proxy*. Juntamente com cada *username* existe uma senha, mas as senhas não são visíveis na tabela. Elas estão criptografadas em algumas colunas dessa tabela. Um usuário que não esteja previamente cadastrado nessa tabela não poderá se registrar via SIP e ficar online. A coluna domain sempre conterá o valor localhost para todos os registros.

Finalmente, foi dada a introdução sobre SIP e SIP Proxy. Contudo, atualmente outra tecnologia vem desportando como ferramenta de comunicação *peer to peer*, para transmissão de media em tempo real. É a WebRTC. Essa tecnologia pode ser usada por *browsers* e dispositivos móveis. Até está dito que servirá para as coisas da IoT, como dito nessa página [71]. Mas, se for assim, então essa tecnologia ajuda na economia de bateria de dispositivos com capacidades restritas? Essa tecnologia não define o protocolo de sinalização. Qualquer um pode ser usado. E algumas explicações vistas na Internet, a partir da página [71], contêm exemplos de implementações da WebRTC, com sinalização que não é o SIP. Por tudo isso, o assunto WebRTC é bem adequado a essa disciplina e serve muito bem como tema de pesquisa e apresentação oral por parte de aluno, em sala de aula!!

BÔNUS: Certificados e TLS usados no OpenSIPS

Para usar mensagens SIP criptografadas com TLS, pode-se usar o OpenSIPS e arquivos de certificados de segurança. Para tal, é necessário criar os arquivos de certificados. Como bônus (fora do escopo dessa disciplina) segue instruções a quem queira criar os arquivos de certificados.

O OpenSIPS e os *smartphones* já são preparados para trabalhar com TLS e podem se corresponder através então das mensagens SIP criptografadas. No OpenSIPS há uma configuração a ser feita para tal. E nos *smartphones* geralmente basta marcar um flag de uso de TLS.

A tecnologia TLS usa certificados e esses precisam ser criados para cada projeto. Ou seja, para um dado projeto podem ser criados os certificados a residirem no próprio *SIP Proxy*. Quando o *proxy* usa o TLS, ele criptografa as mensagens SIP e envia um certificado de cliente para o *smartphone*. O *smartphone* analisa o certificado e, se aceito, usa-o para descriptografar a mensagem. Um *smartphone* pode recusar um certificado do *proxy*, se não reconhecer o como verdadeiro. Se isso ocorrer, a mensagem não é descriptografada e não ocorre qualquer estabelecimento de sessão SIP entre fone e *proxy*. Por outro lado, quando um *smartphone* contata o *proxy*, ele deve enviar ao *proxy* o certificado próprio e o *proxy* deve aceitá-lo. Com o aceite, uma sessão SIP pode então ocorrer.

Também é possível configurar os *softphones* (Ex: no software ZOIPER) para aceitarem qualquer certificado. Dessa forma, eles aceitam certificados de qualquer *proxy*. Isso evita ter que fazer a instalação no *smartphone* de uma parte do certificado que deve constar previamente no cliente do *proxy*. Mesmo assim, a criptografia das mensagens ainda ocorre e é válida.

Já no OpenSIPS pode ser usada uma configuração de tal forma que ele não requer do *smartphone* (*softphone*) o certificado enviado do lado do cliente. Ou seja, um software como o Zoiper pode trabalhar sem ter que enviar o seu certificado. Isso ainda não invalida a criptografia. A desvantagem aqui é que qualquer *softphone* poderá tentar se comunicar com o *proxy* e qualquer *proxy* poderá se comunicar com um *smartphone*.

Como gerar os certificados (no Linux)

1 – Instale o OpenSSL no computador. É ele que cria os certificados. O OpenSSL deve estar corretamente instalado na máquina incluindo os arquivos .h de seu código fonte. Obs: Algumas bibliotecas do OpenSSL foram instaladas por instruções constantes anteriormente nessa apostila.

2 – Crie a seguinte pasta:

- testca

3 – Criar um arquivo .sh e colocar o seguinte conteúdo nele:

```
sudo rm -Rf client server
cd testca
sudo rm -Rf certs private cacert.cer cacert.pem index.txt
sudo mkdir certs private
sudo chmod 700 private
sudo chmod 777 serial
cat myserial > serial
sudo touch index.txt
```

```

sudo openssl req -x509 -config openssl.cnf -newkey rsa:2048 -days 365000 -out cacert.pem -outform PEM -subj /CN=acme_CA/ -nodes
sudo openssl x509 -in cacert.pem -out cacert.cer -outform DER
cd ..
sudo mkdir server
cd server
sudo openssl genrsa -out key.pem 2048
sudo openssl req -new -key key.pem -out req.pem -outform PEM -subj /CN=acme_CA/O=server/ -nodes
cd ../testca
sudo openssl ca -config openssl.cnf -in ../server/req.pem -out ../server/cert.pem -notext -batch -extensions server_ca_extensions
cd ../server
sudo openssl pkcs12 -export -out keycert.p12 -in cert.pem -inkey key.pem -passout pass:siscacme2015
cd ..
sudo mkdir client
cd client
sudo openssl genrsa -out key.pem 2048
sudo openssl req -new -key key.pem -out req.pem -outform PEM -subj /CN=acme_CA/O=client/ -nodes
cd ../testca
sudo openssl ca -config openssl.cnf -in ../client/req.pem -out ../client/cert.pem -notext -batch -extensions client_ca_extensions
cd ../client
sudo openssl pkcs12 -export -out keycert.p12 -in cert.pem -inkey key.pem -passout pass:siscacme2015

```

4 – Na pasta testca, crie 1 arquivo chamado *myserial*, com o seguinte conteúdo:

01

5 – No diretório testca, crie 1 arquivo chamado *openssl.cnf*, com o seguinte conteúdo:

```

[ ca ]
default_ca = testca

[ testca ]
dir = .
certificate = $dir/cacert.pem
database = $dir/index.txt
new_certs_dir = $dir/certs
private_key = $dir/private/cakey.pem
serial = $dir/serial

default_crl_days = 7
default_days = 365000
default_md = sha1

policy = testca_policy
x509_extensions = certificate_extensions

[ testca_policy ]
commonName = supplied
stateOrProvinceName = optional
countryName = optional
emailAddress = optional
organizationName = optional
organizationalUnitName = optional

[ certificate_extensions ]
basicConstraints = CA:false

[ req ]
default_bits = 2048
default_keyfile = ./private/cakey.pem
default_md = sha1
prompt = yes
distinguished_name = root_ca_distinguished_name
x509_extensions = root_ca_extensions

[ root_ca_distinguished_name ]
commonName = hostname

[ root_ca_extensions ]
basicConstraints = CA:true
keyUsage = keyCertSign, cRLSign

[ client_ca_extensions ]
basicConstraints = CA:false
keyUsage = digitalSignature
extendedKeyUsage = 1.3.6.1.5.5.7.3.2

[ server_ca_extensions ]
basicConstraints = CA:false
keyUsage = keyEncipherment
extendedKeyUsage = 1.3.6.1.5.5.7.3.1

```

6 – Execute o arquivo .sh criado. Ele irá fazer a geração dos certificados, usando o OpenSSL para tal.

Os certificados criados estarão contidos nas pastas presentes nesse diretório.

Exemplo:

```
"certificate", "./server/cert.pem"  
"private_key", "./server/key.pem"  
"ca_list",    "./testca/cacert.pem"  
"ca_dir",     "./testca/"
```

No arquivo /testca/openssl.cnf há mais configurações que podem se tornar interessantes.

Testes com TLS e softphone ZOIPER

A primeira coisa que deve ser feita aqui é configurar o OpenSIPS corretamente, para ele utilizar os arquivos de certificados, para fazer uso adequado do TLS. Tal configuração contem as seguintes linhas que devem ser escritas no arquivo opensips.cfg:

```
fork=yes # para usar TLS deixe o fork = yes.  
  
listen=tls:eth0:5061 # CUSTOMIZE ME  
listen=tls:eth0:5071  
listen=tls:lo:5061 # CUSTOMIZE ME  
listen=tls:lo:5071  
  
loadmodule "proto_tls.so"  
modparam("proto_tls","verify_cert", "0")  
modparam("proto_tls","require_cert", "0") #0 means *do not* force the client to present a certificate where as 1 means *do* ask the client to present a cert.  
modparam("proto_tls","tls_method", "TLSv1") #If you want RFC3261 conformance and all your clients support TLSv1 (or you are planning to use encrypted "tunnels" only between different  
modparam("proto_tls", "certificate", "/usr/local/etc/opensips/tls/rootCA/certs/cert.pem")  
modparam("proto_tls", "private_key", "/usr/local/etc/opensips/tls/rootCA/private/key.pem")  
modparam("proto_tls", "ca_list", "/usr/local/etc/opensips/tls/rootCA/cacert.pem")  
modparam("proto_tls", "ca_dir", "/usr/local/etc/opensips/tls/rootCA")
```

Esse código acima considera os *filepaths* que devem existir na mesma máquina onde roda OpenSIPS.

Com essa configuração feita no OpenSIPS, ele deve ser reiniciado. Para testar o mecanismo com TLS no PC , pode ser feito o seguinte:

```
> cd .../testca/  
> sudo openssl s_client -showcerts -debug -connect <IP corrente da máquina>:5061 -no_ssl2 -bugs -CAfile ./cacert.pem
```

O resultado desse comando deve ser: Verify return code: 0 (ok)

Para tirar dúvidas sobre ZOIPER, ler aqui: <http://community.zoiper.com/>

Veja como configurar o ZOIPER para fazer o papel do *peer client* e permitir a execução de testes.

Fazer o seguinte no zoiper:

No celular, após tentar registrar no OpenSIPS usando TLS, o telefone irá avisar que um certificado desconhecido foi recebido. Nesse momento, basta aceitar o certificado e deixar que o celular passe a ignorar esse tipo de arquivo. Assim as sessões SIP serão estabelecidas com o *softphone* ZOIPER no Celular.

Para o caso do ZOIPER no PC, fazer o seguinte:

- a) Settings->preferences->advanced->security.
- b) marcar o checkbox “disable certification verification”.

Com essa configuração, o ZOIPER no PC irá também conseguir estabelecer sessão SIP com o SIP Proxy OpenSIPS.

Com o ZOIPER online pode-se ligar para um outro *softphone* também online no mesmo *proxy* e ver se a sessão é estabelecida. Se for, funcionou o uso de TLS e criptografia. Para esse caso, não adianta tentar ver as mensagens SIP no Wireshark, porque não irão aparecer lá.

Capítulo V – Micros-serviços

Este capítulo tem como objetivo apresentar a arquitetura baseada em micros-serviços bem como suas características, vantagens e desvantagens comparando-as com a arquitetura monolítica em fatores como escalabilidade, autonomia, disponibilidade, acoplamento, desempenho, produtividade, coesão e resiliência, entre outros. Relata também sobre os desafios encontrados em iniciar uma aplicação utilizando a arquitetura de micros-serviços e a de decompor em micros-serviços uma aplicação monolítica já existente. Visando desta forma contribuir para a decisão de quando utilizar ou não esta arquitetura e se a decisão for tomada, de que forma esta deve ser abordada e aplicada. Todo o texto desse capítulo foi copiado do artigo [72], na íntegra. Créditos vão para Crislaine da Silva Tripoli, aluna dessa mesma pós-graduação em 2016.

Introdução

A busca por melhores formas de se construir sistemas computacionais tem sido intensa e contínua. Nesta era de disponibilidade de Internet, propagação dos dispositivos móveis, juntamente com o advento da internet das coisas (IOT – *Internet of Things*) e a computação nas nuvens, desenvolver sistemas que utilizem destes recursos e que ainda possam suportar a alta demanda de usuários e suas requisições, bem como a diversidade de tipos de clientes existentes neste cenário, pode ser um grande desafio.

De acordo com [73] no ano de 2016 estima-se que haverá 6,4 bilhões de “coisas” conectadas à rede mundial de computadores, partindo de um aumento de 30% em 2015 e chegando a 20,8 bilhões até 2020, sendo que em 2016 a previsão é de 5,5 milhões de novas “coisas” que se conectarão à rede todos os dias.

Diante deste cenário fatores como escalabilidade, desempenho, disponibilidade e produtividade surgem como pontos importantes a serem considerados no momento de se construir uma aplicação. E para alcançar estes itens, muitos conceitos têm sido discutidos e novas formas de se organizar e construir sistemas computacionais vêm sendo colocadas em prática, deixando de lado formas tradicionais de se desenvolver uma aplicação, como é o caso das aplicações monolíticas, cujo o perfil nem sempre se encaixa nesta atual perspectiva.

A arquitetura baseada em micros-serviços surge neste panorama como uma alternativa ao tradicional padrão arquitetural monolítico. Muito tem se falado deste estilo arquitetural, colocando-a no topo das expectativas exageradas de diversas pesquisas de 2015 do Gartner Hype Cycle, explicado em [95], como por exemplo sobre serviços em [94], desenvolvimento em [92] e arquitetura de aplicações em [93].

Este capítulo tem como objetivo apresentar esta arquitetura comparando suas características, vantagens e desvantagens em relação ao estilo arquitetural monolítico, bem como, apresentar os cenários onde a escolha deste estilo se torna conveniente. Passando também pelos desafios de se

construir um sistema desde o início utilizando esta arquitetura e o de decompor um sistema monolítico já existente, contribuindo na decisão de quando e como utilizar ou não o padrão de arquitetura em micros-serviços. Também é abordada a relação entre este padrão arquitetural e *Service Oriented Architecture* – SOA, comentando rapidamente também sobre o padrão de linguagem existente para a construção de sistemas baseados em micros-serviços.

A arquitetura monolítica

Tradicionalmente aplicações empresariais (*Enterprise Applications*) são compostas de três partes principais: cliente, servidor e banco de dados como explicado por [74]. A parte cliente se refere à interface com o usuário e é baseada geralmente em páginas *HyperText Markup Language* - HTML e *javascript* que rodam no navegador de um computador ou dispositivo móvel, por exemplo. Na parte de banco de dados é tradicionalmente utilizado um sistema gerenciador de banco de dados relacional, onde se encontram todas as tabelas utilizadas e compartilhadas por todas as funcionalidades do sistema. E por último, a parte servidor onde é executada toda a lógica de negócios da aplicação, manipulação de requisições *Hypertext Transfer Protocol* - HTTP, integração e troca de mensagens com outros sistemas quando necessário, atualização e recuperação dos dados no banco de dados e gerenciamento do que deve ser enviado e/ou mostrado ao cliente (HTML, *JavaScript Object Notation* – JSON, *eXtensible Markup Language* – XML, etc).

Esta última parte é o que se refere à arquitetura monolítica a qual é concretizada na forma de uma única unidade e toda a sua lógica e processamento de requisições rodam em um único processo, agrupando diversas funcionalidades dentro de um único sistema que pode ser organizado em classes, *namespaces*, funções e métodos, utilizando recursos de alguma linguagem de programação. Usando como exemplo a plataforma Java, tal sistema poderia consistir em um único arquivo *Web Application Archive* (WAR) rodando em um *container web* como o Tomcat. A Figura 46 ilustra uma aplicação web seguindo esta arquitetura.

Arquitetura Tradicional de uma Aplicação WEB

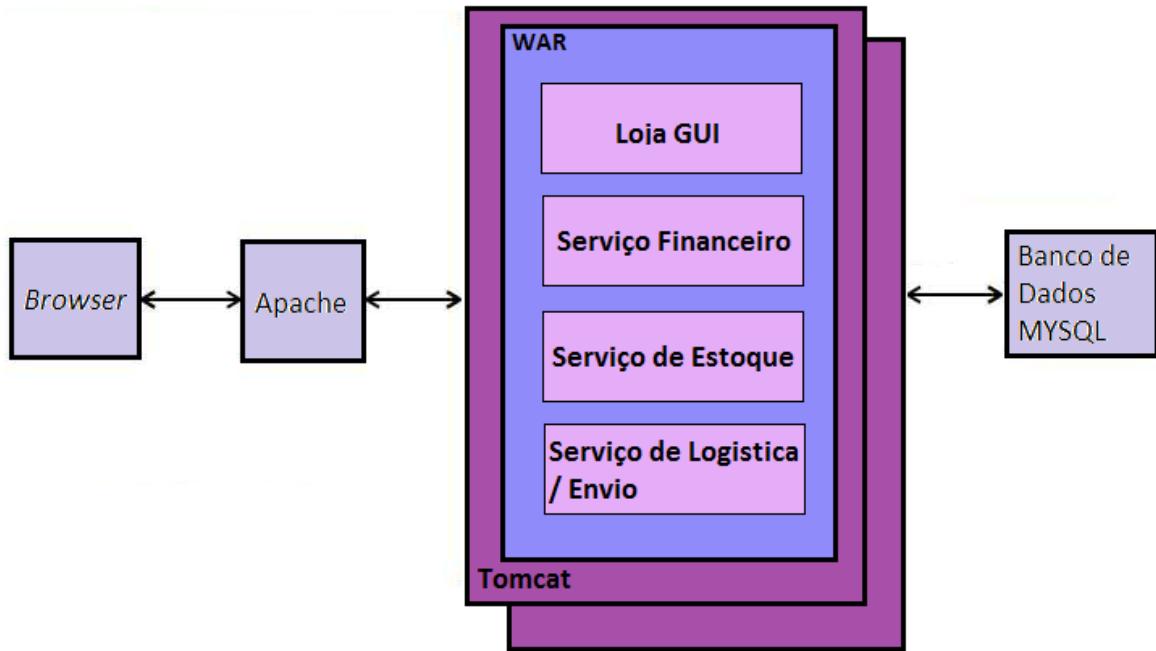


Fig. 46. Tradicional Aplicação Web Baseada em uma Arquitetura Monolítica. Fonte: [75]

Aplicações monolíticas são simples de se desenvolver comparadas a sistemas em micros-serviços, podendo contar com o suporte de inúmeras ferramentas e IDEs (*Integrated Development Environment*). Também são fáceis de implantar, pois o banco de dados evoluí junto com a aplicação para todas as funcionalidades como citado em [77] e como no caso do exemplo mostrado na Figura 46, necessitando apenas de um arquivo, como um WAR file, implantado em um servidor de aplicações. É também relativamente simples de escalar, instalando várias cópias da aplicação em múltiplas unidades de processamento que podem ser acessadas através de um平衡ador de cargas como descreve [75].

Porém, existem alguns cenários em que este tipo de arquitetura pode trazer diversos inconvenientes significativos. Como é o caso do momento em que a aplicação, com tal arquitetura, começa a crescer, tanto em quantidade de código, quanto em complexidade, demandando um time maior de desenvolvedores e formas mais eficientes de se escalar, testar e implantar, além de fatores como desempenho, disponibilidade e tolerância a falhas se tornarem cruciais para o sucesso da aplicação, uma vez que esta provavelmente precisará interagir com um grande volume de dispositivos ou usuários conectados simultaneamente.

A) Quando e como uma arquitetura monolítica se torna desvantajosa.

A Amazon, conforme citado em [85], explica que aplicações iniciam com uma abordagem monolítica, pois o desenvolvimento é muito mais rápido. Porém, à medida em que o projeto amadurece e cresce, estas ficam sobrecarregadas e seus ciclos de vida se tornam lentos demais.

Como comentado por [76], códigos crescem à medida que novos recursos ou funções são adicionadas e ao longo do tempo códigos muito grandes dificultam desenvolvedores a saberem onde alterações devem ser feitas. Funções similares começam a se espalhar por todo o código da

aplicação, fazendo com que correções de erros e novas implementações se tornem mais difíceis de serem feitas. Muitas vezes, baixa coesão e alto acoplamento se tornam comuns nestes códigos e mesmo que exista uma estrutura de componentes, estes últimos, bem como seus ciclos de vida estão todos inseridos em um único pacote ou unidade com a mesma base de código, conforme explica [77].

Estes elementos acabam se tornando um desafio arquitetural, quando se trabalha em uma arquitetura monolítica, uma vez que alterações em uma pequena parte da aplicação requer que toda aplicação seja reimplantada. E se uma parte da aplicação tem um aumento de demanda, para atendê-la é necessário escalar toda a aplicação, além de alto acoplamento, uma vez que uma mudança em um módulo provavelmente afetará vários outros lugares do software.

Abaixo seguem mais algumas desvantagens desta arquitetura, quando se tem uma grande aplicação monolítica:

- Códigos monolíticos grandes intimidam desenvolvedores, principalmente os que são novos nos times, uma vez que esses códigos são, na maioria das vezes, difíceis de se entender e modificar.
- Códigos grandes demais compostos por muitas linhas podem sobrecarregar IDEs tornando-as lentas, o que pode diminuir a produtividade.
- Códigos muito grandes podem sobrecarregar também o *container web*, demorando muito no processo de inicialização e de implantação, o que também impacta na produtividade, uma vez que desenvolvedores precisam desperdiçar parte do seu tempo com espera.
- Implantação Contínua (*Continuous Deployment*) se torna difícil. Além da grande quantidade de tempo ocioso de espera que frequentes implantações podem gerar, a atualização de um único componente faz com que a aplicação inteira seja reimplantada. Há também aumento do risco de que componentes não atualizados falhem ao serem inicializados, o que pode desencorajar atualizações frequentes.
- Aplicações monolíticas podem ser escaladas apenas em uma única dimensão, horizontalmente (eixo X), utilizando diversas cópias da mesma que são acessadas através de um平衡ador de cargas, escalando assim a aplicação inteira. Isto impossibilita que os módulos ou componentes sejam escalados de forma independente, o que pode trazer desperdício de recursos, uma vez que cada uma das partes da aplicação possui diferentes demandas e necessidades. Por exemplo, algumas funções precisam mais de processamento e outras mais de memória. A Figura 47 ilustra as dimensões possíveis para tornar uma aplicação escalável.
- Dificuldade em coordenar o desenvolvimento entre vários times. Com o crescimento da aplicação é natural que ela se divida entre times focando em áreas funcionais como, por exemplo, interface com o usuário, gerenciamento financeiro, gerenciamento de estoque, gerenciamento de entrega, banco de dados etc. Portanto, é necessário que os times se esforcem para alinhar atualizações e implantações que possam afetar outros times.

- Dificuldade em adotar novas tecnologias e linguagens. É extremamente custoso, tanto em termos monetários quanto em termos de tempo, reescrever milhares de linhas de código utilizando uma nova linguagem, *framework* ou até mesmo um novo banco de dados. Ainda que essas novas tecnologias sejam consideradas melhores, esse tipo de mudança é de alto risco, podendo impactar grande parte do sistema. Por estas razões, na maioria das vezes, aplicações monolíticas tornam-se presas às tecnologias que foram escolhidas no início do desenvolvimento.
- Tolerância a falhas pode ser um desafio. S. Newman descreve em [76] que quando um sistema monolítico falha, toda aplicação para de trabalhar, tornando-se indisponível.

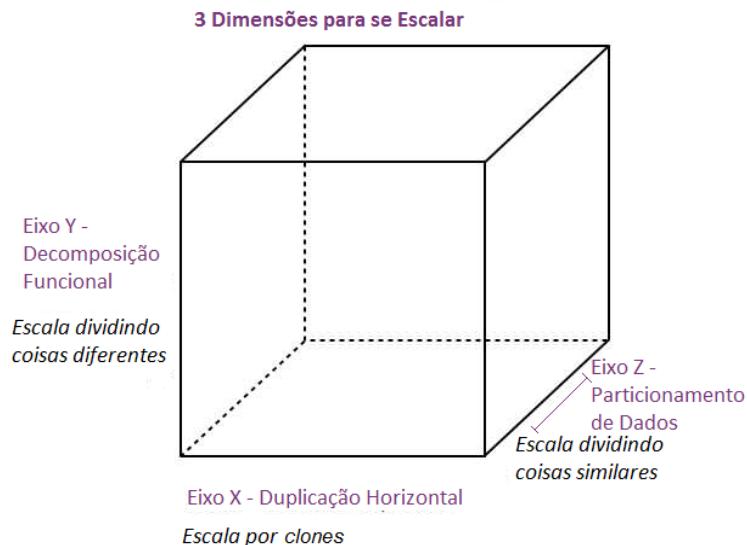


Fig. 47. Cubo Escala: modelo de escalabilidade em 3 dimensões. Fonte: [86]

Diante a estes desafios arquiteturais juntamente com a rápida evolução de tecnologias que surgem a todo momento, outras formas de se construir um sistema vêm sendo adotadas. A arquitetura baseada em micros-serviços (*Microservices*) surge neste panorama. A sessão seguinte tratará desta arquitetura incluindo suas vantagens e características.

A arquitetura baseada em micros-serviços, características e vantagens em relação à arquitetura monolítica.

Em [76], o autor define de forma resumida *Microservices* como serviços pequenos e autônomos que trabalham juntos. Em [74], os autores acrescentam que cada um dos serviços deste conjunto rodam em seus próprios processos e se comunicam através de mecanismos leves tanto síncronos, geralmente através de *Representational State Transfer* (REST), quanto assíncronos por barramento de mensagens, tais como RabbitMQ ou ZeroMQ. (Outra tecnologia usada para esse tipo de barramento é o Dbus. Uma apresentação oral sobre Dbus, feita por aluno dessa disciplina, será apreciável. Como usar, por exemplo, o Dbus no Qt ?) Eles são implantados e escalados independentemente, bem como possuem fronteiras ou limites bem definidos, podem ser escritos em

diferentes linguagens, utilizam diferentes recursos para armazenamento de dados e podem ser geridos por times distintos. A Figura 48 mostra a mesma aplicação apresentada na Figura 46 desenhada em uma arquitetura baseada em micros-serviços.

Para [74] não existe de fato uma definição formal para a arquitetura baseada em micros-serviços, o que existe são atributos comuns que foram percebidos em aplicações que seguem este estilo arquitetural. Não é obrigatório encontrar todas essas características em todos os sistemas que seguem esta arquitetura, mas a maioria delas estarão presentes.

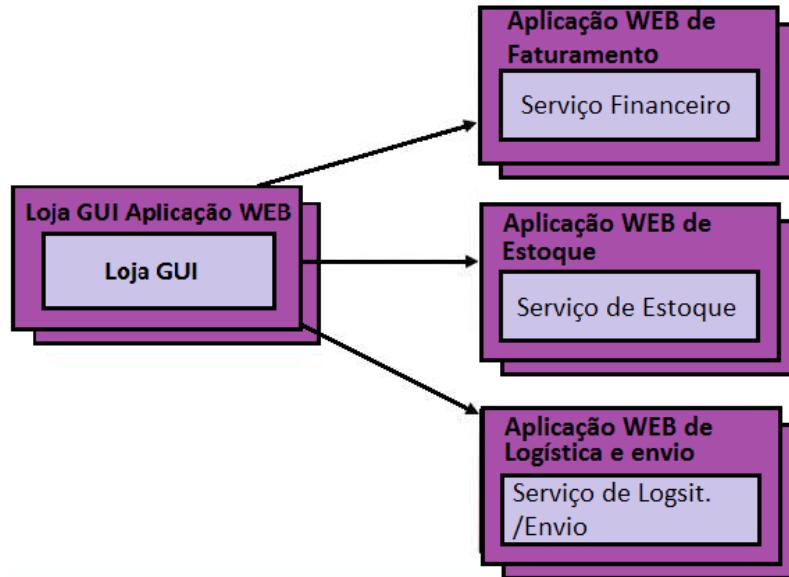


Fig. 48. Aplicação Web Baseada em uma Arquitetura *de* micros-serviços. Fonte: [86]

Abaixo serão citadas vantagens e características que são inerentes à arquitetura baseada em micros-serviços e que fazem com que esta arquitetura seja diferente:

- Componentização via pequenos serviços independentes com limites e responsabilidades bem definidas que sejam desacoplados e coesos seguindo o Princípio da Responsabilidade Única (*Single Responsibility Principle - SRP*) cujo lema é “juntar aquelas coisas que mudam pela mesma razão e separar aquelas que mudam por razões diferentes” em [81].
- Serviços são divididos em pequenos times por capacidades de negócios e estes são responsáveis por todas as camadas de tecnologia presentes na aplicação, englobando interface com o usuário, persistência de dados, lógica de negócios, etc. Então os times trabalham de forma multifuncional ao invés de times que são separados por camadas de tecnologias, conforme ilustra as Figuras 49 e 50. Além de que times menores trabalhando em uma base de código menor tendem a serem mais produtivos, afirma [76].
- Serviços pequenos possuem uma base de código menor, são mais fáceis de se desenvolver, de se manter e de serem entendidos por novos integrantes de times. Não sobrecarregam IDEs e nem o *container web*, contribuindo assim, com a alta produtividade dos desenvolvedores.

- Serviços autônomos que possam ser alterados e implantados rápida e independentemente de outros serviços, sem afetar seus consumidores e sem a necessidade de que a aplicação inteira tenha que ser reimplantada, tornando o processo de implantação muito mais rápido e possibilitando a prática de Implantação Contínua (*Continuous Deployment*).
- Conceito de produto ao invés de projeto. Times se tornam responsáveis por todo o ciclo de vida do produto incluindo o suporte em produção inspirando-se na Amazon com o lema "*you build, you run it*", cuja a tradução seria “você constrói, você executa”, referenciando à responsabilidade do desenvolvedor no suporte ao software em produção, visto em [82]. Aumentando assim o foco sobre qualidade do código que é entregue à produção, uma vez que qualquer problema que ocorra, os próprios desenvolvedores serão responsáveis por consertar.
- Comunicação inspirada no estilo Unix Clássico: receber requisição, processar e responder. Utilizando protocolos simples síncronos ou assíncronos utilizados na Web: HTTP/REST (requisição e resposta) ou *Advanced Message Queuing Protocol* - AMQP e *Simple (or Streaming) Text Orientated Messaging Protocol* – STOMP (baseado em eventos).
- Serviços podem ser construídos utilizando tecnologias diferentes de acordo com as necessidades de cada um deles.
- A adoção ou mudança para novas tecnologias ou melhores implementações são mais fáceis de administrar, produzem menos impactos e se tornam menos arriscadas quando realizadas em um escopo menor. A fonte [76] acrescenta que times que trabalham com a abordagem de micros-serviços sentem-se confortáveis quando é preciso reescrever um serviço ou até mesmo excluí-lo se este não é mais necessário.
- Isolamento de falhas e resiliência. Aplicações que utilizam serviços como componentes devem se organizar de forma que eles se tornem tolerantes a falhas. Quando acontecer uma falha em um serviço os outros serviços não serão afetados e continuarão processando requisições. Em [85] cita-se o Spotify como exemplo em que os desenvolvedores constroem seus sistemas assumindo que serviços podem falhar a qualquer momento.
- Decentralização de banco de dados. Cada serviço pode ter seu banco de dados exclusivo ao invés de utilizar um único banco de dados para toda a aplicação. Cada área pode ter um modelo de dados de domínio compartilhado entre diferentes contextos porém com representações internas que façam mais sentido para cada um dos serviços, como explica [76], além de se poder escolher formas de armazenagem que possam satisfazer as diferentes necessidades de cada serviço (bancos de dados relacionais, orientado a grafos, NoSQL. etc) aumentando assim a autonomia dos mesmos. Não compartilhar tabelas entre serviços também aumenta a coesão e diminui o acoplamento entre eles, uma vez que alterações feitas em tabelas não afetarão o modelo de dados de outros serviços e alterações não precisarão ser replicadas em vários locais. A Figura 51 ilustra um banco compartilhado de uma aplicação monolítica e um banco decentralizado em uma arquitetura de micros-serviços.
- A complexidade adquirida pela arquitetura distribuída dos micros-serviços, traz forte

indicação para utilização de técnicas para a automação da infraestrutura e processos de desenvolvimento bem como o compartilhamento de ferramentas *open source* que auxiliam desenvolvedores na criação de artefatos, administração do código, monitoramento de serviços e *logs* como é o caso do pacote *open source* da Netflix. Com o Netflix Open Source Software Center, a empresa compartilha códigos úteis, testados em produção que ajudam desenvolvedores a resolverem problemas de forma similar a eles e deixa-os abertos para que outras abordagens possam ser adotadas e compartilhadas. Uma pesquisa sobre o Netflix Open Source Software Center e apresentação sobre o mesmo, seria um bom trabalho a ser feito por aluno dessa disciplina.

- » Cada serviço pode ser escalado independentemente de outros serviços de acordo com a sua necessidade e a demanda de cada funcionalidade, aplicando a escala no eixo Y como ilustra a Figura 52.

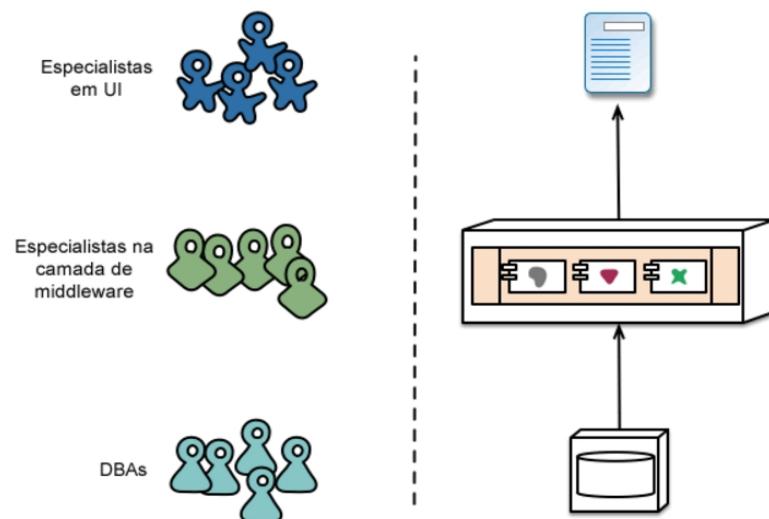


Fig. 49. Ilustração de times divididos por camadas de tecnologia. Fonte: [74]

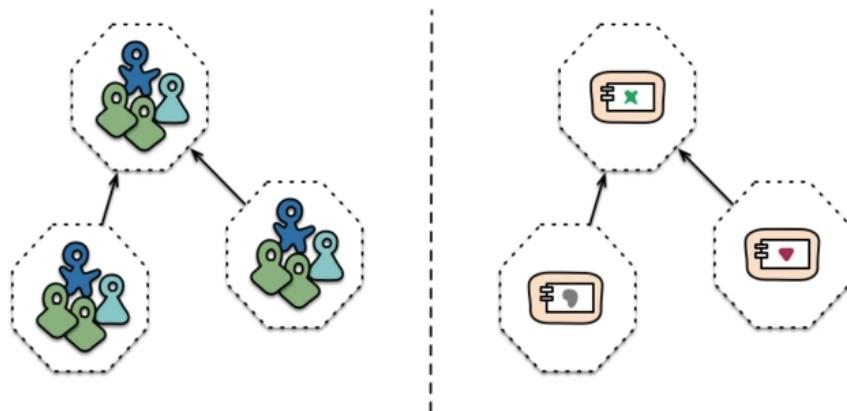


Fig. 50. Ilustração de times multifuncionais divididos por área de negócio. Fonte: [74]

Embora têm sido observadas experiências positivas em times e empresas que utilizam desta

arquitetura, ainda é muito cedo para afirmar que a arquitetura de micros-serviços se estabelecerá como o padrão de arquitetura de software do futuro. Em [74] está descrito que os verdadeiros efeitos e implicações da escolha de uma arquitetura se tornam notórios apenas anos e até décadas depois da aplicação da mesma. Assim, se for pesquisado algum caso de sucesso com micros-serviços, ocorrido na indústria de TI, e se tal caso for demonstrado em detalhes, isso será muito relevante como um trabalho a ser apresentado por aluno dessa disciplina.

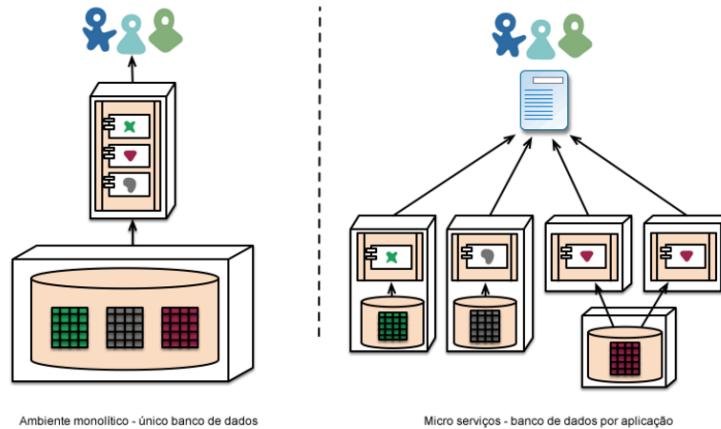


Fig. 51. Ilustração de uma aplicação monolítica utilizando um banco compartilhado e uma aplicação em micros-serviços utilizando banco de dados decentralizado. Fonte: [74]

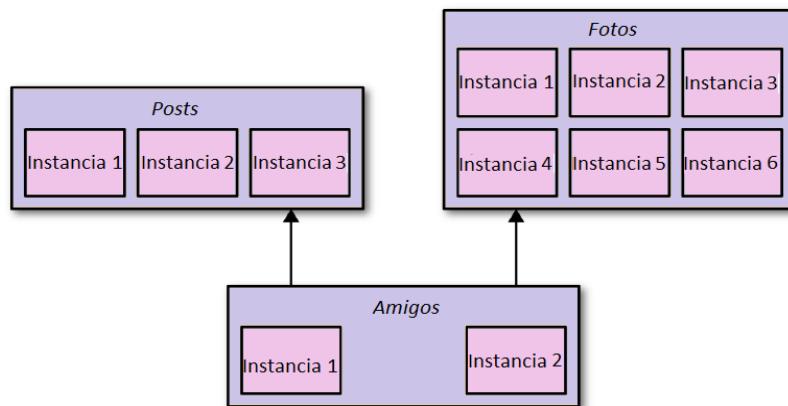


Fig. 52. Ilustração de escalonamento feito utilizando eixo Y. Escalando serviços de acordo com a demanda. Fonte: [76]

Portanto, é necessário que diversos sistemas ainda sejam construídos baseados em micros-serviços e que estes atinjam idade suficiente para assim poder julgar com propriedade a real maturidade e qual eficiência de modularização deste padrão.

A) Micros-serviços e SOA.

Apesar de micros-serviços ser um termo que passou a ser ouvido recentemente, a ideia em si não é nova e de acordo com [74] ela se remete pelo menos aos princípios de *design* do Unix. Alguns adeptos da arquitetura consideram micros-serviços como uma forma de *Service Oriented*

Architecture - SOA, arquitetura que surgiu no início deste século, porém feita de forma certa, como é o caso da Netflix que em [79] descreve micros-serviços como “SOA bem feito” (*fine grained SOA*). Todavia, outros defensores e discussões em *workshops* e congressos de arquitetura de *software* chegaram à conclusão de que micros-serviços era o nome mais apropriado diante das características comuns presentes neste estilo arquitetural, rejeitando totalmente o rótulo SOA, uma vez que para eles esse termo pode significar muitas coisas diferentes como explicado em [80].

O autor de [76] acrescenta que o novo estilo de arquitetura auxilia no aspecto de evitar armadilhas encontradas em inúmeras implementações de SOA, visto que ele promove a integração de tecnologias novas juntamente com técnicas que afloraram ao longo da última década. Para [104], aplicações via componentes desacoplados realmente não é algo novo, o ponto é que micros-serviços é mais claro do que SOA em suas definições de características, como o de proporcionar um *framework* do mundo real que satisfaça os requerimentos de arquitetura das aplicações modernas. E complementa que os estilos de comunicação e processamento leves é mais um dos atributos que distingue micros-serviços de SOA.

Além do fator comercial e tecnologias *Web Service Specification* (WS-*) em [100], *Enterprise Service Bus* (ESB) em [101] e o modelo canônico (*Canonical Schema*) em [103], os quais são rejeitados pela abordagem de micros-serviços, muito do convencional conhecimento em torno de SOA não ajuda a entender como dividir algo grande em partes menores ou qual tamanho deve ser considerado grande demais (entretanto a nossa disciplina nessa pós-graduação, de determinação de interfaces de serviços, nos ajudou com esta questão). SOA não aborda maneiras práticas de garantir que um serviço não se torne excessivamente acoplado. Já a arquitetura de micros-serviços surgiu de práticas constatadas no mundo real fazendo com que fosse possível compreender melhor esta arquitetura e fazer SOA corretamente, reforça [76]. E tais práticas suportam as características vantajosas enumeradas anteriormente nesse documento.

Além destas características citadas acima, o autor em [103] faz uma comparação superficial na tabela 1 pontuando as principais diferenças para ele entre as duas arquiteturas.

	SOA	Micros-serviços
Tamanho de componente	Grandes partes da lógica de negócio por componente	Pequenas partes da lógica de negócios por componente
Acoplamento	Geralmente acoplamento baixo	Sempre acoplamento baixo
Estrutura Organizacional	Qualquer um	Pequenos e dedicados times multifuncionais
Governança	Foco em governança	Foco em governança

	centralizada	descentralizada
Metas principais	Garantir interoperabilidade entre aplicações	Implantar novas funcionalidades e escalar aplicações rapidamente

Tabela. 1.Comparação SOA e Micros-serviços. Fonte: [103]

Conceitos e padrões largamente discutidos e difundidos atualmente, boas práticas e formas melhores de se construir e implantar sistemas como *Domain Driven Design* (DDD), Entrega Contínua (*Continuous Delivery*), virtualização por demanda, automatização de infraestrutura, times pequenos e autônomos e sistemas em escala são relativos ao universo ao qual emergiu a arquitetura de micros-serviços. Não são invenções ou previsões que ainda não se concretizaram, mas sim referem-se aos fatos do mundo real, se estabelecendo desta forma, como tendência e padrão para o desenvolvimento de aplicações, explica [76].

B) Desvantagens da arquitetura baseada em micros-serviços.

Apesar de propor soluções para diversos problemas encontrados na abordagem monolítica, a arquitetura de micros-serviços traz também algumas desvantagens, são elas:

- ▀ Times são obrigados a lidar com a complexidade presente nos sistemas distribuídos.
- ▀ IDEs e ferramentas são voltadas para a construção de aplicações monolíticas, não dando muito suporte para sistemas distribuídos.
- ▀ Testar aplicações distribuídas é mais complexo. Como explica [83] é trivial testar uma API REST, por exemplo, de uma aplicação web monolítica, ou um único serviço individualmente. Porém no caso da arquitetura de micros-serviços, além do próprio serviço que está se testando, é necessário também que todos os serviços que este depende ou pelo menos "stubs" destes serviços estejam disponíveis também. Estes ambientes são difíceis de serem reproduzidos de forma consistente tanto para testes manuais quanto para testes automatizados e quando a assincronia e mensagens dinâmicas são adicionadas, o nível de dificuldade é ainda maior para se testar, complementa [97]. Comportamentos não esperados originados da interação entre os serviços são comuns em ambientes tão dinâmicos.
- ▀ Desenvolvedores devem implementar todos os mecanismos de comunicações entre os serviços (Invocação de chamadas remotas - *Remote Procedure Invocation* ou Mensageria) além da comunicação externa com clientes, bem como tratamento de casos de indisponibilidade ou lentidão de resposta do serviço.
- ▀ Gerenciar e manter a consistência de múltiplos bancos de dados é um desafio. Transações distribuídas são um problema para a arquitetura de micros-serviços tanto pelo Teorema de CAP, explicado em [91], quanto pelo fato de que muitos dos bancos de dados NoSQL utilizados hoje em dia não suportam este tipo de transação, como cita [83].

- Casos de uso que utilizam de vários serviços podem requerer atualizações em vários serviços dependentes e alinhamento com vários times explica [83]. Porém, casos assim, devem ser raros nesta arquitetura.
- Maior complexidade de implantação. Aplicações baseadas em micros-serviços geralmente consistem de muitos serviços, como exemplo, pode-se citar a Netflix que conta com cerca de 600 serviços como citado em [84]. Cada um destes serviços, quando escalados, podem possuir inúmeras instâncias. Toda esta quantidade de serviços precisam ser configurados, implantados, escalados e monitorados. Mecanismos para monitoramento, bem como para mapear a localização e dependências com outros serviços, devem ser implementados. Sendo assim, é necessário bons métodos de controle do processo de implantação dos sistemas desenvolvidos e um alto nível de automação de todo o processo, explica [83].
- Aumento de consumo de memória. Se cada serviço roda em sua própria JVM, por exemplo, ou se ainda cada um deles rodar em uma máquina virtual própria a carga de memória necessária para a execução destes serviços, é ainda maior, explica [86].
- Desenvolver arquiteturas distribuídas aumentam o tempo de desenvolvimento, o que não é muito atraente para aplicações “*startups*”, cujo o maior objetivo é evoluir rapidamente o modelo de negócio acompanhado com a aplicação. O autor em [97] ressalta que é necessário fazer um grande investimento em desenvolvimento de *scripts* e implementações para gerenciar os processos antes de iniciar a escrever uma única linha de código que realmente entregará valor de negócio.
- Aumento de latência. Fazer chamadas para um único processo em uma aplicação monolítica é diferente de chamar vários serviços que chamam outros serviços, isso faz com que a latência cresça em cada uma dessas chamadas, cita [85].
- A rede não é confiável. Sistemas distribuídos estão estreitamente relacionados à rede. O autor de [76] reforça que redes podem e irão falhar, elas podem falhar rápido ou lentamente, além de poderem mal formar seus pacotes. Essas falhas devem ser consideradas e tratadas para evitar indisponibilidades e más experiências aos usuários
- Duplicação de código e dados. Para garantir o baixo acoplamento entre os serviços, que é um dos princípios fundamentais da arquitetura baseada em micros-serviços, muitas vezes, se faz necessário a convivência com a duplicação de código e de alguns dados. Como nos casos de ter um único banco por serviço e evitar códigos comuns entre serviços como bibliotecas compartilhadas. Em [76], o autor reforça que os males provocados por auto acoplamento são muito piores do que os causados pela duplicação de código ou dados.
- Se uma grande aplicação monolítica é composta de uma dúzia de módulos ou partes, usando micros-serviços, este número deve expandir de 3 a 5 vezes em número de serviços, fazendo com que manutenção e gerenciamento seja um grande desafio, de acordo com [90].

Além desses pontos, o autor em [98] cita que deve haver uma mudança de cultura em relação à colaboração entre as áreas de desenvolvimento e operação. Esta nova cultura a ser introduzida é chamada de DevOps e é aprofundada em [99]. Mudanças de hábitos e cultura são pontos difíceis de

se concretizar, principalmente em empresas muito grandes ou equipes mais antigas, que já tem sua forma de trabalho enraizada. O assunto DevOps é passível de apresentação oral por aluno dessa disciplina, sendo muito apropriado a explicação detalhada da afirmação no parágrafo seguinte.

Um alto grau de qualidade em DevOps é necessário para se manter uma arquitetura em micro-serviços explica [97], e ele complementa que desenvolvedores com forte perfil DevOps são raros de se encontrar, o que torna este desafio ainda maior.

c) *Quando utilizar uma arquitetura baseada em micro-serviços faz sentido.*

Questões como quando faz sentido utilizar esta arquitetura ou não e como e quando deve-se decompor uma aplicação monolítica são comuns. O autor em [83] cita que aplicações monolíticas fazem sentido apenas para aplicações simples e leves e apesar de toda a complexidade de implementação e desvantagens envolvidas em arquiteturas baseadas em micros-serviços, esta é a melhor opção para aplicações maiores e mais complexas. Mas, como saber se uma aplicação já atingiu tamanho suficiente para ser convertida de monolítica a micros-services? Existe alguma metodologia para responder essa questão em qualquer projeto? Uma pesquisa para responder essas duas questões, explicada com exemplos, mesmo que fictícios, seria um ótimo trabalho a ser apresentado oralmente por aluno dessa disciplina. Como julgar se um sistema já está complexo demais para continuar como monolítico?

Se ainda não existe uma metodologia para responder estas questões, então alguma poderia ser criada, o que daria um ótimo TCC passível de publicação em revista.

De acordo com [88], a base para saber se a arquitetura baseada em micros-serviços deve ou não ser usada é a complexidade do sistema. A abordagem de micros-serviços gira em torno da manipulação e gerenciamento de um sistema complexo, portanto a adoção desta abordagem introduz por si só, seu próprio conjunto de complexidade, pois uma vez adotada esta arquitetura é preciso trabalhar também com implantação automatizada, monitoramento, tratamento de falhas, consistência e vários outros fatores e questões trazidas por um sistema distribuído.

O autor em [97] comenta que micros-serviços é uma boa ideia na prática, porém todo tipo de complexidade vem a tona quando esta arquitetura se encontra com a realidade.

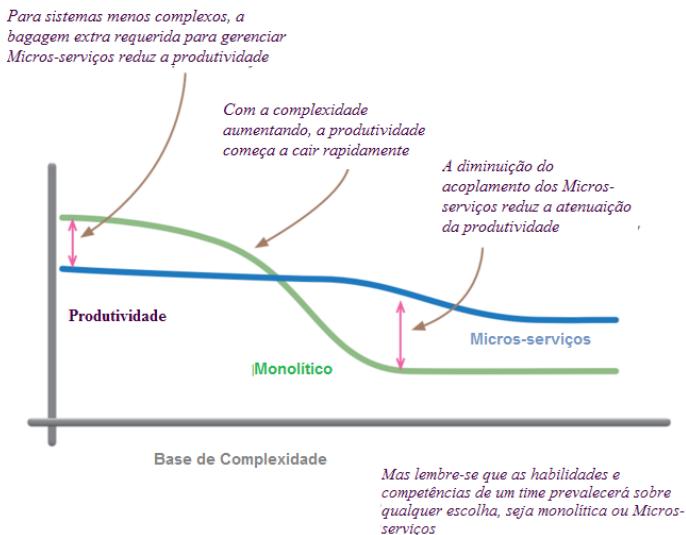


Fig. 53. Ilustração da relação Produtividade vs Complexidade comparando aplicações monolíticas e micro-serviços. Fonte: [88]

A Figura 53 ilustra a relação entre Produtividade vs Complexidade comparando sistemas monolíticos e micros-serviços. Pode-se perceber que utilizando micros-serviços em um sistema com baixo nível de complexidade, a curva de aprendizado necessária para possuir a bagagem de conhecimento exigida para gerenciar e construir este tipo de sistema, faz com que exista uma redução de produtividade. Já em uma arquitetura monolítica quando a complexidade aumenta, a produtividade diminui bruscamente, diferentemente da arquitetura baseada em micros-serviços com a qual não se tem uma queda acentuada de produtividade, neste caso.

Por este gráfico é possível perceber então a estreita relação entre complexidade e o uso da arquitetura baseada em micros-serviços.

Esta complexidade se origina de diversas fontes, dentre elas, lidar com times muito grandes, alta diversidade de modelos de interação com o usuário, independência entre as funcionalidades e módulos e escalonamento, mas o fator fundamental e mais importante é a dimensão do sistema, que quando se tratando de uma unidade monolítica, um grande gargalo é encontrado no momento de modificar e implantar esta aplicação. Sendo assim, [88] ressalta que ao menos que um sistema encontre com tais complexidades, o sistema deve se manter simples o suficiente para evitar a necessidade do padrão de micros-serviços.

D) Iniciar um sistema utilizando uma arquitetura em micros-serviços VS decompor um sistema monolítico.

O autor de [76] defende que decompor prematuramente uma arquitetura monolítica em micros-serviços pode ser custoso e dolorido, principalmente se o domínio/negócio ao qual pertence a aplicação ainda não é bem conhecido, embora, em muitos aspectos, decompor um código monolítico já existente, seja muito mais fácil, do que iniciar a construção de uma aplicação utilizando micros-serviços desde o início. Tanto a abordagem de iniciar uma aplicação desde o início usando esta arquitetura, quanto decompor uma arquitetura monolítica já existente, têm suas vantagens e desvantagens.

No início da construção de uma aplicação não existe a necessidade de uma arquitetura em micros-serviços, os problemas encontrados em uma arquitetura monolítica ainda não são visíveis ou perceptíveis, além de que o desenvolvimento de aplicações baseadas em micros-serviços apoia-se em um processo consideravelmente mais lento, tornando esta arquitetura menos atraente para quem inicia a construção de uma aplicação. Por outro lado, depois de um tempo, quando as desvantagens e problemas de uma arquitetura monolítica estiverem presentes, fazer a refatoração de toda a aplicação, dependendo do tamanho e complexidade da mesma, pode ser bastante árduo e ariscado.

Levando em consideração este cenário, [87] explica que foi percebido um padrão comum entre os times ou empresas que adotaram a arquitetura baseada em micros-serviços. E neste padrão pode-se perceber que a maioria dos casos de sucesso foram de histórias que começaram com uma arquitetura monolítica que se tornou grande demais e foi decomposta. (Essa é uma boa dica para quem decide propor uma metodologia para decidir quando usar micros-serviços)

Outro ponto que se observou foi que quase todos os casos que construíram um sistema baseado em micros-serviços iniciando do zero, acabaram com grandes problemas. Este padrão observado levou a conclusão de que não se deve iniciar um novo projeto diretamente em uma arquitetura de micros-serviços, mesmo sendo certo que esta aplicação será grande o suficiente para utilizá-la um dia. A Figura 54 ilustra este padrão.

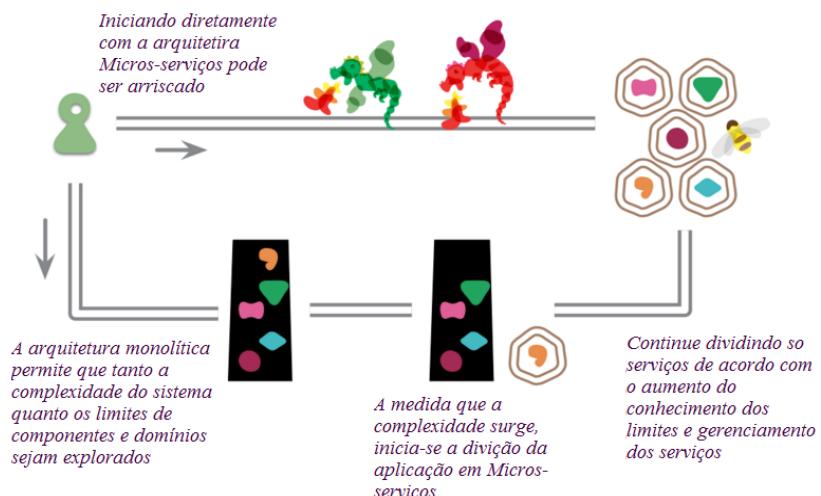


Fig. 54. Ilustração do fluxo de uma aplicação iniciada em micros-serviços e outra iniciada como monolítica e decomposta posteriormente. Fonte: [87]

As principais razões para esta abordagem é que quando se inicia a construção de uma aplicação não se tem a certeza de quão útil ela será para os usuários e o melhor a se fazer para se ter ideia da utilidade desta aplicação é construir uma versão simples (protótipo) que pode ser um sistema monolítico. Outra razão é que se construindo primeiro uma aplicação monolítica torna-se possível descobrir mais facilmente quais são os contextos e fronteiras corretos pertencentes à aplicação e isto facilita muito a decomposição em serviços com limites bem definidos e granularidade fina, garantindo baixo acoplamento e coesão e que assim, os benefícios e vantagens desta arquitetura sejam alcançados.

Já para [96] iniciar com uma aplicação monolítica pode não ser uma boa ideia, pois para o autor, o início da construção de um sistema é o momento correto onde se deve pensar em como modelar e dividi-lo em partes menores. Para ele, esta divisão se torna mais difícil em cima de um sistema monolítico já existente, embora [96] concorde que o domínio do sistema que se está construindo deve ser muito bem conhecido antes de tentar dividi-lo e é preciso ter certeza de que a aplicação será grande o suficiente para justificar a utilização desta abordagem.

O autor em [96] complementa que quando se constrói um sistema monolítico bem estruturado e modularizado, como se fossem diversos micros-serviços escondidos, prontos para serem extraídos, provavelmente a necessidade de utilizar micros-serviços não existirá, embora estas não sejam as únicas razões para se trabalhar com tal estilo arquitetural. O autor em [74] complementa que nem sempre boas interfaces entre processos internos são boas interfaces de serviços.

Conectar partes da aplicação que não deveriam se conhecer e acoplar fortemente as que deveriam, pode ser evitado seguindo regras e estabelecendo limites claros de cada módulo dentro de uma aplicação monolítica e micros-serviços apenas fazem com que esses erros sejam mais difíceis de serem atingidos. Porém, geralmente não é o que se encontra em sistemas com padrões monolíticos. A figura 55 representa a expectativa versus realidade de sistemas utilizando a arquitetura monolítica.

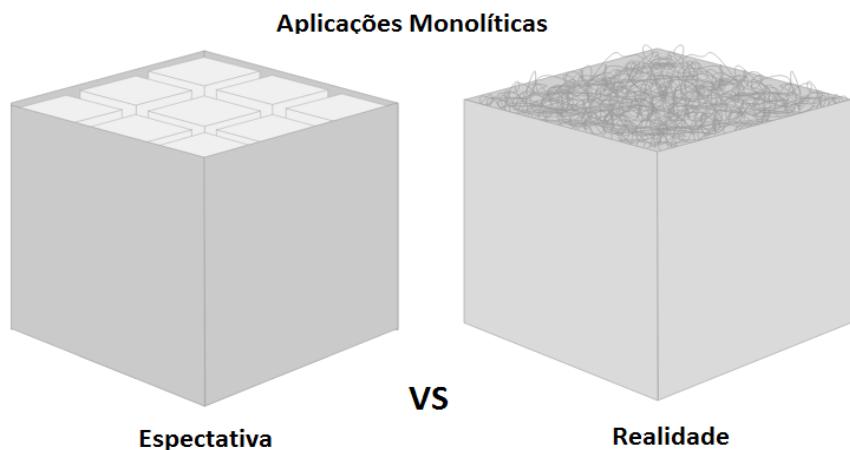


Fig. 55. Expectativa vs Realidade de uma aplicação construindo em uma arquitetura monolítica. Fonte [96]

Geralmente, evitar conexões desnecessárias e alto acoplamento em aplicações monolíticas é um grande desafio, embora seja perfeitamente possível existir módulos com limites sólidos e bem definidos, mas isso requer um alto grau de disciplina.

Dependência da plataforma, compartilhamento de bibliotecas, objetos de domínio, modelos de persistência e meios de comunicação disponíveis apenas no mesmo processo são fatores que dificultam extremamente dividir uma aplicação monolítica já existente em micros-serviços, segundo [96].

De qualquer forma é importante colocar os domínios dentro de partes separadas e independentes, tanto utilizando a abordagem de iniciar com arquitetura de micros-serviços ou com a monolítica. Às vezes para determinados casos, pode ser preferível a abordagem monolítica primeiramente e para outros não.

Muitas grandes e conhecidas empresas tem migrado suas aplicações monolíticas para micros-serviços, tornando-se casos de sucesso. Como exemplos pode-se citar: Netflix, Amazon, Spotify, Walmart, entre outras.

Citado em [85], Walmart afirma que a arquitetura baseada em micros-serviços é a chave para estar a frente das grandes demandas do mercado e se manter competitivo. Spotify indica como um dos benefícios trazidos pela arquitetura baseada em micros-serviços, a possibilidade da existência de um grande número de serviços indisponíveis ao mesmo tempo sem que os usuários percebam isto, não contribuindo com experiências ruins ao usar o aplicativo. Para a Amazon, micros-serviços possibilitou a criação de uma arquitetura altamente desacoplada com serviços interagindo independentemente de outros serviços.

Porém como mencionado por [89], nenhuma arquitetura é uma “bala de prata” que resolve todos os problemas e funciona para todos os casos. Todas elas têm seus problemas e desvantagens e devem ser analisadas conforme seu contexto de aplicação e suas características e objetivos.

Padrão de linguagem para a arquitetura de micros-serviços

Uma boa forma de discutir sobre tecnologias é através de padrões, explica [89]. Padrões são soluções reutilizáveis para problemas que ocorrem em um determinado contexto. E alguns autores já catalogaram um conjunto de padrões que devem ser utilizados e considerados na construção de sistemas baseados na arquitetura de micros-serviços.

Estes padrões podem referir-se a soluções alternativas para o mesmo problema, podem ser também padrões que são soluções de problemas introduzidos pela aplicação de outros padrões, ou ainda, especializações de outros padrões.

A Figura 56 ilustra o padrão de linguagem sugerido por [86] para micros-serviços, contendo uma coleção de padrões que ajudam a entender a complexidade de implementar este tipo de sistema. O autor cita que esta é uma versão inicial e que novos padrões tem sido adicionados nos últimos meses e cujo o plano é expandir, complementando com mais padrões.

O autor em [86] categoriza os padrões utilizando as seguinte seções:

- ▀ Padrões Núcleo
 - Padrão Monolítico
 - Padrão de Micros-serviços
- ▀ Padrões de Implantação
 - Múltiplas Instâncias de Serviço por *Host*
 - Única Instância de Serviço por *Host*
 - Instância de Serviço por Máquina Virtual
 - Instância de Serviço por *Container*

🎬 Padrões de Comunicação

- API Externas
 - API *Gateway*
 - Backend for Frontend
- Descobrimento de Serviços
 - Descobrimento do Lado Cliente
 - Descobrimento do Lado Servidor
 - Registro de Serviços
 - Auto Registro
 - 3rd Party Registration
- Estilo de Comunicação
 - Mensagens
 - Remote Procedure Invocation

🎬 Padrões de Gerenciamento de Dados

- Banco de Dados por Serviço
- Banco de Dados Compartilhado
- Arquitetura Orientada a Eventos
- Fornecimento de Eventos (*Event Sourcing*)
- *Command Query Responsibility Segregation* - CQRS
- Transaction Log Tailing
- *Triggers* de Bando de Dados
- Eventos de Aplicativos

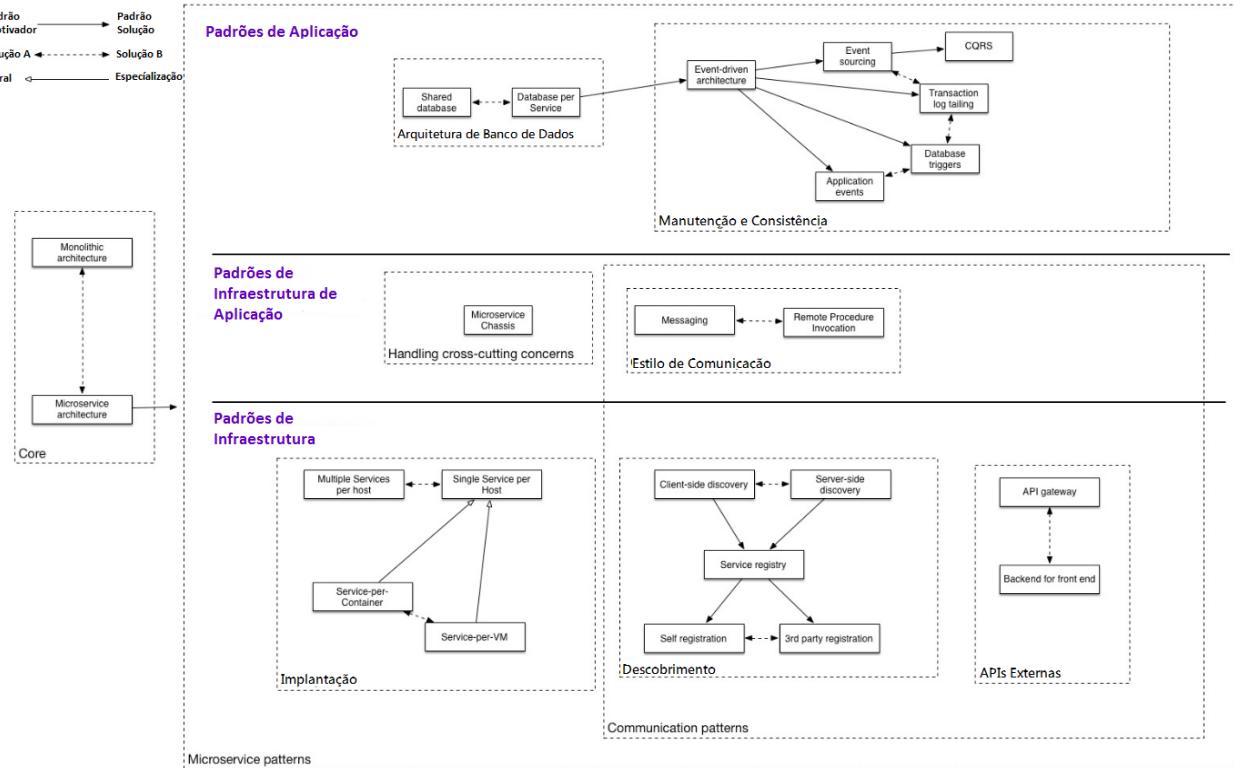


Fig. 56. Padrão de linguagem inicial para a arquitetura baseada em micro-serviços. Fonte: [86]

Este capítulo não visa explicar ou aprofundar estes padrões, o objetivo desta sessão é apenas citá-los. Deixando como sugestão para trabalhos futuros um estudo mais aprofundado sobre estes padrões e suas aplicações.

Conclusões

No cenário atual relacionado ao desenvolvimento e arquitetura de softwares, o padrão arquitetural baseado em micro-serviços vem recebendo bastante atenção, sendo mencionado em diversos artigos, blogs, conferências e discussões em geral. Aparecendo inclusive no pico de expectativas exageradas nos estudos do Instituto Gartner - [Gartner Hype Cycle](#), em [92], [93] e [94].

Desta forma, muito se ouve sobre os benefícios da aplicação desta arquitetura e casos de sucesso que utilizam desta. Porém, nem sempre este estilo arquitetural se encaixa no perfil da aplicação que será construída, fazendo com que muitas vezes a utilização deste padrão arquitetônico seja frustrante.

A arquitetura de micro-serviços possui inúmeras vantagens em relação à arquitetura monolítica, porém possui também diversos pontos desvantajosos em relação àquela, os quais introduzem maior complexidade ao processo de desenvolvimento e implantação de um sistema, exigindo diversas implementações de mecanismos que auxiliem em monitoramento, descobrimento de serviços, comunicação, armazenamento de dados, entre outros fatores, além de um alto grau de automatização de todo o ciclo de vida da aplicação.

A grande maioria dos sistemas podem ser construídos como uma aplicação monolítica utilizando uma modularização bem definida, sem que estes módulos sejam transformados em serviços, ainda que para isso seja necessário um alto grau de disciplina por parte dos desenvolvedores. Então se um sistema pode ser simples o bastante para evitar a necessidade da arquitetura em micros-serviços, ele deve se manter desta forma.

Sendo assim, esta arquitetura não deve ser utilizada para aplicações simples e com baixo grau de complexidade. Do mesmo modo não é aconselhável que um software se inicie utilizando esta arquitetura, e sim que ele seja decomposto posteriormente, após a maturidade do desenvolvimento da aplicação juntamente com a compreensão dos contextos pertencentes a esta, bem como com a definição correta de seus limites e entendimento do processo de negócio e seus domínios, ainda que esta abordagem possa ser bastante trabalhosa, principalmente se este sistema monolítico for altamente acoplado e com fraca definição de limites entre seus módulos internos.

O ideal é que se existir forte indício de que esta aplicação monolítica alcançará o nível de complexidade para ser decomposta em um sistema de micros-serviços, que esta seja construída cuidadosamente de forma mais coesa e desacoplada possível, considerando a extração futura desses módulos em micros-serviços.

Apesar de todas estas indicações é importante analisar individualmente cada caso, este capítulo não visa ditar uma regra para todos os sistemas construídos baseados em micros-serviços, mas sim expor fatores importantes a se considerar no momento desta decisão, pois para cada aplicação estes aspectos terão um peso diferente. Fatores como maturidade e capacidade do time também são impactantes, times medianos construirão sistemas medianos, independente do estilo arquitetural utilizado. A arquitetura baseada em micros-serviços exige uma bagagem de conhecimento e nestes casos times mais experientes e habilidosos contarão positivamente no momento de lidar com toda a complexidade e problemas relacionados a esta arquitetura e a sistemas distribuídos em geral.

Neste capítulo foram levantados os pontos positivos e negativos de utilizar a arquitetura baseada em micros-serviços, além de diversos outros fatores e questões que giram em torno desta abordagem, comparando à arquitetura monolítica, com o objetivo de auxiliar na decisão de adotar ou não este estilo arquitetural na construção de um sistema. Também foi abordado sua relação com SOA, além de citados os principais padrões de *design* existentes para a arquitetura baseada em micros-serviços até o momento da conclusão deste capítulo. Como sugestão para futuros trabalhos, podem ser estudados estes padrões de forma mais profunda, explicando e ou exemplificando o papel de cada um deles neste estilo arquitetural.

Como trabalho de conclusão de curso para essa pós-graduação (TCC), um projeto do ICC poderia ser desenvolvido e implementado utilizando micros-serviços. O resultado poderia ser documentado em artigo. Para tal, o(a) aluno(a) pode, por exemplo, guiar um projeto do ICC com duas formas simultâneas: monolítica e com micro-serviços, a partir de um ponto no tempo do desenvolvimento do mesmo. É claro que não daria para fazer isso sozinho(a). Os envolvidos no projeto teriam que concordar em ajudar (inclusive gerente) e seguir essas 2 linhas, dividindo a equipe em duas. Dessa forma, seria feita observação do andamento do projeto nas duas vertentes, coletando dados de tempo de programação de requisitos, dificuldades vencidas em cada vertente, resultados alcançados,

etc. O aluno teria que tabular os resultados vistos em forma de tempo gasto para alcançar cada meta do ciclo de vida do desenvolvimento em cada vertente. Em seguida, os resultados seriam comparados, para concluir qual arquitetura foi a melhor para resultados melhores. Com tal comparação, poderia ser criada uma sistemática na prática de como decidir qual arquitetura usar em próximos projetos. Os resultados teriam que ser embasados com afirmações de autores da literatura da área, com referências a livros sobre o assunto. O resultado final deveria mostrar as medidas quantitativas de tempo e elas deveriam provar que a conclusão é verdadeira e que então a sistemática proposta estaria mesmo bem fundamentada com resultados na prática.

Em 1/10/2015 a empresa Sensedia estava contratando profissionais com conhecimentos em microservices. A quem vá fazer uma apresentação oral sobre o assunto micro-services, é fortemente recomendável ver o vídeo [104]. Esse vídeo é da Sensedia e mostra vários aspectos de microservices. Ele também mostra duas ferramentas para o monitoramento desses serviços, que são a New Relic e a Zabbix. Com o monitoramento, fica possível ver a execução dos microservices. Essas ferramentas mostram o consumo de memória de cada um, qual é mais demandado no tempo, qual gasta mais CPU, etc. Mostram também o tempo de resposta de cada serviço. Essas ferramentas poderiam ser mostradas oralmente, por aluno dessa disciplina, por exemplo na forma de um mini tutorial. Além desse vídeo, há outro muito conveniente a ser visto em [105]. Nesse segundo vídeo a prática de conversão de um sistema monolítico em micro serviços é mostrada mais detalhadamente. Ou seja, esse vídeo mostra bem todos os passos pelos quais uma empresa deve passar em tal conversão. Dá para perceber que esse aprendizado valioso, obtido na prática, pode levar de 2 a 3 anos. Mas, a teoria pode ser demonstrada antes disso.

Uma das dificuldades em trabalhar com micro serviços é a divisão de um banco de dados em outros menores, sendo que cada serviço pode ter o seu próprio banco. Nesse caso, manipular muitos dados em muitos bancos, para manter a consistência das informações entre os serviços pode requerer uma ferramenta adequada para tal. Uma ferramenta com esse propósito é o Facebook Presto, cujos usuários incluem Facebook, Netflix e Airbnb. Vale apenas apresentar o Facebook Presto oralmente durante a apresentação sobre micro-serviços, se desejado.

Finalmente, um bom material didático para aprendizado sobre micro serviços está no site [83]. Esse web site é o material mais completo sobre micro serviços, dentre os referenciados nessa apostila. O conteúdo dele por si só já é suficiente para a preparação de uma ótima apresentação oral sobre o assunto!!!

Boa sorte.

Capítulo VI - Realidade Aumentada

O que é a realidade aumentada?

A inserção de um conjunto de informação virtual na realidade à estende de tal forma que dizemos que ela está aumentada. Portanto, o aumento da realidade não significa aumento de tamanho, mas sim ganho de informação virtual adicional, que pode ocorrer através do uso de dispositivos eletrônicos, como *smartphones* e computadores. E mais recentemente através de *smart glasses*.

Se à realidade é acrescentada informação virtual (informação que não pode ser vista ou percebida na realidade, mas somente através de um dispositivo com tela e/ou fone, por exemplo), que pode ser som, texto, música ou imagem 3D, então a realidade se torna maior e nos dá mais informação relacionada com objetos do mundo real, o que pode ser muito útil. Por exemplo, através da tela de um *smartphone* podemos ver a fachada de um prédio real e além disso uma informação textual (não existente na fachada do prédio) dizendo que aquele prédio é um hotel. Bom, isso parece comum, apenas uma imagem e um texto. Mas, o interessante da realidade aumentada é que ela ocorre através de tecnologia que produz, em tempo real, o reconhecimento de uma imagem, para então causar a adição de informação virtual relacionada com tal imagem. Ou seja, com tecnologias de realidade aumentada pode-se fazer um aplicativo de *smartphone*, capaz de reconhecer a imagem de um objeto ou local, buscar informação sobre a coisa reconhecida e então produzir uma imagem 3D ou texto diretamente na tela do dispositivo, acrescentando então informação ao contexto real.

Resumidamente, é possível usar técnicas de realidade aumentada para produzir imagens, textos e sons em dispositivos eletrônicos, como PCs e *smartphones*, conteúdos tais que se relacionam visualmente, por exemplo, com imagens de objetos reais vistos nas telas desses dispositivos. Isso dá a sensação de que a realidade tem algo a mais que nos dá informação útil sobre o que está sendo visto.

Ao usar a realidade aumentada pode ser necessário que uma imagem seja capturada pela câmera do dispositivo. A imagem pode ser uma figura impressa no papel, ou a filmagem de um objeto real. No final das contas, um aplicativo de realidade aumentada poderá ter, como entrada, a imagem resultante daquilo que está sendo capturado pela câmera do dispositivo. Essas imagens são necessárias porque é através do reconhecimento delas que o aplicativo pode buscar informação sobre as mesmas. Sendo que tais informações podem estar num banco de dados local no dispositivo ou na nuvem, por exemplo. Portanto, as imagens para disparar a execução da realidade aumentada funcionam como chaves primárias de tabelas de bancos de dados, porque é através delas que é possível indexar a informação a ser recuperada. Essas imagens são chamadas de **marcadores**. Um marcador pode ser a imagem de um QR code, uma foto de um imóvel, a imagem do painel de um carro, a imagem do motor de um avião, a imagem do tabuleiro do jogo de xadrez, etc. Até um rótulo de garrafa de vinho pode ser um marcador e isso abre a possibilidade, por exemplo, de mostrar no *smartphone* informações sobre o vinho, que não constam no rótulo talvez por falta de espaço nele mesmo. Usando a imaginação de agora em diante, já fica possível pensar em várias aplicações para

a realidade aumentada.

Sem uso dos marcadores ainda é possível usar a realidade aumentada, por exemplo com coordenadas de GPS. Mas, o uso de marcadores é o que transmite a melhor sensação de conexão entre a coisa virtual e a coisa real. Porque a virtual parece estar conectada fisicamente à real. Qual é o caso do jogo Pokemon Go ?

Por que esse assunto faz parte dessa apostila?

A tecnologia da realidade aumentada surgiu há algumas décadas, mas ficou popular com o uso de *smartphones* mais modernos que são capazes de processar as imagens, acessar dados remotos, capturar imagens em alta definição, trabalhar com coordenadas geográficas e tudo mais que a realidade aumentada precisa para ser utilizada.

Essa tecnologia despontou com esses dispositivos e produziu um momento quando ela parecia ser algo que iria deslanchar muito e abrir muitas oportunidades no mundo de Computação. Entretanto, muito do seu uso inicial foi aplicado em jogos, e como chamariz em algumas campanhas de marketing. Mas foi só isso? O que se discute hoje é que a realidade aumentada tem sim grande potencial para criação de aplicações de grande utilidade às pessoas. O que faltou até agora foi a criação de aplicações para suprir grandes demandas reprimidas. Mas, quais são essas demandas? Só será possível descobrir quais são elas através da inserção de aplicações no mercado, com a realidade aumentada (RA). Então, cabe aos profissionais de Computação criarem aplicações de grande utilidade, talvez em parceria com profissionais de outras áreas, como Medicina, Turismo, Arquitetura, Mecânica, etc. Um aluno dessa disciplina, sendo profissional da Computação, pode bolar um projeto que use realidade aumentada e apresentá-lo oralmente à classe, por exemplo. Ou seja, apresentar apenas as ideias, não necessariamente código. Boas ideias nessa área de RA terão grande valor.

Em alguns casos, os *smart glasses* já estão iniciando uso da realidade aumentada e há quem diz que a RA poderá deslanchar com eles. A Boing, por exemplo, já faz esse tipo de uso em sua fábrica, para a montagem do cabeamento dentro da fuselagem das aeronaves. Se a RA virá com força total junto com os *smart glasses*, então vale a pena já ter um primeiro contato agora com essa tecnologia. Um aluno dessa disciplina poderia fazer uma apresentação oral sobre o Google Glass (O que é? Para que serve? Que tecnologias usa? Como programar para ele? Etc), mostrando se o mesmo já tem caso de uso com RA.

Como a RA tem grande potencial para se tornar tecnologia de muitas oportunidades na Computação, é válido deixá-la constar nesse material. Mas, é claro que um pouco disso é aposta também. Ou seja, dos assuntos presentes nesse material, pelo menos um deles tem um pouco de aposta no futuro, acrescentando uma pitada de risco em gastar tempo para entendê-la. Mas, sem riscos, não há grandes ganhos.

Em 2014, pesquisadores de Engenharia Civil e Arquitetura da Unicamp já apostavam em RA, segundo o vídeo [106].

Exemplos iniciais

Para iniciar o entendimento em RA, veja um exemplo de 2009 no vídeo da empresa Tok&Stok em [107]. Nesse vídeo o que ocorre é que vários marcadores são usados diante de um monitor que os filma. No monitor, são produzidas imagens de móveis da Tok&Stok sobre esses marcadores. Então, ao movimentar os marcadores, trocando-os de posição, fica possível ver a variação da disposição espacial dos móveis, o que ajuda muito perceber como eles podem ficar melhor dispostos dentro de uma residência, sem ter que colocá-los dentro da mesma, realmente. Isso é um 'atalho' para concluir quais móveis combinar entre si, antes de comprá-los. A realidade é formada apenas por cartões de papel e a informação virtual adicionada são as imagens dos móveis, através do monitor que os demonstra. Esse vídeo dá uma noção muito boa do quanto útil pode ser a RA. Sempre que ela aumentar a informação disponível, ela estará sendo útil!

Há também outro vídeo, como bom exemplo da realidade aumentada, em [108], de 2016. Esse caso foi premiado na AWE. A AWE (*Augmented World Expo*) é a maior exposição mundial de RA. Esse vídeo mostra que uma aplicação pode definir o marcador e depois disso colocar um objeto virtual sobre o mesmo marcador. O legal aqui é que o marcador é definido pelo usuário final. Geralmente as aplicações de RA já vêm com seus marcadores definidos em seu banco de dados de imagens de marcadores. Mas, permitir o usuário criar seus próprios marcadores dá muito mais flexibilidade à aplicação. Imagine o mundo de possibilidades existentes quando é possível deixar os usuários definirem seus próprios marcadores e depois compartilhar os mesmos pela nuvem ! A AWE ocorre nos EUA, na Europa e na Ásia. A mais recente ocorreu em 2020, nos dias 26 a 29 de maio, de forma online devido a pandemia. Mais informação aqui: <https://augmentedworldexpo.com/>.

Construção de aplicativo com RA e marcadores físicos, em linhas gerais

Para se construir um aplicativo de RA, com marcadores físicos, a primeira coisa a se fazer é obter as imagens desses marcadores. Por exemplo, se o marcador é uma dada rua, vista de um dado ponto e direção, então uma imagem (Ex: fotografia) dessa visão deve estar disponível, impressa por exemplo. Quase todas as imagens servem como marcador. Nem todas, porque certas características precisam estar presentes nas imagens, o que será visto mais adiante.

Inicialmente, a tecnologia de RA usava marcadores com estilo igual aos mostrados a seguir:

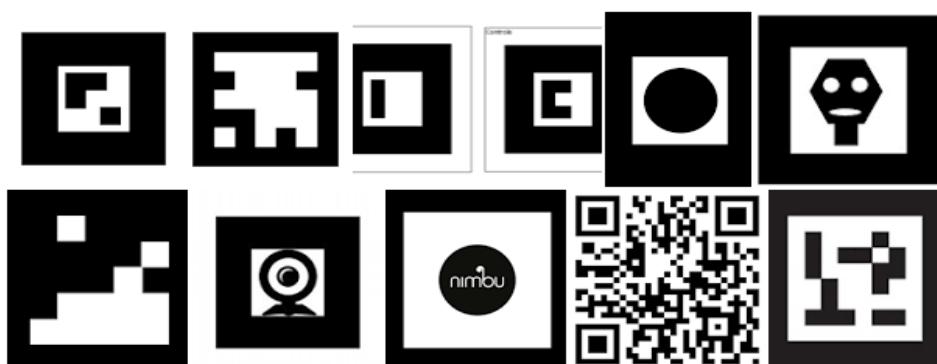


Figura 57

Atualmente os marcadores podem ter características naturais.

Em segundo lugar é necessário desenhar um objeto 3D virtual, por exemplo. Isso pode ser feito na ferramenta Unity 3D. O objeto pode ser qualquer coisa. Por exemplo, o desenho de uma placa de trânsito indicando a velocidade pela qual pode-se dirigir um veículo na rua alvo (marcador). A imagem do desenho em 3D deve ser salva.

Em seguida a imagem do marcador deve ser colocada no banco de dados de marcadores a ser utilizado pela aplicação. Ou seja, a imagem é jogada dentro de um arquivo que é o banco de dados. Para isso, existe ferramenta disponível, em web site, como explicado mais adiante.

Outro passo a ser feito é associar a imagem desenhada 3D ao marcador. Essa associação pode ser feita no código do aplicativo de RA. Tal código pode ser programado com o auxílio de bibliotecas ou SDKs disponíveis atualmente. Exemplos de bibliotecas são:

- ArUco: essa é de código aberto. Usa OpenCV, biblioteca mais usada mundialmente para visão computacional. A documentação é boa e essa biblioteca é boa para fins de pesquisas e didática. Uma apresentação oral sobre essa biblioteca Aruco, relacionando-a à RA e explicando um pouco sobre a OpenCV, seria apropriada a essa disciplina. O web site [109] pode ser um bom ponto de partida sobre o assunto.

- Vuforia: essa é uma biblioteca de uso gratuito, mas o código não é aberto. É uma ótima biblioteca para reconhecimento de marcadores. Essa biblioteca tem a capacidade de descobrir qual é a pose do marcador, para então corrigir a pose do objeto virtual a ser mostrado. Conforme dito no site [110] (2016) “*Vuforia is the most widely used augmented reality software platform in the world, with a global network of more than 230,000 registered developers. More than 25,000 Vuforia-powered applications have driven over 260 million app installs.*”

A imagem abaixo mostra o crescimento do número de desenvolvedores com Vuforia. É interessante notar que houve um crescimento significativo de 2014 a 2015. 38% de crescimento. Então essa tecnologia está sendo cada vez mais considerada por desenvolvedores. O crescimento chinês é o mais destacado aqui.



Figura 58. Fonte [111]

Já o crescimento de aplicativos usando RA, disponíveis em *app stores* cresceu apenas 7% no mesmo período, como visto na imagem seguinte:

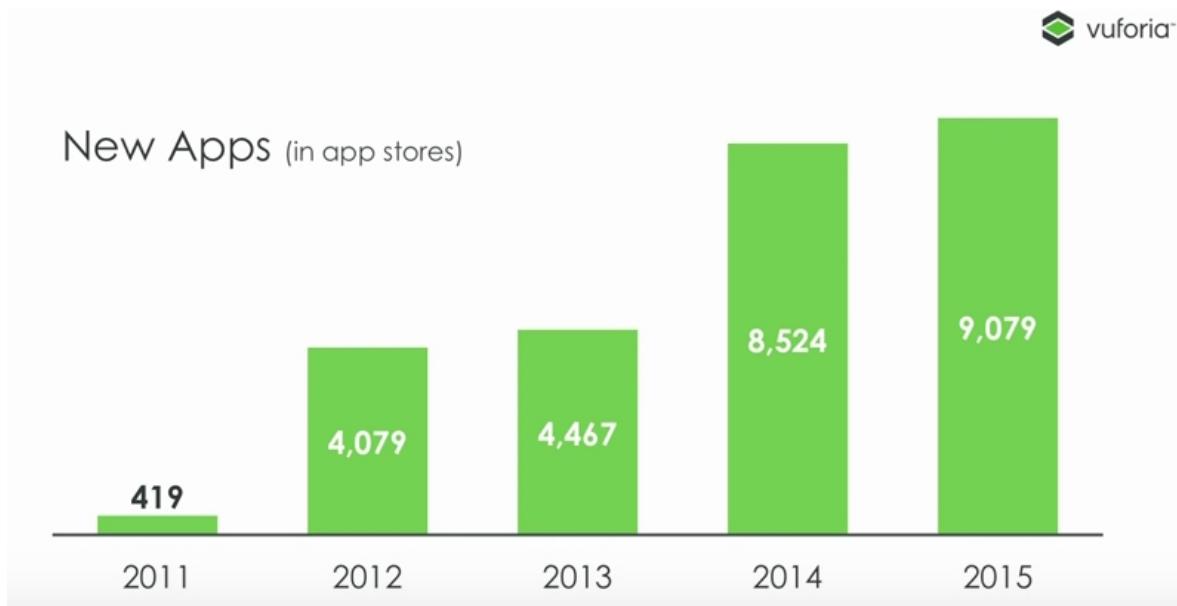


Figura 59. Fonte [111]

O que ocorre é que muitos dos novos projetos não foram disponibilizados em *app stores*, porque ficaram confinados nos domínios de empresas. Ou seja, existe uma crescente demanda por RA, mas com o propósito de projetos de grandes empresas. Portanto, a euforia inicial com Vuforia, entre as pessoas, pode ter acabado. Mas, como dito antes, deve haver muita demanda reprimida para novas aplicações com RA. A própria PTC, que atualmente mantém a tecnologia Vuforia, fez o levantamento dos dados na imagem seguinte. Reparar que aqui o crescimento de projetos no mesmo período foi de 86%! Muitos desses projetos já podem estar usando *smart glasses*, mas isso é apenas uma suposição. Vale a pena uma pesquisa sobre isso.



Figura 60. Fonte [111].

Os gráficos acima foram mostrados por Jay Wright (SVP, PTC Vuforia), no AWE de 2016 que ocorreu na China.

Vuforia

A tecnologia Vuforia foi criada pela Qualcomm. Posteriormente, em 12/10/2015, foi vendida à PTC por 65.000.000 de dólares, cuja aquisição foi completada em 3 de novembro daquele ano. A PTC planeja incluir a RA com Vuforia na IoT. Pelo menos a PTC prevê uma grande variedade de *use cases* com RA. Para as pessoas comuns, a PTC sugere um exemplo do emprego de RA: sensores de um carro podem enviar informações à nuvem e posteriormente essas mesmas informações poderão ser recuperadas por *smartphones*, quando os mesmos são apontados, por exemplo, para o motor do carro. Aí entra a RA mostrando imagens sobre , por exemplo, como verificar o nível do óleo e como trocá-lo, se o nível estiver abaixo do recomendado e se isso foi relatado previamente pelos sensores. Ótimo para quem não entende de troca de óleo de carro, por exemplo.

No mundo corporativo, de comércio de produtos ou serviços, a PTC entende que a RA poderá substituir manuais em PDF, por imagens que vão direto ao 'foco' do que deve ser visto e aprendido como usar, dentre exemplos dessa empresa.

A Qualcomm criou a Vuforia para de certa forma destacar seus chips. Mas, ela não conseguiu monetizar o uso da Vuforia. Já a PTC pretende usar a tecnologia de forma mais relacionada com *business core* de outras empresas. No vídeo [111] o CEO da PTC traça os planejamentos dessa empresa em relação à Vuforia. Esse vídeo pode ser visto por um aluno dessa disciplina, para adicionar mais informações à apresentação oral que será feita sobre Vuforia.

Como trabalhar com Vuforia?

- Entenda esta tecnologia. (Documentação Rica)
- Use as ferramentas indicadas pela PTC.
- Crie o seu software usando o SDK. Ou use a ferramenta Unity 3D.
- Determine as imagens alvo. Inclua estas imagens no aplicativo.
- Projete as imagens que aumentarão a realidade. Crie estas imagens em ferramentas para gráficos vetoriais.
- Identifique demandas por realidade aumentada.
- Ponto de partida = <https://developer.vuforia.com/>

Componentes da tecnologia:

Ao trabalhar com Vuforia o desenvolvedor deve ter uma visão geral de quais são os componentes dessa tecnologia, que ele pode usar e que lhe trará utilidades facilitadoras ao desenvolvimento de projetos de RA. A imagem seguinte mostra uma visão geral desses componentes e a interação entre eles. Felizmente Vuforia foi criada de forma facilmente entendida e atualmente o web site [112]

dessa tecnologia está muito bem organizado, um verdadeiro apoio a quem irá iniciar trabalhos nessa área. A empresa PTC está fazendo um bom trabalho com a Vuforia.

Como visto na figura seguinte, existe uma parte da tecnologia cujas utilidades são direcionadas às ações dos desenvolvedores. O resto é responsabilidade da própria Vuforia em desenvolver ações computacionais de apoio aos projetos em construção. Por exemplo, em uma aplicação para *smartphone*, usando Vuforia, o desenvolvedor fará uso do SDK adequado e programará a lógica do software, como a renderização da imagem ou demonstração da informação virtual sobre o marcador. Com essa programação o sistema também permitirá que o usuário faça ações no aplicativo, no momento de usá-lo. Por outro lado, a tecnologia Vuforia, ainda na mesma aplicação, se encarregará de buscar, de um banco de dados, os atributos das imagens marcadoras até que seja reconhecida a imagem sendo capturada correntemente pela câmera do *device*. Esses bancos de dados podem estar localizados no mesmo dispositivo ou na nuvem. O reconhecimento de um marcador fica por conta da Vuforia, que deve buscar no banco de dados informações suficientes para tal.

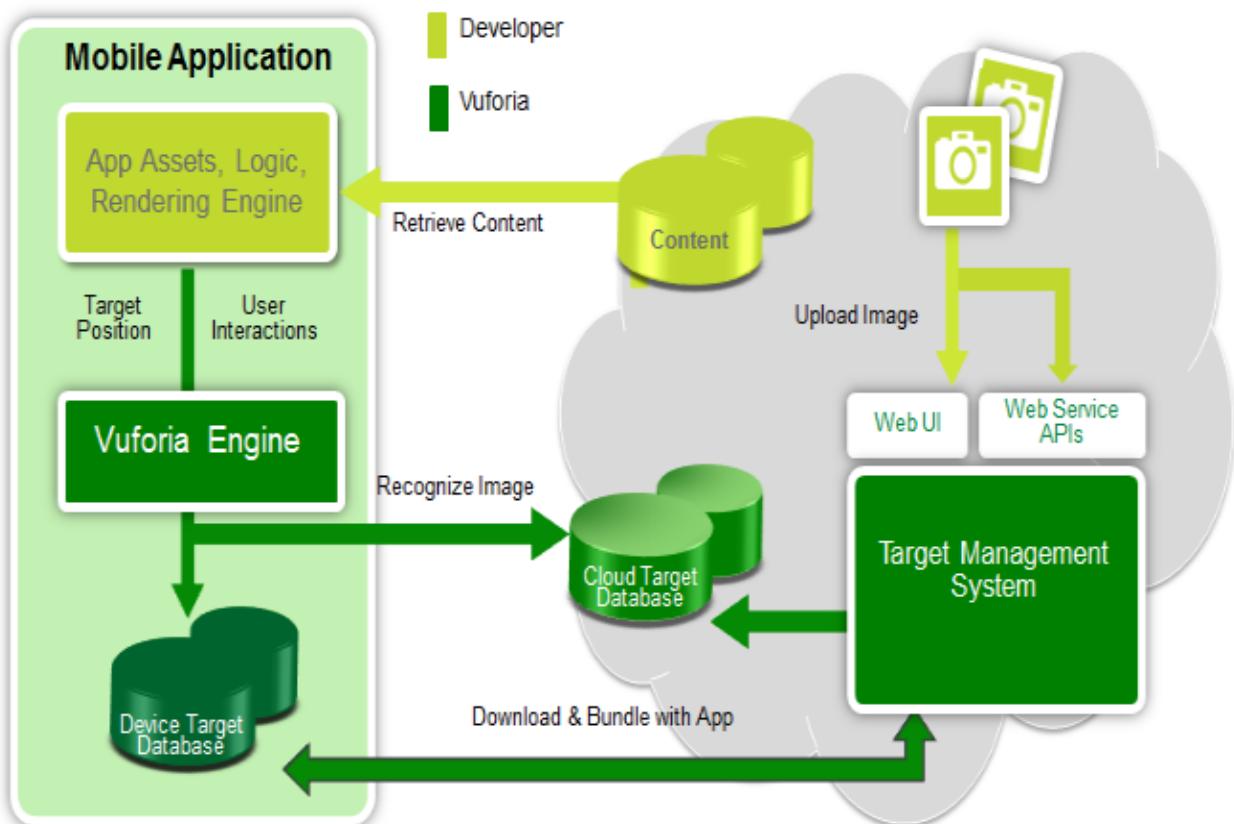


Figura 61

Para colocar informações de marcadores dentro dos bancos de dados, tem-se que usar o sistema de gerenciamento de *Targets*. Esse é um sistema que fica na *web*, para o qual pode-se fazer *uploads* de imagens, por exemplo. Dessa forma, o sistema de gerenciamento de *Targets* retira as informações que serão usadas no futuro para o reconhecimento de marcadores. Esse sistema gera então um conteúdo que pode ser transferido para um banco de dados do *device* ou para um banco de

dados da nuvem. Para usar o sistema de gerenciamento de *Targs*, pode-se acessá-lo via API de *web services* (RESTful), ou via páginas *web*. Existem formas para a aplicação interagir com o banco de dados na nuvem e com o banco de dados local no *device*. Além do conteúdo prévio em banco de dados, uma aplicação com Vuforia também pode obter imagens, como marcadores, em tempo de execução, se desejado pelo usuário, e depois utilizar esses marcadores mantendo-os em bancos de dados, como comentado com mais detalhes adiante nesse capítulo.

De curiosidade, algumas empresas famosas já usaram Vuforia em aplicações para celulares, para criar um melhor destaque de suas marcas.



Figura 62

Veja o que pode ser reconhecido com Vuforia :

Objetos do mundo real

A tecnologia Vuforia permite que objetos do mundo real (objetos físicos, palpáveis, que ocupam o espaço 3D) sejam reconhecidos por um aplicativo, para que informações virtuais sobre os mesmos sejam demonstradas. Existem várias ferramentas da Vuforia que podem ser usadas para construir o reconhecimento desses objetos. Esses objetos podem ser brinquedos, outros objetos dos quais seja necessário, por exemplo, mostrar um manual de utilização, produtos, objetos com geometria complexa, etc. Nem todos os objetos podem ser reconhecíveis.

Objetos ideais do mundo real não podem ser brilhantes, devem apresentar um bom contraste das partes de sua superfície, não podem ter partes articuladas (estas podem se mover e mudar a aparência reconhecível como cadastrada nas informações do banco de dados), podem ser coloridos ou em escalas de cinza. Segue abaixo 2 exemplos de objetos reconhecíveis pela Vuforia e que podem então serem usados como marcadores.



Figura 63. Fonte [112]



Figura 64. Fonte [112]

Esses objetos são definidos em tempo de desenvolvimento da aplicação. E o desenvolvedor pode usar o **Vuforia Object Scanner** para capturar as imagens do mundo real, mas de tal forma que elas se tornem verdadeiros marcadores eficientes. O desenvolvedor deve obter as imagens dos objetos em certas condições adequadas. Por exemplo, é melhor usar luz difusa sobre os objetos e ter um fundo cinza por trás deles. O ambiente dentro de casa é apropriado então. Também é bom evitar sombras próximas de outros objetos ou de pessoas.

Os três passos a seguir mostram o que deve ser feito até que se tenha obtido imagens de objetos do mundo real, de tal forma que elas se transformarão nos marcadores para a aplicação de RA em desenvolvimento:

- a) Escanear um objeto físico com o **Vuforia Object Scanner** para criar um arquivo de Dados de Objeto.
- b) O arquivo de Dados de Objeto é então enviado para o **Vuforia Target Manager** (sistema gerenciador Vuforia de alvos/marcadores). No gerenciador Vuforia de marcadores um *Object Target* (marcador/alvo) pode ser gerado e depois colocado dentro de *Device Database*. Segundo o web site da Vuforia, até 20 imagens alvo (marcadores) podem ser colocados dentro do *Device Database*. Contudo, essa parece ser uma pequena quantidade disponível e quem irá pesquisar mais sobre Vuforia deveria investigar mais a fundo essa informação para ver se existe mesmo tal limite.

c) O banco de dados do gerenciador Vuforia de alvos/marcadores pode ser então baixado e adicionado no **Vuforia Object Recognition project** desenvolvido no Eclipse, ou no Xcode ou Unity, por exemplo.

O **Vuforia Object Scanner**

O **Vuforia Object Scanner** é um aplicativo apenas para Android e tem a capacidade de escanear um objeto 3D. O resultado do *scan* é um arquivo .OD. Esse arquivo contém informação sobre o objeto (sobre a imagem dele) que pode ser enviada ao Vuforia Target Manager. Essas informações são suficientes para o *Vuforia Target Manager* criar então o marcador reconhecível. O resultado disso é que o marcador reconhecível, depois de incluído no *Device Database*, permite o *device* reconhecer o mesmo objeto que fora escaneado outrora, através de visualização do mesmo pela câmera.

As duas imagens seguintes mostram exemplos diferentes de ambientes apropriados para escanear o objeto:



Figura 65. Fonte [112]



Figura 66. Fonte [112]

O Vuforia Object Scanner pode ser encontrado nesse link [114].

O objeto a ser escaneado deve ser colocado sobre uma superfície com uma imagem específica estampada nela mesma. A imagem é formada por figuras triangulares, em cor cinza. Essa imagem é imprescindível, porque ela é o referencial de posicionamento do objeto a ser escaneado. Ou seja, em tal imagem existe as coordenadas espaciais (0,0,0) que coincidem com o centro dela. Dessa origem (0,0,0) partem os eixos X,Y e Z. Então, a posição do objeto colocado sobre tal superfície, em relação aos eixos, determinará que partes da imagem devem ser escaneadas. Isso possibilita escanear toda a imagem ou apenas uma fração dela, caso seja desejado. Enquanto o objeto é escaneado, a superfície referencial não pode ser movimentada e nem o objeto. Caso contrário o resultado é corrompido.

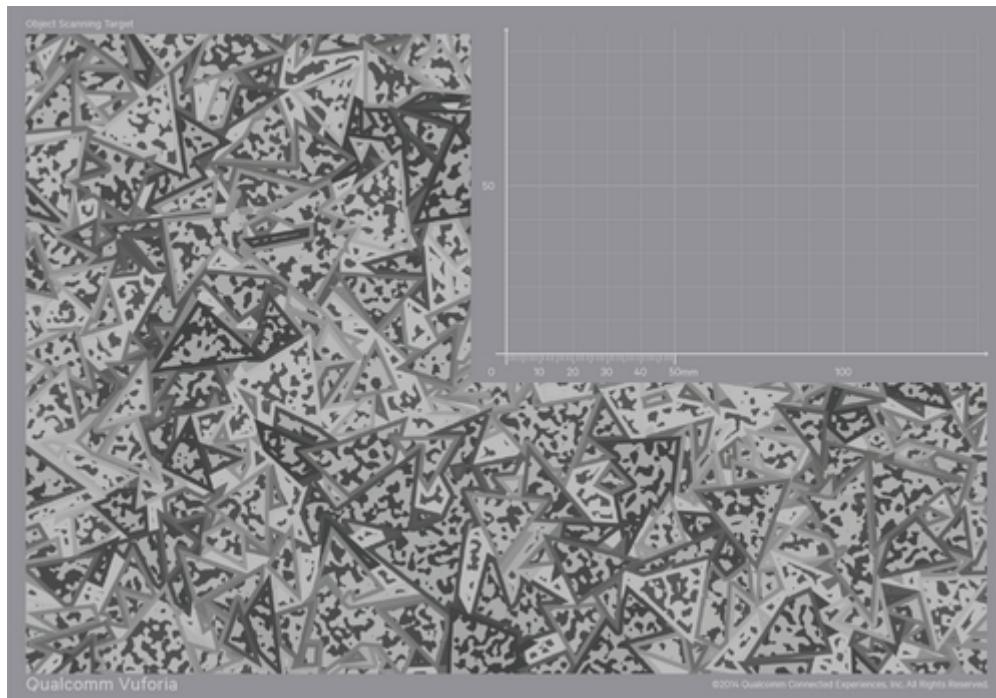


Figura 67. Fonte [112]

Essa imagem referencial deve ser impressa em 100% de seu tamanho original. O Vuforia Object Scanner está preparado para compreender o posicionamento do objeto em relação a essa superfície referencial.



Figura 68. Fonte [112]

Então o objeto pode ser posicionado, como mostrado na figura acima, a fim de ser 100% escaneado. O próprio *Vuforia Object Scanner* usa RA e mostra os eixos X,Y e Z, com projeções sobre X e Y, de tal forma que fica fácil ver na tela do *smartphone* se o objeto está bem posicionado.

Alguns brinquedos, por exemplo, contêm partes semelhantes e o reconhecimento visual dos mesmos dar-se-á apenas pelas partes incomuns. Esse é um caso quando apenas uma fração do objeto precisa ser escaneada. Como visto na figura seguinte.

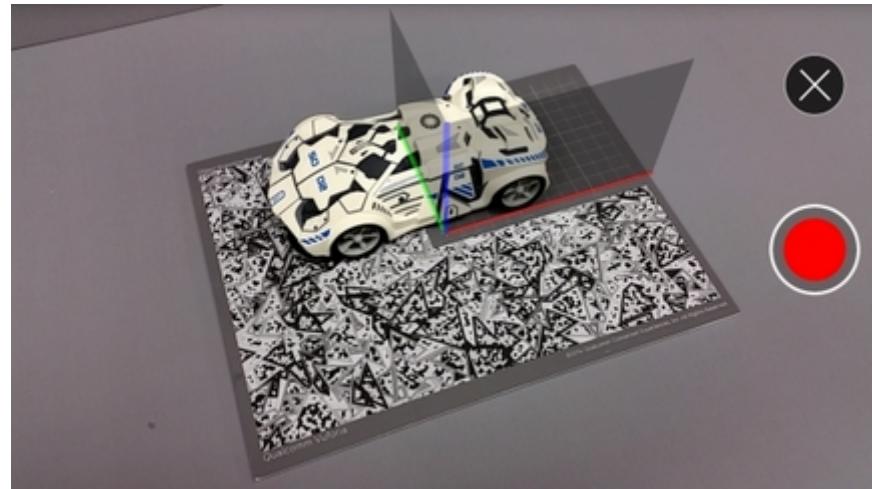


Figura69. Fonte [112]

Se o objeto a ser escaneado é bem maior que a superfície referencial isso pode causar algum problema ? A pergunta poderia ser respondida por quem irá apresentar Vuforial oralmente.

Assim que o botão de gravar (executar o *scan*) é acionado o *Vuforia Object Scanner* começa a mostrar partes da superfície do objeto que já estão escaneadas.



Figura 70. Fonte [112]

Nesse momento o *smartphone* executando *Vuforia Object Scanner* deve ser movido em volta do objeto escaneado, para toda a superfície do objeto ser vista.

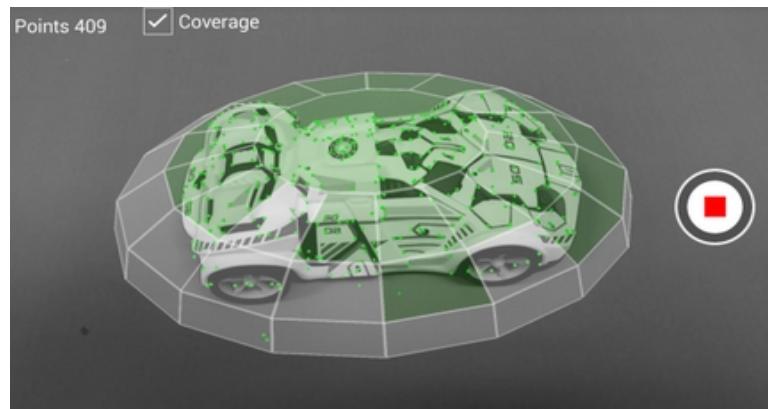


Figura 71. Fonte [112]

O resultado obtido pode ser analisado e testado no próprio *Vuforia Object Scanner*.

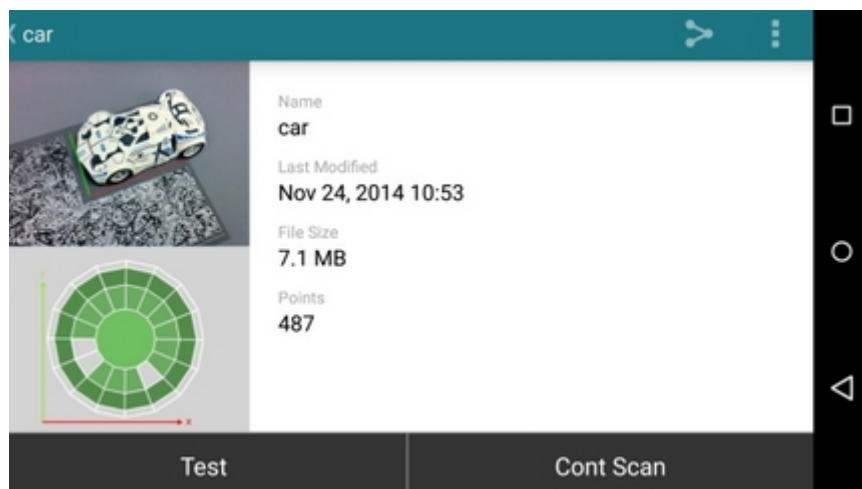


Figura 72. Fonte [112]

Pode-se ver quanto da superfície do objeto 3D foi escaneada com sucesso. E ao pressionar o botão

Teste, o *Vuforia Object Scanner* irá testar a reconhecibilidade do objeto real. Durante o teste, uma imagem aumentada aparecerá na origem do objeto real. Isso mostrará que o objeto está sendo reconhecido com sucesso. É bom testar variando o *background*, para ver se o objeto real é reconhecido em lugares diferentes.



Figura 73. Fonte [112]

Como essa apostila só pode mostrar a teoria, o aluno que fará uma apresentação oral sobre Vuforia poderia mostrar o *Vuforia Object Scanner* funcionando na real.

O *Vuforia Object Scanner* permite enviar o resultado do *scanner* para o *Target Manager*. Mas antes é preciso passar para o PC. A imagem seguinte mostra formas de transferir o *Object Data file* para o PC.

Um arquivo .zip é enviado. Dentro dele está o .OD. Esse .OD pode ser enviado para o *Device Database* através do *Vuforia 5 Target Manager*.

Resumo:

Escanear -> enviar arquivo .zip para o PC -> extrair .OD -> usar o *Target Manager* passando o .OD para ele -> fazer o *download* do marcador reconhecível para o *Device Database*. É o *Target Manager* que cria o *Device Database*.

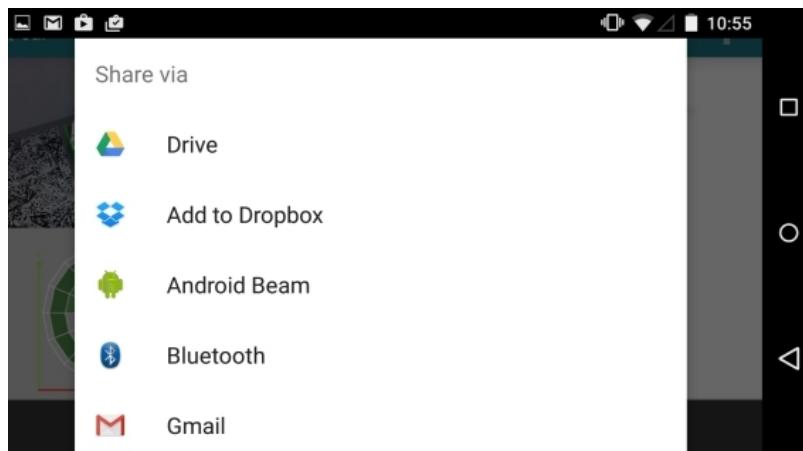


Figura 74. Fonte [112]

Outro tópico relacionado com Vuforia, que pode ser explorado e apresentado oralmente pelos alunos dessa disciplina é : Como adicionar um *Device Database* a um projeto Vuforia. Além disso, outro assunto interessante a apresentar pode ser : Como usar o reconhecimento de objetos em uma aplicação Android.

Qualquer protótipo de aplicação RA usando Vuforia poderia fazer parte de uma apresentação sobre Vuforia, feita por alunos dessa disciplina. Ou alguma aplicação simples desenvolvida e funcionando.

Imagens obtidas pelos usuários

São imagens escolhidas pelo próprio usuário, como fotografias capturadas de coisas do mundo real, que podem então ser transferidas para a aplicação de RA e a partir daí tais imagens passam a ser alvos reconhecíveis. Em outras palavras, o usuário da aplicação de RA cria seus próprios marcadores. Isso dá grande poder à tecnologia de RA. Imagine o poder da RA se tais marcadores puderem ser compartilhados pela nuvem. Mas, as imagens resultantes como marcadores precisam conter certas características. Para tanto, a aplicação de RA usa a Vuforia API. E é necessário passar ao usuário as regras de como capturar as imagens e que imagens podem ser escolhidas. Na própria aplicação o desenvolvedor deve comunicar ao usuário sobre as qualidades de uma boa imagem alvo, de tal forma que o usuário consiga selecionar imagens que suportarão o reconhecimento robusto delas mesmas pela aplicação futuramente.

Atributos de imagens ideais são:

- Ricas em detalhes, como ruas, grupo de pessoas, mistura de itens, etc.
- Bom contraste, ou seja, partes claras e escuras bem definidas.
- Sem padrões repetitivos. Um chão gramado daria uma imagem ruim, por exemplo.

Segundo o web site oficial da Vuforia há uma ferramenta no SDK que pode ser usada nas aplicações e que indica ao usuário que ele está escolhendo uma boa imagem para ser um marcador, quando é o caso.

O aluno que irá apresentar sua pesquisa sobre Vuforia deve preferencialmente estudar o conteúdo do web site oficial, visto em [112], entender o básico da codificação com o SDK Vuforia e então mostrar e explicar, por exemplo, como fazer uma aplicação ter a capacidade de capturar imagens para servirem de marcadores, em termos de código fonte. Isso pode ser visto na web page [113].

Felizmente, existe exemplo de projeto para *download* no site da Vuforia, que mostra então como criar uma aplicação que permitirá o próprio usuário definir suas imagens alvo (marcadores). E esse exemplo mostra como desenhar uma interface e elementos de experiência de usuário de tal forma a capacitar uma aplicação de RA que permite o usuário obter seus marcadores efetivamente. Tal download pode ser executado a partir do site [115].

Imagens planas obtidas em tempo de desenvolvimento



Figura 75. Fonte [112]

A imagem acima mostra um alvo (marcador) plano e uma imagem 3D de um bule virtual mostrado sobre ela. Esse tipo de aplicação é adequada, por exemplo, para mostrar informação adicional sobre caixas de produtos, se as caixas não forem cilíndricas.

Em geral, as imagens alvo devem estar em superfícies planas e rígidas. Imagens que não são rígidas, como em jornais, devem ser testadas cuidadosamente. A figura seguinte mostra alguns bons exemplos.



Figura 76. Fonte [112]

Características das imagens alvo (marcadores)

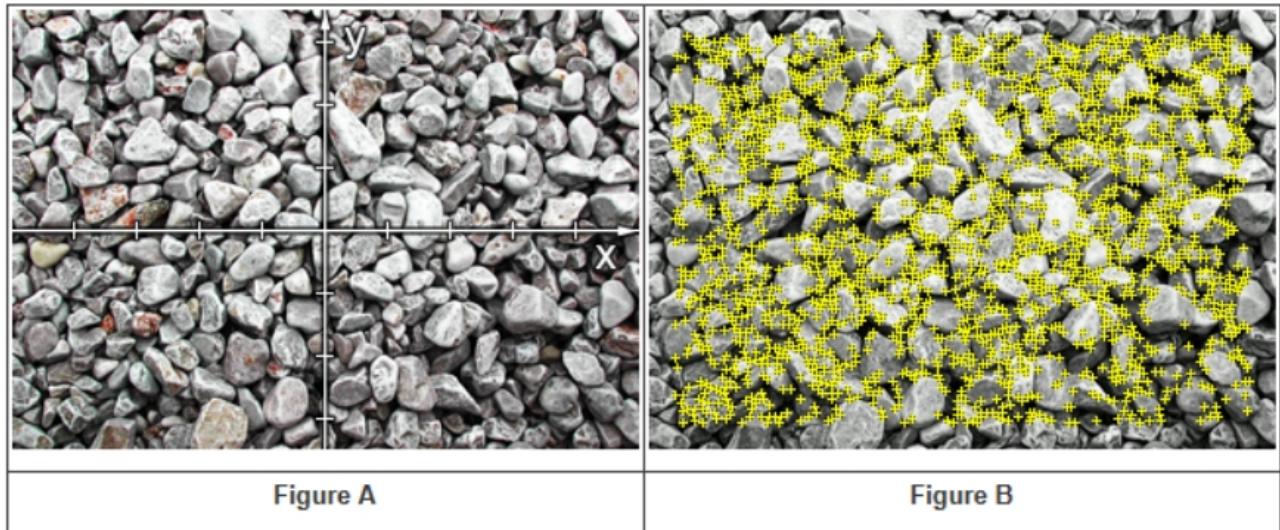


Figura 77.

A figura A representa um marcador com características naturais. A imagem B mostra onde estão estas características. Cada cruz amarela é uma característica. A posição dessas cruzes seria diferente se a figura A fosse outra imagem. Então, aquele posicionamento específico das características representa um mapeamento particular da figura A. Isso implica que, sempre que o mesmo mapeamento for visualizado no marcador, pela Vuforia, a imagem respectiva estará reconhecida. É justamente através das características particulares que uma imagem alvo pode ser reconhecida na Vuforia.

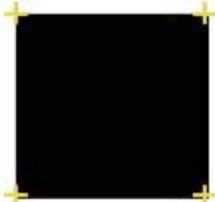
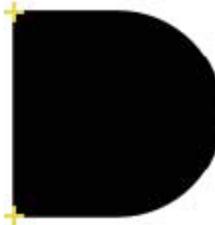
	Um quadrado contem 4 características, uma em cada canto.
	Um disco tem zero características
	O objeto ao lado tem duas características. Conclusão: formas orgânicas não dão bons marcadores.

Figura 78. Fonte[112]

O *Target Manager* pode analisar uma imagem e dar uma nota a ela de 0 a 5. Uma imagem com nota

5 dará um ótimo marcador e será facilmente reconhecido pelo SDK da Vuforia. Uma característica é uma ponta, ou algo que se aproxima disso, em contraste com a área ao redor. As figuras seguintes mostram a pontuação de imagens, dada pela Target Manager.

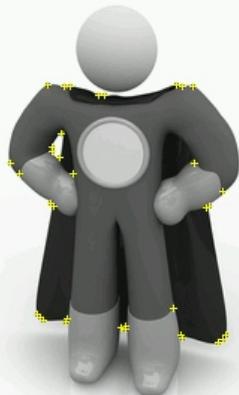
Uploaded Image	Analyzed Image	Star Rating
		

Figura 79. Fonte [112]

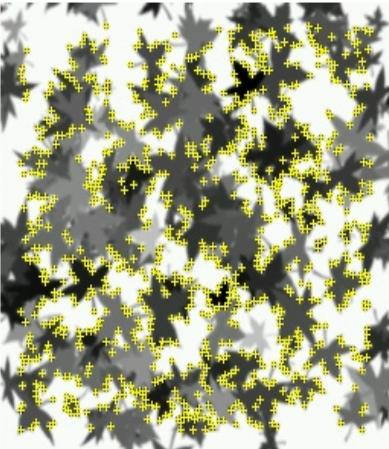
		
--	---	---

Figura 80. Figura [112]

Boas imagens para servirem de alvo devem ter as características uniformemente distribuídas. O que não é o caso daquele boneco de capa vermelha. A imagem seguinte é um bom exemplo de marcador inadequado:



Figura 81. Fonte [112]

Essa imagem terá características mal distribuídas, portanto terá uma pontuação baixa no *Target Manager*.

A imagem seguinte é um exemplo muito ruim para servir de marcador:

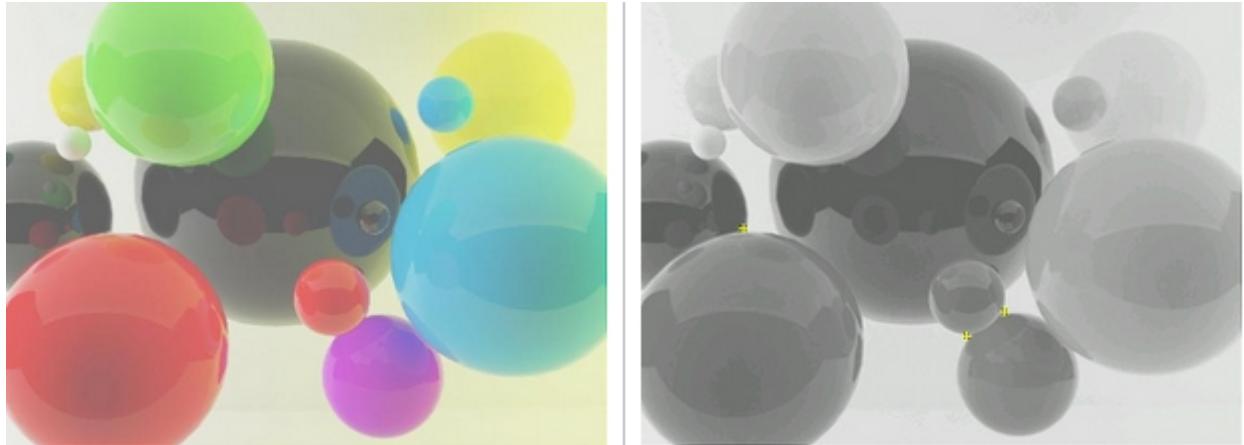


Figura 82. Fonte [112]

Padrões repetitivos devem ser evitados. Ou melhor, evitar imagens com essas características.

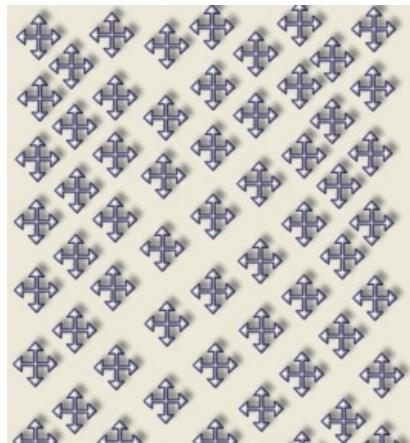


Figura 83. Fonte [112]

Nesse tipo de imagem o SDK tem dificuldade em reconhecer qual parte da imagem está sendo visualizada pela câmera do dispositivo. Por exemplo, se a câmera estiver vendo a parte sul da imagem, como o SDK poderá saber que não é a parte norte?

As imagens devem ser colocadas no *Targe Manager* no formato JPG ou PNG.

Outra vantagem interessante da Vuforia é a possibilidade de inserir botões virtuais sobre as imagens alvo. E o SDK pode detectar quando o usuário colocar o dedo sobre um desses botões. Ou seja, há uma forma de criar uma interação entre usuário final e marcador.

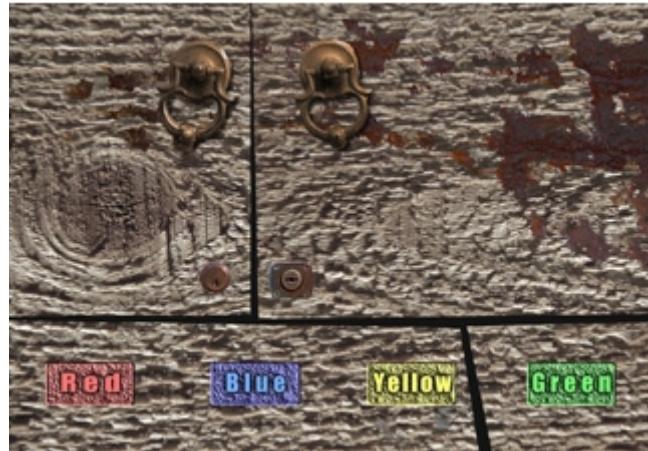


Figura 84. Fonte [112]

Imagine um botão virtual [Preço] sobre a caixa de um produto. O usuário poderia sempre ver o preço atualizado, por exemplo dentro de um supermercado, sem que funcionários do estabelecimento tivessem que atualizar etiquetas. E o fabricante não precisa mudar a arte gráfica da sua embalagem para adicionar um desenho de botão reconhecível. Prático! Na RA o que tem valor maior é a criatividade de quem lida com ela.

Detalhes de como adicionar um botão virtual a um objeto real poderiam ser estudados e apresentados oralmente pelos alunos que irão estudar e comentar a Vuforia.

Botões também podem ser criados em tempo de execução. Mas que utilidade isso teria?

Cilindros, caixas e textos

Outras coisas do mundo real reconhecíveis pela Vuforia são cilindros, textos e caixas. Essas três categorias de objetos têm suas particularidades ao se trabalhar com Vuforia tomando-as como marcadores. Esses objetos poderiam ser explicados numa apresentação oral sobre Vuforia, indicando os casos de usos mais comuns para esses tipos de objetos e a RA.

VuMark

Esta é uma novidade inserida na tecnologia Vuforia pela PTC. Não existia quando a Qualcomm ainda era dona da Vuforia. VuMarks participam da Vuforia a partir da versão 6, que foi lançada em agosto de 2016.

Os VuMarks são a próxima geração dos *bar codes*.



Figura 85. Fonte [112]

VuMarks são figuras, como marcas de empresas, que aceitam códigos e imagens como logomarcas. Nos VuMarks podem estar presentes códigos identificadores, URLs e imagens que disparam a realidade aumentada. Eles são melhores que os *bar codes* porque aceitam a inserção de logomarcas nos mesmos, o que os deixam mais adequados às características visuais de um produto identificado por eles. Ou seja, adaptam melhor às marcas, causando um melhor *look and feel* às pessoas.

Esse VuMarks podem ser construídos no Adobe Illustrator. Instruções sobre isso podem ser vistas na página [116]. Caso algum aluno tenha conhecimento em Adobe Illustrator e queira participar da apresentação sobre Vuforia, então uma apresentação de como construir VuMarkes nessa ferramenta seria algo muito interessante para essa disciplina.

Para a criação de um VuMark algumas observações devem ser tomadas:

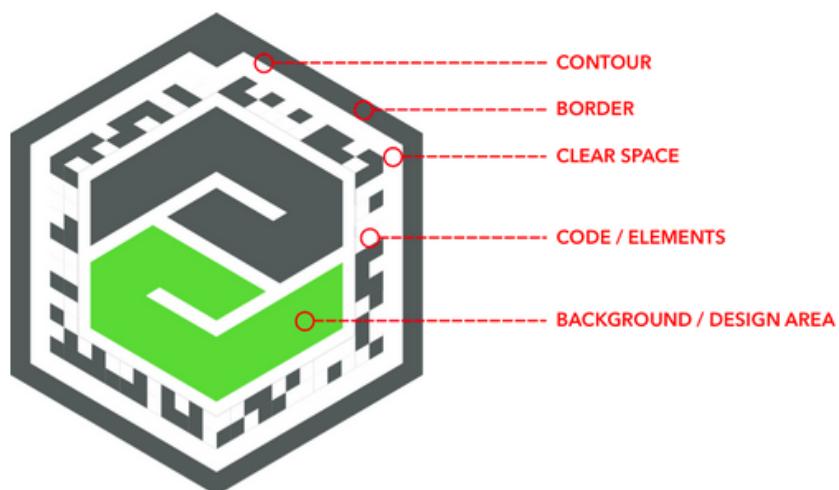


Figura 86. Fonte [112]

Deve existir uma borda formada por linhas retas. Mínimo de 4 linhas. O ângulo entre duas linhas da

borda não pode ser maior que 150° . Todo VuMark tem uma altura e uma largura. A diferença entre essas medidas não podem ser menor que 10%. A borda deve estar adjacente a um contorno. Nesse contorno são inseridos os elementos de informação/código. Essa informação pode ser um número. Quanto maior é o número, maior é a quantidade de elementos de informação presente.

Os VuMarks podem ser usados, por exemplo, em superfícies de produtos ou máquinas nos quais é necessário mostrar mais informação virtual, impossível na real por falta de espaço na superfície, por exemplo.

Cada VuMark feito no Adobe Illustrator pode ser exportado como um *Scalable Vector Graphics (SVG) file* e então enviado para o *Target Manager*. Então o VuMark é adicionado ao banco de VuMarks e em seguida tal banco pode ser baixado para a aplicação em desenvolvimento. Dessa forma os VuMarks podem ser adicionados às aplicações em desenvolvimento no Unity, Android Studio, Xcode ou Visual Studio 2015.

Em relação ao Vuforia, existem *web services* e APIs disponibilizando vários serviços úteis aos desenvolvedores e às aplicações para tempo de execução. Uma visão geral sobre tais APIs e *web services* poderia ser incluída na apresentação oral sobre Vuforia. O ponto inicial de estudo para isso é o *web site* oficial dessa tecnologia. Vale procurar também por *forums* de discussões ou blogs sobre Vuforia.

Existem vários outros assuntos interessantes de Vuforia. Somente essa tecnologia já daria uma disciplina inteira para essa pós-graduação. Outros assuntos que podem ser explorados, estudados e explicados pelos alunos dessa disciplina são:

- Smart Terrain

- Extended Tracking

- Vuforia e *digital eyewear devices*. Ver aqui [117].

- Precificação do uso de Vuforia. Nada sobre preços de licenças foi dado nesse documento. Mas, como esse é um assunto importante e que chama a atenção de pessoas com poder de decisão sobre projetos, vale muito a pena fazer um estudo de tal precificação e apresentar os detalhes sobre isso oralmente em sala de aula.

Studio Enterprise

Para terminar o assunto sobre Vuforia é bom saber que existe o Vuforia Studio Enterprise, criado pela PTC. Esse é mais um assunto que os alunos interessados em Vuforia poderiam estudar e apresentar. Mais detalhes na página [118].

Wikitude

A tecnologia Vuforia é a líder mundial em plataforma de desenvolvimento de aplicações de RA. Mas, não é tudo. A empresa Wikitude também tem seu destaque.

Conforme a empresa Wikitude, web site em [118], a RA já pode ser um mercado de 90 bilhões de dólares a partir de 2020. E essa empresa clama ser a principal fornecedora de SDK para RA, mundialmente. Ela tem soluções para RA baseada em geo localização e sensores, bem como reconhecimento de imagens 2D.

Em Joinville existe uma empresa chamada Informant Serviços em Tecnologia da Informação. Essa empresa é uma parceira *premium* da Wikitude. A Informant criou o aplicativo *Globo Rio 2016*, usando RA, para celulares iOS e Android. Esse aplicativo foi o primeiro grande projeto criado no Brasil usando *Augmented Reality*, conforme [119]. Foi encomendado pela Rede Globo de Televisão. O projeto foi suportado pelo SDK da Wikitude e teve o objetivo de ser um guia para os espectadores dos jogos olímpicos do Rio, em 2016, provendo formas para os usuários encontrarem seus atletas preferidos na Vila Olímpica, além de localizar partidas de jogos na cidade. E conteúdo exclusivo era mostrado em modo de RA. Segundo a empresa Informant, o SDK da Wikitude foi essencial para alcançar os objetivos do desenvolvimento do aplicativo. Ela considera que criar experiências de RA tornou-se algo simples, porque todas as ferramentas necessárias estão no SDK da Wikitude, prontas para serem usadas. Segue abaixo, de curiosidade, algumas interfaces do aplicativo.

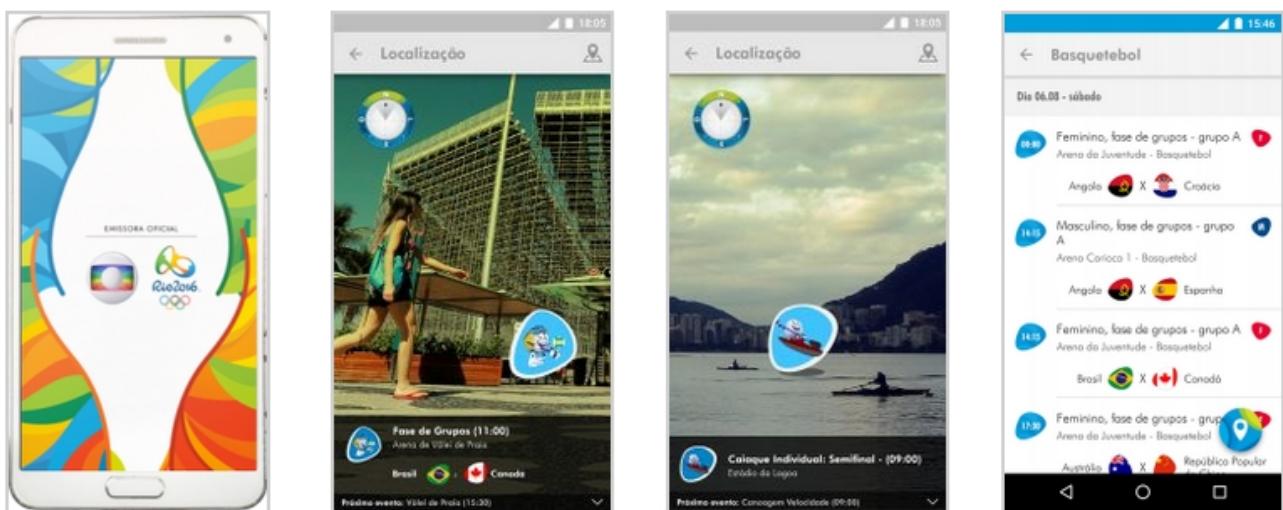


Figura 87. Fonte [118]

Ao apontar o celular para a arena, o usuário via que partida estava acontecendo lá dentro, que equipes se enfrentavam e até o placar do jogo corrente, segundo a notícia [120]. Esse é um exemplo de realidade aumentada que adiciona informação à informação já existente do mundo real. Por exemplo, ao ver uma quadra de tênis, mas apontar o celular para ela, imediatamente podia-se ver informações adicionadas ao mundo real, de forma virtual, para então saber mais sobre o que ocorria em tal lugar. Provavelmente o aplicativo reconhecia as coordenadas do local, para então demonstrar as informações pertinentes. As tecnologias usadas nesse aplicativo foram: Wikitude SDK e Geo-based AR.

A Ford, na Malásia, criou um aplicativo com SDK da Wikitude, para mostrar, através de RA, informações sobre seus veículos, nos pontos de venda dos mesmos. Ou seja, quando um comprador em potencial entra na loja, ele é convidado a usar o *app* dentro do carro, e ver na tela do celular informações interessantes sobre o que o celular está visualizando dentro do carro. Funciona como um guia de usuário sobre o carro. A *app* de RA identifica objetos apontados pelo celular e mostra então as informações respectivas. Esse aplicativo mostra então pontos de destaque, por exemplo nos painéis dos veículos. Ou seja, como uma boa imagem vale mais que 1000 palavras, é até melhor usar esse aplicativo e ver os pontos destacados, que escutar o vendedor contar várias explicações sobre os detalhes visto ali, as vezes rapidamente.



Figura 88. Fonte [118]

As tecnologias aplicadas nesse aplicativo foram: Wikitude SDK, Wikitude Studio e Image Recognition.

Hotels.com criou um aplicativo com RA, suportada por tecnologias da Wikitude, para ajudar o usuário a localizar hotéis perto na região corrente. Basta apontar o celular para uma direção, na rua, e o aplicativo mostrará onde estão os hotéis alí e informações sobre os mesmos. A imagem seguinte explica isso bem melhor.

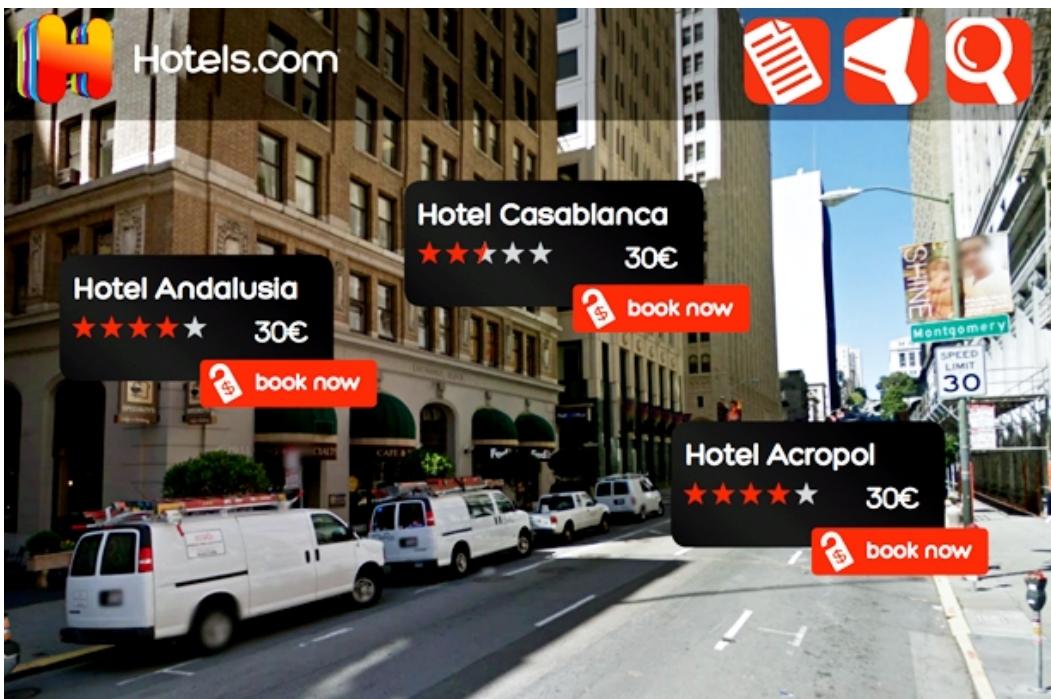


Figura 89. Fonte [118]

Outra empresa, usando tecnologia da Wikitude, construiu um aplicativo para ajudar a encontrar casas a venda.

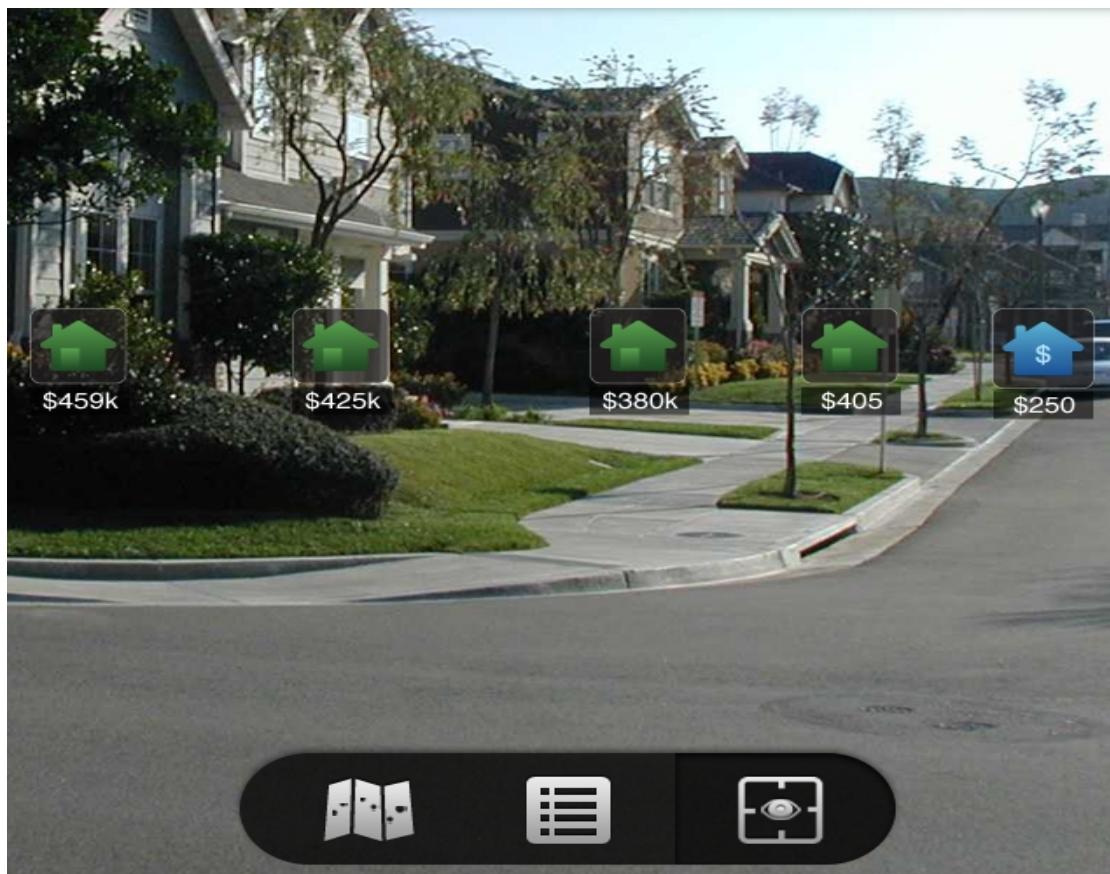


Figura 90. Fonte [118]

Tecnologias da Wikitude:

Wikitude's 3D recognition and tracking = permite uma aplicação de celular com RA reconhecer, por exemplo, uma parte de um motor ou peça em um equipamento. O reconhecimento permite a recuperação de informações, da Internet, sobre a mesma coisa vista pela aplicação.

"Wikitude's SDK works on iOS and Android"

Segundo o pessoal da Wikitude, a RA pode ser considerada como uma ponte visual entre vários objetos do mundo real e a Internet, em termos de acesso desses objetos às informações sobre eles mesmos. Portanto, a RA pode ser considerada como uma forma de criar IoT. Ou seja, mesmo que os objetos não estejam de fato conectados à Internet, ainda assim é possível extrair informações de lá, relacionadas com os mesmos. De certa forma eles estão corretos, porque com a RA pode-se ter coisas associadas a várias informações virtuais contidas na Internet.

O video [121] contem um pouco de *marketing* da Wikitude. Vale a pena ver também.

A Wikitude também tem o Wikitude Studio, que permite a criação de uma aplicação de RA em minutos e sua publicação dentro de uma Wikitude App. Mas, para quem quer criar a própria aplicação original, pode-se usar o Wikitude Studio juntamente com o Wikitude SDK. O Wikitude Studio é usado via web browser. As ferramentas são caras para uma pessoa trabalhar individualmente, mas para projetos de empresas essas ferramentas podem ser de custo apropriado. O Wikitude Studio permite definir o marcador, o aumento da realidade para tal marcador e depois exportar tudo para o Wikitude App. Essa ferramenta pode ser experimentada gratuitamente aqui: [122].

A Wikitude tem o *cloud-based image recognition service*. Com esse serviço uma única aplicação com RA pode reconhecer milhares de imagens. Um aplicativo com tecnologia da Wikitude pode reconhecer até 1000 imagens, mesmo offline. Mas, com o serviço de reconhecimento de imagens baseado em nuvem, esse limite não existe.

Algumas empresas usando a tecnologia da Wikitude:



Figura 91. Fonte [118]

Futuramente, RA e *smart glasses* irão trabalhar juntos para facilitar a vida ainda mais. Então, *smart glasses* poderia ser um dos assuntos dessa apostila, em 2018. Um exemplo sobre isso está no vídeo da Boing aqui [123].

Como visto até aqui, há muito o que estudar em Realidade Aumentada (RA). Qualquer aluno dessa disciplina que tenha interesse, pode desenvolver um projeto de RA e apresentá-lo como TCC. Aqui as oportunidades parecem ser muitas e certamente a demanda reprimida é grande. *Quem pensou que a realidade aumentada estava esquecida, estava desinformado!*

No Brasil também há o *web site* [124], que tem tutoriais sobre construção de software com RA e é uma boa fonte de estudos sobre RA, tudo em Português. E fala da ferramenta ARToolkit.

O Aluno que apresentará a RA oralmente, além de falar sobre Vuforia, poderá falar mais aprofundadamente sobre a Wikitude (já que esse material nem “arranhou” esse assunto), mostrar suas ferramentas, mostrar algum protótipo feito com elas, etc. A ferramenta ARToolkit também aparece nos textos de algumas das bibliografias vistas aqui, mas não foi analisada. Portanto, é outro assunto a ser explorado e possivelmente demonstrado oralmente. No site [125] encontra-se um blog atualizado sobre RA, que vale a pena o aluno investigar e extrair mais novidades dele, para trazer à sala de aula em forma de apresentação oral. Esse blog contém *webinars*. Há um *webinar* que fala sobre os melhores *devices* para aplicar a realidade em cada caso específico, por exemplo.

Finalmente, é notável que o campo de estudo é grande o suficiente para o aluno escolher o que gosta em RA e estudar mais sobre isso. Novidades demonstradas em sala de aula, para os colegas, será grande ganho para todos os participantes dessa disciplina, inclusive para o professor.

Boa sorte e bons estudos!

Anexo I – Script opensips.cfg

O script abaixo é utilizado durante as aulas, quando as funcionalidades do OpenSIPS são demostradas na prática.

```
# OpenSIPS residential configuration script
#      by OpenSIPS Solutions <team@opensips-solutions.com>
#
# This script was generated via "make menuconfig", from
#   the "Residential" scenario.
# You can enable / disable more features / functionalities by
#   re-generating the scenario with different options.#
#
# Please refer to the Core CookBook at:
#      http://www.opensips.org/Resources/DocsCookbooks
# for a explanation of possible statements, functions and parameters.
#

##### Global Parameters #####
log_level=4
log_stderor=no          **** no implica em enviar o log para o /var/log/syslog ****
log_facility=LOG_LOCAL0

children=4

/* uncomment the following lines to enable debugging */
#debug_mode=yes

/* uncomment the next line to enable the auto temporary blacklisting of
   not available destinations (default disabled) */
#disable_dns_blacklist=no

/* uncomment the next line to enable IPv6 lookup after IPv4 dns
   lookup failures (default disabled) */
#dns_try_ipv6=yes

/* comment the next line to enable the auto discovery of local aliases
   based on revers DNS on IPs */
auto_aliases=no

listen=udp:wlan0:5060  # CUSTOMIZE ME
listen=tcp:wlan0:5060  # CUSTOMIZE ME
#listen=tls:wlan0:5061  # CUSTOMIZE ME

##### Modules Section #####
#set module path
mpath="/usr/local//lib/opensips/modules/"  #Se necessário, usar lib64 e não lib.

#=====
#          CARREGAMENTO DE MÓDULOS.
#=====

#### SIGNALING module
loadmodule "signaling.so"

#### StateLess module
loadmodule "sl.so"

#### Transaction Module
loadmodule "tm.so"
modparam("tm", "fr_timeout", 5)
modparam("tm", "fr_inv_timeout", 30)
modparam("tm", "restart_fr_on_each_reply", 0)
modparam("tm", "onreply_avp_mode", 1)

#### Record Route Module
loadmodule "rr.so"
/* do not append from tag to the RR (no need for this script) */
modparam("rr", "append_fromtag", 0)

#### MAX ForWarD module
```

```

loadmodule "maxfwd.so"

##### SIP MSG OPerationS module
loadmodule "sipmsgops.so"

##### FIFO Management Interface
loadmodule "mi_fifo.so"
modparam("mi_fifo", "fifo_name", "/tmp/opensips_fifo")
modparam("mi_fifo", "fifo_mode", 0666)

##### URI module
loadmodule "uri.so"
modparam("uri", "use_uri_table", 0)

##### SQLITE module
loadmodule "db_sqlite.so"

===== Código para executar queries no banco de dados ======START
=====
##### AVPop module
loadmodule "avpops.so" # Esse módulo nos permite executar queries SQL no nosso banco de dados SQLite, para obter
informações de lá.
# default URL
modparam("avpops", "db_url", "sqlite:///usr/local/etc/opensips/sqlite") # CUSTOMIZE ME
===== Código para executar queries no banco de dados ======END
=====

##### USer LOCation module
loadmodule "usrloc.so"
modparam("usrloc", "nat_bflag", "NAT")
modparam("usrloc", "db_mode", 1)
modparam("usrloc", "db_url",
"sqlite:///usr/local/etc/opensips/sqlite") # CUSTOMIZE ME
===== em relação do db_mode acima, temos : =====
# 1 = grava na RAM e no banco imediatamente.
# 2 = Grava só na RAM e depois de 3 segundos no banco.
=====

##### REGISTRAR module
loadmodule "registrar.so"
modparam("registrar", "tcp_persistent_flag", "TCP_PERSISTENT")
modparam("registrar", "received_avp", "$avp(received_nh)")
/* uncomment the next line not to allow more than 10 contacts per AOR */
modparam("registrar", "max_contacts", 1) ***** Permite apenas um contato para cada AOR *****OK
modparam("registrar", "attr_avp", "$avp(attr)") ***** Associa um avp a uma coluna da tabela de location, para
registros. *****
modparam("registrar", "default_expires", 20) ***** Tempo default para expirar um registro de peer que não declarou o expire
time *****
modparam("registrar", "min_expires", 20) ***** Tempo mínimo permitido para definir o expire time *****
modparam("registrar", "max_expires", 30) ***** Tempo máximo para o expire time, definido pelo peer, em
segundos...*****OK

##### ACCounting module
loadmodule "acc.so"
/* what special events should be accounted ? */
modparam("acc", "early_media", 0)
modparam("acc", "report_cancels", 0)
/* by default we do not adjust the direct of the sequential requests.
if you enable this parameter, be sure the enable "append_fromtag"
in "rr" module */
modparam("acc", "detect_direction", 0)

===== Código para gravar dados de mensagens atendidas, perdidas e falhadas ======START=====
#modparam("acc", "failed_transaction_flag", "ACC_FAILED")
/* account triggers (flags) */
#modparam("acc", "db_flag", "ACC_DO")
#modparam("acc", "db_missed_flag", "ACC_MISSED")
#modparam("acc", "db_extra", "callerId=$fU; callerName=$fn")
# 1 - essa linha acima permite salvar dados extras na tabela acc.
# 0 dado extra a ser salvo vai para a coluna callerId e é o id do chamador.
# 2 - essa linha permite salvar dados extras na tabela acc.
# 0 dado extra a ser salvo vai para a coluna callerName e é o Nome legivel do chamador.
===== Código para gravar dados de mensagens atendidas, perdidas e falhadas ======END=====
modparam("acc", "db_url",
"sqlite:///usr/local/etc/opensips/sqlite") # CUSTOMIZE ME

##### AUTHentication modules
loadmodule "auth.so"
loadmodule "auth_db.so"
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "password_column", "password") ***** Manter senha descriptografada no banco de dados *****
# modparam("auth_db", "password_column", "ha1") # ***** Manter senha criptografada no banco de dados *****
modparam("auth_db|uri", "db_url",
"sqlite:///usr/local/etc/opensips/sqlite") # CUSTOMIZE ME
modparam("auth_db", "load_credentials", "")

# ---- db_sqlite params ----
modparam("db_sqlite", "load_extension", "/usr/lib/sqlite3/pcre.so")

```

```

##### DIALOG module
loadmodule "dialog.so"
modparam("dialog", "dlg_match_mode", 1)
modparam("dialog", "default_timeout", 60) ***** Diálogos não podem durar mais que um minuto *****
modparam("dialog", "db_mode", 2)
modparam("dialog", "db_url",
        "sqlite:///usr/local/etc/opensips/sqlite") # CUSTOMIZE ME
# About the db mode parameter =====
#   0 - NO_DB - the memory content is not flushed into DB;
#   1 - REALTIME - any dialog information changes will be reflected into the database immediately.
#   2 - DELAYED - the dialog information changes will be flushed into the DB periodically, based on a timer routine.
#   3 - SHUTDOWN - the dialog information will be flushed into DB only at shutdown - no runtime updates.
#=====

##### NAT modules
loadmodule "nathelper.so"
modparam("nathelper", "natping_interval", 10)
modparam("nathelper", "ping_nated_only", 1)
modparam("nathelper", "sipping_bflag", "SIP_PING_FLAG") #===== Código provavelmente para pingar peer e verificar se
estão online ==
modparam("nathelper", "sipping_from", "sip:opensips@wlan0") #===== Código provavelmente para pingar peer e verificar se
estão online ==
modparam("nathelper", "received_avp", "$avp(received_nh)")

#loadmodule "rtpproxy.so"
#modparam("rtpproxy", "rtpproxy_sock", "udp:localhost:12221") # CUSTOMIZE ME

loadmodule "proto_udp.so"

loadmodule "proto_tcp.so"
#loadmodule "proto_tls.so"
#modparam("proto_tls","verify_cert", "1")
#modparam("proto_tls","require_cert", "0")
#modparam("proto_tls","tls_method", "TLSv1")
#modparam("proto_tls","certificate", "/usr/local/etc/opensips/tls/user/user-cert.pem")
#modparam("proto_tls","private_key", "/usr/local/etc/opensips/tls/user/user-privkey.pem")
#modparam("proto_tls","ca_list", "/usr/local/etc/opensips/tls/user/user-ca-list.pem")

#=====#
#          LÓGICA DE ROTEAMENTO DE MENSAGENS SIP.
#=====

# main request routing logic

route{
    force_rport();
    if (nat_uac_test("23")) {
        if (is_method("REGISTER")) {
            fix_nated_register();
            setbflag(NAT);
            $avp(attr) = "in_another_network";
        } else {
            fix_nated_contact();
            setflag(NAT);
        }
    }

    if (!mf_process_maxfwd_header("10")) {
        sl_send_reply("483", "Too Many Hops");
        exit;
    }

    if (has_totag()) {
        # sequential request withing a dialog should
        # take the path determined by record-routing
        if (loose_route()) {

            # validate the sequential request against dialog
            if ( $DLG_status!=NULL && !validate_dialog() ) {
                xlog("In-Dialog $rm from $si (callid=$ci) is not valid according to dialog\n");
                ## exit;
            }

            if (is_method("BYE")) {
                # do accounting even if the transaction fails
                #do_accounting("db","failed"); ***** Linha comentada. Não quero saber sobre momentos de
ligações encerradas *****
            } else if (is_method("INVITE")) {
                # even if in most of the cases is useless, do RR for
                # re-INVITEs also, as some buggy clients do change route set
                # during the dialog.
                record_route();
            }
            if (check_route_param("nat=yes"))
                setflag(NAT);
        }
        # route it out to whatever destination was set by loose_route()
        # in $du (destination URI).
    }
}

```

```

        route(relay);
    } else {
        if ( is_method("ACK") ) {
            if ( t_check_trans() ) {
                # non loose-route, but stateful ACK; must be an ACK after
                # a 487 or e.g. 404 from upstream server
                t_relay();
                exit;
            } else {
                # ACK without matching transaction ->
                # ignore and discard
                exit;
            }
        }
        sl_send_reply("404","Not here");
    }
    exit;
}

# CANCEL processing
if (is_method("CANCEL"))
{
    if (t_check_trans())
        t_relay();
    exit;
}

t_check_trans();

if ( !(is_method("REGISTER") ) ) {
    if (from_uri==myself)
    {
        # authenticate if from local subscriber
        # authenticate all initial non-REGISTER request that pretend to be
        # generated by local subscriber (domain from FROM URI is local)
        if (!proxy_authorize("", "subscriber")) {
            proxy_challenge("", "0");
            exit;
        }
        if (!db_check_from()) {
            sl_send_reply("403","Forbidden auth ID");
            exit;
        }
        consume_credentials();
        # caller authenticated
    } else {
        # if caller is not local, then called number must be local
        #if (!uri==myself) { ***** Regra de negócios desnecessária. Foi comentada *****
        #    send_reply("403","Rely forbidden");
        #} exit;
    }
}

# preloaded route checking
#if (loose_route()) { ***** Regra de negócios desnecessária. Foi comentada *****
#    xlog("_ERR",
#        "Attempt to route with preloaded Route's [$fu/$tu/$ru/$ci]");
#    if (!is_method("ACK"))
#        sl_send_reply("403","Preload Route denied");
#    exit;
#}

# record routing
if (!is_method("REGISTER|MESSAGE"))
    record_route();

# account only INVITEs
if (is_method("INVITE")) {

    ===== Código para proibir e liberar certas ligações === START =====OK
    if (((ru=="1000")|| (ru=="2000")) && ($fu!="5000" )) {
        t_reply("486", "Chamada proibida !."); # Estah proibido ligar para o número 1000 ou 2000, se o chamador não é
5000($fu==$avp(I_B))
        exit;
    }
    ===== Código para proibir e liberar certas ligações === END =====

    ===== Código para proibir mais que 2 ligações === START =====
    if ($DLG_count==2){
        # Já existem 2 ligações. Retorne sinal de ocupado (486) ou outro aviso SIP coerente (Ex: 603 = decline).
        # Ex: 600 = ocupado em todo lugar.
        t_reply("600", "Sistema já ocupado com 2 ligações. Tente mais tarde.");
        exit;
    }
    ===== Código para proibir mais que 2 ligações === END =====

    # create dialog with timeout
}

```

```

        if ( !create_dialog("B") ) {
            send_reply("500","Internal Server Error");
            exit;
        }

$DLG_timeout = 2 * 60; ##### Estabelece que uma ligação só pode durar 2 minutos *****OK

# === Código para barrar a ligação vinda de qualquer softphone se o 5000 já estiver num dialogo. ===== START =====
if ( get_dialog_info("callee","$var(x)","caller","5000") ) {
    send_reply("486","Residente ocupado. Aguarde e tente novamente.");
    exit;
}
# === Código para barrar a ligação vinda de qualquer softphone se o 5000 já estiver num dialogo. ===== END =====
do_accounting("db");

# ++++++ Registra o identificador do chamador e do chamado em variaveis relacioandas com a dialog atual. =====
START =====
$dlg_val(caller) = $fU;
$dlg_val(callee) = $rU;
# ++++++ Registra o identificador do chamador e do chamado em variaveis relacioandas com a dialog atual. =====
END =====
}

if (!uri==myself) {
    append_hf("P-hint: outbound\r\n");

    # if you have some interdomain connections via TLS
    ## CUSTOMIZE IF NEEDED
    ##if ($rd=="tls_domain1.net"
    ## || $rd=="tls_domain2.net"
    ##) {
    ##    force_send_socket(tls:127.0.0.1:5061); # CUSTOMIZE
    ##}

    route(relay);
}

# requests for my domain
if (is_method("PUBLISH|SUBSCRIBE"))
{
    sl_send_reply("503", "Service Unavailable");
    exit;
}

if (is_method("REGISTER"))
{
    # authenticate the REGISTER requests
    if (!www_authorize("", "subscriber"))
    {
        www_challenge("", "0");
        exit;
    }

    if (!db_check_to())
    {
        sl_send_reply("403", "Forbidden auth ID");
        exit;
    }

    if ( proto==TCP || proto==TLS || 0 ) setflag(TCP_PERSISTENT);

    if (isflagset(NAT))
        setbflag(SIP_PING_FLAG);
    }

    if (!save("location"))
        sl_reply_error();

#===== Código para registrar nome de assinante ===== START =====
# Cadastrar na tabela locations (tabela de usuarios online) o nome legivel do usuario, juntamente com o seu login.
# Será usada a nova coluna callerName para conter o nome legivel.
$avp(ContatoRegister) = ${ct.fields(uri){$.select,0,;}};
xlog("ESSE EH O CONTATO DA PESSOA REGISTRADA AGORA = $avp(ContatoRegister), Caller name = $fn \n ");
if($fn != NULL){
    avp_db_query("UPDATE location SET callerName='$fn' where contact like '$avp(ContatoRegister)%'");
    xlog("CALLER NAME REGISTRADO NO BANCO DE DADOS. \n ");
}
$avp(ContatoRegister) = NULL;
#===== Código para registrar nome de assinante =====END =====
}

exit;
}

if ($rU==NULL) {
    # request with no Username in RURI
    sl_send_reply("484","Address Incomplete");
    exit;
}

```

```

# do lookup with method filtering
if (!lookup("location","m")) {
    if (!db_does_uri_exist()) {
        send_reply("420","Bad Extension");
        exit;
    }

    t_newtran();
    t_reply("404", " Número não responde !");
    exit;
}

if (isbflagset(NAT)) setflag(NAT);

# when routing via usrloc, log the missed calls also
do_accounting("db","missed");

***** APÓS FAZER TODAS AS CHECAGENS, FINALMENTE VAI INICIAR O ENCAMINHAMENTO DA MENSAGEM SIP. *****
route(relay);
}

route[relay] {
    # for INVITEs enable some additional helper routes
    if (is_method("INVITE")) {

        if (isflagset(NAT)) {
            #rtpproxy_offer("ro");
        }

        t_on_branch("per_branch_ops");
        t_on_reply("handle_nat");
        t_on_failure("missed_call");
    }

    if (isflagset(NAT)) {
        add_rr_param(";nat=yes");
    }

    if (!t_relay()) {
        send_reply("500","Internal Error");
    };
    exit;
}

branch_route[per_branch_ops] {
    xlog("new branch at $ru\n");
}

onreply_route[handle_nat] {
    if (nat_uac_test("1"))
        fix_nated_contact();
    if ( isflagset(NAT) )
        #rtpproxy_answer("ro");
    xlog("incoming reply\n");
}

failure_route[missed_call] {
    if (t_was_cancelled()) {
        exit;
    }

    # uncomment the following lines if you want to block client
    # redirect based on 3xx replies.
    ##if (t_check_status("3[0-9][0-9])) {
    ##t_reply("404","Not found");
    ##    exit;
    ##}
}

local_route {
    if (is_method("BYE") && $DLG_dir=="UPSTREAM") {
        # acc_db_request("200 Dialog Timeout", "acc"); ***** Linha comentada, porque não quero contabilizar BYEs
    }
}

```


Anexo II – Instalação de software necessário ao trabalho prático sobre AWS-IOT.

Instalação do JavaScript SDK para AWS IoT, Node.js e teste do ambiente (Windows ou Linux).

Obs: faça as instalações abaixo usando **USUÁRIO ADMINISTRADOR DA MÁQUINA OU ROOT**.

1 – (**Windows**) Instale o Node.js no computador. Para tal, acesse a página do Node.js, faça o download do instalador (LTS) e instale-o. Para fazer o download , acesse a página:

<https://nodejs.org/>

(Linux - Ubuntu)

- (A) `curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash -`
- (B) `sudo apt-get install -y nodejs`

Se necessário, veja mais detalhes aqui: <https://github.com/nodesource/distributions>

2 – Verifique se a instalação correu bem. Execute no prompt de comando (prompt do DOS, no Windows) :

`node -v`

isso mostrar-lhe-á a versão do Node.js instalado agora.

ATENÇÃO: antes de rodar o comando abaixo (item 3) , crie uma pasta chamada AWSIOT (na raiz do drive C, ou D, ou E, ou F, ou H, ou ~, etc...). Então, rode o comando abaixo (item 3) dentro da pasta AWSIOT (use o prompt, no Windows ou Linux). Mas, escolha o drive em que PODE-SE escrever. Ou seja, o aluno deve ter permissão de escrita no mesmo drive onde executará o comando abaixo (no caso Windows).

3 - Instale o SDK para JavaScript, do AWS IoT. Para tal, execute o seguinte comando no prompt:

`npm install aws-iot-device-sdk`

Esse comando instalará o AWS IoT Device SDK no sistema operacional, na pasta AWSIOT.

4 – Testar a instalação. Para isso, fazer o seguinte:

No drive onde foi criada a pasta AWSIOT (no drive C, ou D, ou E, ou F, ou H, ou ~, etc...), crie um arquivo chamado **teste.js**, **dentro** da pasta AWSIOT. O conteúdo desse arquivo deve ser:

```
const deviceModule = require('./node_modules/aws-iot-device-sdk').device;
function processTest() {
  console.log('Teste Ok - Instalacao finalizada.');
}
processTest();
```

5 – Na mesma pasta onde o arquivo teste.js foi criado, executar o seguinte comando:

node teste.js

Bibliografia

- [1] AWS IoT - https://aws.amazon.com/iot/?nc2=h_l3_ap Acessado em 19/06/2016.
- [2] AWS IoT Developer Guide - <https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html> Acessado em 14/03/2020.
- [3] Core Tenets of IoT – Whitepaper - <http://d0.awsstatic.com/whitepapers/core-tenets-of-iot1.pdf> Acessado em 14/03/2020.
- [4] – AWS IoT Developer Resource - <https://aws.amazon.com/iot/developer-resources/> Acessado em 4/09/2016.
- [5] – Use Your Own Certificate - <http://docs.aws.amazon.com/iot/latest/developerguide/device-certs-your-own.html> . Acessado em 7/09/2016.
- [6] – VeriSign CA Root Certificate -
<https://www.symantec.com/content/en/us/enterprise/verisign/roots/VeriSign-Class%203-Public-Primary-Certification-Authority-G5.pem> Acessado em 7/09/2016
- [7] – Video sobre AWS IoT- Aos 33' 25". <https://www.youtube.com/watch?v=tTazcL61JG8> Acessado em 7/09/2016.
- [8] – SDKs - <http://docs.aws.amazon.com/iot/latest/developerguide/iot-sdks.html> Acessado em 7/09/2016.
- [9] - AWS IoT SDK for JavaScript -
<https://github.com/aws/aws-iot-device-sdk-js/blob/master/README.md> Acessado em 10/09/2016.
- [10] – Explicações em video sobre IAM - <https://youtu.be/Ul6FW4UANGc> . Acessado em 11/09/2016.
- [11] – Vídeo da Sensedia sobre Poder das APIs - <https://youtu.be/sAxj8Z2uU9U> Acessado em 14/09/2016.
- [12] – Street Wize de Los Angeles - <http://streetwize.lacity.org/> Acessado em 15/09/2016.
- [13] – Ersi - <http://www.esri.com/> Acessado em 15/09/2016.
- [14] – Xamarin Studio - <https://www.xamarin.com/studio> Acessado em 16/09/2016.
- [15] – APIs Rest para mapas do ERSI - <http://resources.arcgis.com/en/help/arcgis-rest-api/> Acessado em 17/09/2016.
- [16] – Documentação de SDKs Ersi - <https://developers.arcgis.com/documentation/> Acessado em 17/09/2016.
- [17] – Tutorial Ersi com SDK para Android e importação de mapa -
<https://developers.arcgis.com/android/guide/develop-your-first-map-app.htm> Acessado em 17/09/2016.
- [18] - Como open data e APIs estão transformando cidades -
<http://mundoapi.com.br/materias/como-open-data-e-apis-estao-transformando-cidades/> Acessado em 17/09/2016.
- [19] – Cnova - <http://www.cnova.com/en/> Acessado em 17/09/2016.
- [20] – 99APIs - <http://99apis.com> Acessado em 17/09/2016.
- [21] – API para marketplace do Cnova - <https://desenvolvedores.cnova.com/api-portal/> Acessado em 17/09/2016.
- [22] – Web Site Sensedia - <http://sensedia.com/> Acessado em 18/09/2016.
- [23] – Vídeo sobre o Google Classroom - <https://youtu.be/DeOVe2YV2Io> Acessado em 04/04/2020.
- [24] – Google Classroom API - <https://developers.google.com/classroom/> Acessado em 18/09/2016.
- [25] – Video sobre o Moodle - <https://youtu.be/wop3FMhoLGs> Acessado em 18/09/2016.
- [26] – Moodle website - <https://moodle.org/> Acessado em 18/09/2016.
- [27] – Discogs API <https://www.discogs.com/developers/> Acessado em 18/09/2016.
- [28] – The SeatGeek Platform and API - <http://platform.seatgeek.com/> Acessado em 18/09/2016.

- [29] - Spotify Artist Explorer <https://developer.spotify.com/showcase/item/artist-explorer-spotify/> Acessado em 18/09/2016.
- [30] – Video sobre o Spotify Artist Explorer <https://youtu.be/HLOpEynnUCl> Acessado em 18/09/2016.
- [31] – Klarafy - <https://developer.spotify.com/showcase/item/klarafy/> Acessado em 18/09/2018.
- [32] - The Echo Nest- <http://the.echonest.com/> Acessado em 18/09/2018.
- [33] – Yahoo BOSS PlaceFinder API - <https://developer.yahoo.com/boss/placefinder/> Acessado em 18/09/2016.
- [34] – *Roadtrip Mixtape*- <http://labs.echonest.com/CityServer/roadtrip.html> Acessado em 18/09/2016.
- [35] – API do Spotify - <https://developer.spotify.com/> Acessado em 18/09/2016.
- [36] – Github API - <https://developer.github.com/> Acessado em 18/09/2016.
- [37] – API Salesforce - https://trailhead.salesforce.com/pt-BR/content/learn/modules/api_basics/api_basics_overview – Acessado em 04/04/2020
- [38] – API AWS SNS - <http://docs.aws.amazon.com/sns/latest/api/Welcome.html> – Acessado em 19/09/2016.
- [39] – API AWS SQS -
<http://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference>Welcome.html> – Acessado em 19/09/2016.
- [40] AWS IoT API - <http://docs.aws.amazon.com/iot/latest/apireference/Welcome.html> – Acessado em 19/09/2016.
- [41] – APIs da Ford - <https://developer.ford.com/> Acessdo em 20/09/2016.
- [42] – Video sobre APPs e Ford (Android) - <https://youtu.be/n9zorApPNy0> Acessado em 20/09/2016.
- [43] – APIs da Ford – IOs. <https://youtu.be/QxM1bQ1FRds> Acessado em 20/09/2016.
- [44] – Sobre APIs Ford <https://developer.ford.com/pages/applink/> Acessado em 20/09/2016.
- [45] – APIs Nest no 99APIs - <http://99apis.com/card/nest> – Acessado em 20/09/2016.
- [46] – O que são APIs? <http://sensedia.com/blog/apis/o-que-sao-apis-parte-1-introducao/> Acessado em 21/09/2016.
- [47] Video Samsung SmartThings - <https://vimeo.com/142507829?from=outro-embed> Acessado em 21/09/2016.
- [48] – Acesso à API SmartThings - <http://developer.smarththings.com/> Acessado em 21/09/2016.
- [49] – Viajando nas APIs - <http://sensedia.com/blog/apis/viajando-nas-apis-parte-1/> Acessado em 22/09/2016.
- [50] -British Airways - <https://developer.ba.com/> Acessado em 22/09/2016.
- [51]- APIs variadas - <http://sensedia.com/blog/apis/as-apis-que-voce-precisa-conhecer-variedades-2/> Acesssado em 22/09/2016.
- [52] – QuemInova - <https://queminova.catracalivre.com.br/inventa/ferramenta-informa-num-clique-quem-financia-candidatos/> Acessado em 22/09/2016.
- [53] – Valor Econômico - <http://www.valor.com.br/patrocinado/embratel/tendencias-ti-e-telecom/empresas-optam-cada-vez-mais-por-infraestrutura-em-nuvem> – Acessado em 24/09/2016.
- [54] – Webinars da Sensedia - <http://sensedia.com/blog/webinars/> - Acessado em 24/09/2016.
- [55] – Segurança em API – ProgrammableWeb - <http://www.programmableweb.com/api-university/understanding-realities-api-security> – Acessado em 24/09/2016.
- [56] – APIs Facebook - <https://developers.facebook.com/> - Acessado em 24/09/2016.
- [57] – A. Román-Portabales, E. Pérez-Carrera, F. J. González-Castaño and D. Chaves Diéguez - **IMS signaling for Smart Grid home controllers** - *Quobis Networks, Spain, Universidad de Vigo, Spain, Gradiant, Spain* – 2011. DOI: 10.1109/ICCE.2011.5722728
- [58] - Understanding SIP - Today's Hottest Communications Protocol Comes of Age. [Arquivo PDF – White Paper]. Ubiquity. Disponível em

- http://www.ubiquitysoftware.com/solutions/White_Papers.php – Acessado em 2002.
- [59] - B. J. Alan, “SIP: Understanding the Session Initiation Protocol” Editora Artech House, 2001.
- [60] - Rosenber, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. e Schooler, E., “SIP: Session Initiation Protocol – RFC 3261” [formato HTML] *Internet Engineering Task Force*, Junho 2002. Disponível em : <http://www.ietf.org>.
- [61] - SILVA, Davison G. Da, (12 de dezembro de 2003) *Implementação de um sistema SIP para o sistema operacional LINUX*. [Arquivo PDF]. Universidade Estadual de Campinas – Faculdade de Engenharia Elétrica e Computação – Departamento de Comunicações.
- [62] - Interside- <http://www.interside.org/2015/11/o-que-e-ddns.html> – Acessado em 1/10/2016.
- [63] – SIPtopia – *SIP Simplificado* - <https://www.youtube.com/watch?v=gMcUpktyhOE> - Acessado em 1/10/2016.
- [64] - 3GPP IMS- <http://www.3gpp.org/technologies/keywords-acronyms/109-ims> – Acessado em 2/10/2016.
- [65] - Mohammad Abu Lebdeh, Jagruti Sahoo, Roch Glitho, Constant Wette Tchouati - *Cloudifying the 3GPP IP Multimedia Subsystem for 4G and Beyond : A Survey* – IEEE – Janeiro/2016. CIISE Concordia University, Montreal, QC, Canada & Ericsson, Montreal, QC, Canada. Disponível em <https://arxiv.org/pdf/1512.00434.pdf> – Acessado em 2/10/2016.
- [67] – OpenSIPS Core Variable - <http://www.opensips.org/Documentation/Script-CoreVar-2-2> Acessado em 12/10/2016.
- [68] – OpenSIPS – Módulos - <http://www.opensips.org/Documentation/Modules-2-2> Acessado em 12/10/2016.
- [69] – OpenSIPS- Core Functions - <http://www.opensips.org/Documentation/Script-CoreFunctions-2-2> Acessado em 12/10/2016.
- [70] – OpenSIPS – Dicas de instalação - <https://github.com/OpenSIPS/opensips/blob/master/INSTALL> – Acessado em 12/10/2016.
- [71] – WebRTC - <https://webrtc.org/> - Acessado em 14/10/2016.
- [72] - Crislaine da Silva Tripoli, Rodrigo Pimenta Carvalho - *Micros-serviços: características, benefícios e desvantagens em relação à arquitetura monolítica que impactam na decisão do uso desta arquitetura* – ICC- Junho 2016.
- [73] - Gartner, Inc. (NYSE: IT). Nov, 2015 [Online]. Disponível: <http://www.gartner.com/newsroom/id/3165317>.
- [74] - J. Lewis and M. Fowler. Mar, 2014 [Online]. Disponível: <http://martinfowler.com/articles/microservices.html>
- [75] - C. Richardson. 2014 [Online]. Disponível: <http://microservices.io/patterns/monolithic>.
- [76] - S. Newman, “*Building Microservices*”. O'Reilly Media, Inc.. CA: Gravenstein, 2015, pp. 2
- [77] - Caelum. Mar 2015. [Online]. Disponível: <http://blog.caelum.com.br/arquitetura-de-microservicos-ou-monolitica>
- [78] - A. L. Gonzaga. “*Microservices: Padrão Arquitetural para Construir Modernas Aplicações*”. Monografia. Curso de Pós-Graduação em Estratégias em Arquitetura de Software - IGTI. Belo Horizonte, MG, 2015.
- [79] - The Netflix Tech Blog. Jan, 2013. [Online]. Disponível: http://techblog.netflix.com/2013_01_01_archive.html
- [80] - M. Fowler. Jul, 2015. [Online]. Disponível: <http://martinfowler.com/bliki/ServiceOrientedAmbiguity.html>
- [81] - R. C. Martin. Nov. 2009. [Online]. Disponível:. http://programmer.97things.oreilly.com/wiki/index.php/The_Single_Responsibility_Principle
- [82] - W. Vogels. May. 2006. “*Learning from the Amazon technology platform*”. ACM Queue vol 4. No 4 – p 14-22.

- [83] - C. Richardson. May. 2015. [Online]. Disponível:
<https://www.nginx.com/blog/introduction-to-microservices>
- [84] - B. Darrow Nov. 2015. [Online]. Disponível:
<http://fortune.com/2015/11/16/netflix-spinnaker-multi-cloud>.
- [85] - F. Hámoru. Fev. 2016. [Online]. Disponível:
<https://blog.risingstack.com/how-enterprises-benefit-from-microservices-architectures>
- [86] - C. Richardson. 2014 [Online]. Disponível:
<http://microservices.io/patterns>.
- [87] - M. Fowler. June, 2015 [Online]. Disponível:
<http://martinfowler.com/bliki/MonolithFirst.html>.
- [88] - M. Fowler. May, 2015 [Online]. Disponível:
<http://martinfowler.com/bliki/MicroservicePremium.html>
- [89] - C. Richardson. Mar, 2016 [Online]. Disponível:
<https://www.oreilly.com/ideas/why-a-pattern-language-for-microservices>.
- [90] - L. Krause, “Microservices: Patterns and Application: Designing fine-grained services by applying patterns”. 2015.
- [91] - L. Messinger. Feb. 2013. [Online]. Disponível: <https://dzone.com/articles/better-explaining-cap-theorem>
- [92] - Gartner, Inc. (NYSE: IT). July, 2015. [Online]. Disponível:
<https://www.gartner.com/doc/3101023/hype-cycle-application-development>
- [93] - Gartner, Inc. (NYSE: IT). July, 2015. [Online]. Disponível:
<https://www.gartner.com/doc/3102217/hype-cycle-application-architecture>
- [94] - Gartner, Inc. (NYSE: IT). July, 2015. [Online]. Disponível:
<https://www.gartner.com/doc/3096018/hype-cycle-application-services>
- [95] - Gartner, Inc. (NYSE: IT). <http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp>
- [96] - S. Tilkov. Jun, 2015 [Online]. Disponível: <http://martinfowler.com/articles/dont-start-monolith.html>
- [97] - B. Wootton. Abr, 2014 [Online]. Disponível:
<http://highscalability.com/blog/2014/4/8/microservices-not-a-free-lunch.html>
- [98] - M. Fowler. Jul, 2015 [Online]. Disponível: <http://martinfowler.com/articles/microservice-trade-offs.html>
- [99] - R. Wilenach. Jul. 2015 [Online]. Disponível:
<http://martinfowler.com/bliki/DevOpsCulture.html>
- [100] - WS Wiki [Online]. Disponível: <https://wiki.apache.org/ws/WebServiceSpecifications>
- [101] - J. Kress. Jul, 2013. <http://www.oracle.com/technetwork/articles/soa/ind-soa-esb-1967705.htmls>
- [102] - CISS – Software e Serviços. [Online]. Disponível:
<http://www.cissmart.com.br/webcontent/oqueue/>
- [103]- A. Gunter [Online]. Disponível: <https://www.datawire.io/microservices-vs-soa/>
- [104] – Vídeo sobre Microservices, da Sensedia. <https://www.youtube.com/watch?v=Pl0Dccqjrao> – Acessado em 17/10/2016
- [105] – APIX – 2015 - https://youtu.be/GXBYMKaF_OE . Acessado em 18/10/2016.
- [106] Programa Olhar Digital - <https://www.youtube.com/watch?v=IzbSdFQEfJ0> – Acessado em 25/10/2016.
- [107] – Planta virtual Tok & Stok -<https://www.youtube.com/watch?v=DRfiSpZAhGA#t=153.562386> – Acessado em 9/06/2020.
- [108] – BeoHome - <https://www.youtube.com/watch?v=MSy7GpNXTc0> – Acessado em 9/06/2020.
- [109] - ArUco: a minimal library for Augmented Reality applications based on OpenCV
<http://www.uco.es/investiga/grupos/ava/node/26> - Acessado em 9/06/2020.

- [110] – Tech Briefing.net - **PTC's Vuforia Named 'Best Tool' at 2016 Augmented World Expo Auggie Awards for Fourth Consecutive Year**- <http://www.techbriefing.net/modules.php?op=modload&name=News&file=article&sid=377652>. Acessado em 26/10/2016.
- [111] - **Augmented World Expo (AWE) - How Vuforia is Changing the Future of Work** – <https://www.youtube.com/watch?v=IXGxy41xaH0&index=4&list=PLlri17oDqCEWcVCfp78LnkGfl3efPleg7> - Acessado em 2/5/2022.
- [112] Web Site oficial Vuforia – <http://www.vuforia.com> – Acessado em 27/10/2016.
- [113] Web Site Vuforia - **How To Develop for User Defined Targets Using the Native SDKs** - <https://library.vuforia.com/articles/Solution/How-To-Develop-for-User-Defined-Targets-Using-the-Native-SDKs> – Acessado em 27/10/2016.
- [114] Vuforia, Portal do desenvolvedor – **Downloads** - <https://developer.vuforia.com/downloads/tool> – Acessado em 9/6/2020.
- [115] – Vuforia, Portal do desenvolvedor – Download samples - <https://developer.vuforia.com/downloads/samples> Acessado em 9/6/2020.
- [116] – Vuforia – Como usar o Adobe Illustrator para construir VuMarks – <https://library.vuforia.com/articles/Solution/Designing-a-VuMark-in-Adobe-Illustrator> - Acessado em 9/6/2020.
- [117] – Vuforia devices - <http://www.vuforia.com/Devices> – Acessado em 9/6/2020.
- [118] – Wikitude - <http://www.wikitude.com/> - Acessado em 29/10/2016.
- [119] – Wikitude Show Case - **Olympic Games 2016 – Globo Rio App** - <http://www.wikitude.com/showcase/olympics-2016-globo-rio-app/> - Acessado em 9/6/2020.
- [120] Globo.com - **Aplicativo Globo Rio 2016: guia de serviços e realidade aumentada; baixe!** - <http://redeglobo.globo.com/novidades/noticia/2016/07/aplicativo-globo-rio-2016-guia-de-servicos-e-realidade-aumentada-baixe.html> – Acessado em 9/6/2020.
- [121] Wikitude Marketing Video - <https://www.youtube.com/watch?v=wJmeAH4Q7SY> – Acessado em 9/6/2020.
- [122] – Wikitude Studio - <http://studio.wikitude.com/promo> - Acessado em 29/10/2016.
- [123] – Boing – **Smart Glasses** - <https://apx-labs.com/landing/boeing/#video> - Acessado em 9/6/2020.
- [124] -Realidade Aumentada - <http://realidadeaumentada.com.br/category/tutorial/> - Acessado em 29/10/2016.
- [125] - <http://thearea.org/> - Acessado em 9/6/2020.
- [126] - Ingate ® Systems - **Solving the Firewall/NAT Traversal Issue of SIP: Who Should Control Your Security Infrastructure?** - www.Ingate.com . - Acessado em 9/6/2020.