Uma Arquitetura de Microserviços de Internet das Coisas para Casas Inteligentes

An Internet of Things Microservices Architecture for Smart Homes

Rodrigo Felipe Albuquerque P. de Oliveira [1] orcid.org/0000-0002-3552-0204

Carmelo J. A. Bastos-Filho [1] orcid.org/0000-0002-0924-5341

[1] Escola Politécnica de Pernambuco, Universidade de Pernambuco, Recife, Brasil.

E-mail do autor principal: Rodrigo Felipe Albuquerque P. de Oliveira rfapo@ecomp.poli.br

Resumo

O crescimento de dispositivos conectados à Internet impactou de forma positiva a evolução tecnológica da sociedade. A possibilidade de controlar recursos remotamente e interagir com residências não só facilitou a vida do cidadão como trouxe acessibilidade para quem depende de recursos mais sofisticados para viver. O paradigma de Internet das Coisas surge com diversos cenários de automação e arquiteturas para a relação entre software e objetos conectados. Contudo, ainda existe a necessidade de criar ambientes mais concisos para as interações entre estes dispositivos conectados, unificando e replicando configurações de Casas Inteligentes de forma simples e escalável. Uma arquitetura baseada em microserviços de Internet das Coisas possui as características de simplicidade de customização e escalabilidade da infraestrutura necessária para atingir este cenário mais conciso. Os resultados apresentados neste artigo encontraram características de manutenabilidade e reprodutibilidade promissoras para situações reais de Casas Inteligentes.

Palavras-Chave: *Microserviços; Internet das Coisas; Casas Inteligentes; MQTT; RESTful; Docker.*

Abstract

The quick increase in the number of devices connected to the Internet impacted positively in the technological evolution of the society. New possibilities to remotely control the resources and to Interact with smart devices are making the life of the citizens easer. Besides, it brought more sophisticated tool for people who needs facilities for modern life. The Internet of Things (IoT) paradigm appears in several scenarios and efficient architectures to automates connected devices via software are required. However, there is still a need to create environments to allow easer interactions between these devices, unifying smart homes procedures in a simple and scalable way. One architecture based on micro-services for IoT would be a good option to provide customization and scalability. The results presented in this paper show some interesting features for this purpose, such as maintainability and reproducibility.

Key-words: *Microservices; IoT; Smart Homes; MQTT; RESTful; Docker.*

http://dx.doi.org/10.25286/repa.v2i2.547

1 Introdução

É crescente o número de dispositivos conectados à Internet transmitindo, de forma massiva, dados e informações para as mais diversas aplicações e sistemas. Estima-se que até o ano de 2020, 50 bilhões de equipamentos estarão conectados com algum tipo de software através da Internet **[1]**. A diversidade destes dispositivos é bem abrangente, desde sensores de temperatura, câmeras, medidores inteligentes de energia até geladeiras, aparelhos ar condicionado, dentre outros eletrodomésticos. A motivação para conectar estes equipamentos surge com a necessidade de possuir maior controle e gerência sobre eles, além da possibilidade de automatização de tarefas rotineiras como, por exemplo, abrir ou fechar um portão de uma garagem.

Estes equipamentos possuem variados protocolos e serviços para a comunicação com suas aplicações específicas. Contudo, a ausência de um padrão de protocolo ou serviço de comunicação para estes objetos conectados, dificulta o processo de operação e manipulação dos mesmos.

Uma das arquiteturas viáveis para padronizar a comunicação com múltiplos equipamentos, baseia-se nas plataformas de Internet das Coisas **[2][3]**, na qual, de forma geral, estes sensores e equipamentos, conectam-se com servidores e aplicações centradas na computação em nuvem. As plataformas de Internet das Coisas têm contribuído para o crescimento de soluções inovadoras, como apresentado no gráfico da Figura 1.

Figura 1: Ciclo *hype* de tecnologias emergentes.
Fonte: Gartner (2016).

Com o poder de processamento e a escalabilidade da computação em nuvem, a possibilidade de prover serviços que se comuniquem com qualquer tipo de protocolo com estes dispositivos, torna o acesso aos dados e o desenvolvimento de aplicações mais simples. A difusão em larga escala da arquitetura de Internet das Coisas permitiu ao setor de tecnologia uma contribuição significativa com projetos nas áreas da indústria, da energia, da agricultura, cidades inteligentes, dentre outros, conforme a apresentado na Figura 2.

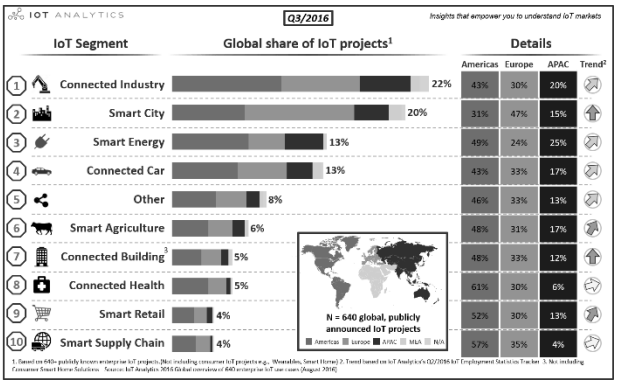

Figura 2: Número de projetos de IoT no mundo.
Fonte:*https://iot-analytics.com/top-10-iot-project-application-areas-q3-2016/ (Acessado em 12/12/2016)*

Apesar das indústrias se beneficiarem com projetos de Internet das Coisas, o mercado potencial de automatização de residências tornou-se uma realidade. O surgimento do conceito de Casas Inteligentes permitiu a usuários comuns a automação de tarefas, o controle dos dispositivos em suas residências, além do acompanhamento do uso dos recursos de uma casa.

1.1 Casas Inteligentes

Alguns dos fatores que contribuíram para o desenvolvimento de Casas Inteligentes incluem o acesso a internet nas residências, o aumento do número de computadores e o acesso as redes móveis **[4]**. Com este cenário de objetos conectados dentro de uma residência, destaca-se três motivações principais para a consolidação do conceito de Casas Inteligentes **[4]**: gerenciamento remoto da casa, acessibilidade e assistência médica residencial para idosos. O gerenciamento remoto das casas permitiu aos usuário o controle dos recursos de água, gás e energia, bem como condições de conforto como aquecimento, refrigeração, iluminação, controle de portas dentre outras facilidades. Por sua vez, a necessidade de contribuir com pessoas idosas e pessoas com necessidades especiais fez com que a utilização e o desenvolvimento de novas tecnologias surgissem para otimizar tarefas do dia a dia nas

residências destes usuários.

A inclusão dessas tecnologias propicia qualidade de vida, além de contribuir para situações de emergência, onde estas casas inteligentes são conectadas a centrais de clínicas hospitalares para atendimento residencial ou mesmo outros serviços públicos de emergência. Na maioria das situações, estas tecnologias e sensores se conectam através de uma rede sem fio residencial. Contudo, apesar de estarem fisicamente conectadas na mesma rede, as aplicações e dispositivos não interagem entre si, o que implica em um cenário com vários dispositivos independentes comunicando-se com suas aplicações específicas.

Uma das causas deste cenário de aplicações independentes em Casas Inteligentes se deve ao fato que cada sensor ou dispositivo possui um protocolo específico de comunicação com sua aplicação. Alguns protocolos e serviços tem se destacado no mundo dos objetos conectados, principalmente os protocolos orientados à Web e protocolos baseados em arquiteturas *publisher/subscriber*.

1.2 Protocolos e Serviços de Internet das Coisas

Os dispositivos atuais já conseguem implementar protocolos de alto nível como o HTTP (*HyperText Transfer Protocol*) **[5]**. Devido aos recursos do próprio protocolo (métodos GET, POST, PUT, DELETE) e a padronização da implementação de serviços web REST (*Representational State Transfer*), a web se tornou programável através de suas API's (*Application Program Interfaces*).

Esta combinação entre um padrão consolidado e dispositivos conectados a internet fez surgir o conceito de WoT (*Web of Things*), na qual requisições realizadas aos equipamentos através de URI's (*Uniform Resources Identifiers*) são efetuadas para consumir dados ou enviar comandos implementados na API do dispositivo.

O HTTP desempenha um ótimo papel como língua franca da internet para comunicação entre dispositivos e suas aplicações **[5]**. Entretanto, a necessidade de comunicação máquina a máquina M2M (*Machine to Machine*) **[6]** fez com que outros protocolos, como por exemplo AMQP (*Advanced Messaging Queuing Protocol*), CoAP (*Constrained Application Protocol*), MQTT (*Message Queue Telemetry Transport)* e *XMPP (Extensible Message and Presence Protocol*), ganhassem espaço devido a simplicidade de implementação, robustez, baixo consumo de energia e portabilidade para dispositivos com baixo recurso de armazenamento e processamento **[6]**.

O MQTT **[7]**, protocolo desenvolvido por engenheiros da IBM, tem se consolidado como um destes protocolos focados em arquiteturas M2M. Possui um modelo baseado em *publisher/subscriber* centrado na comunicação direta com um *broker*, serviço responsável por manipular as mensagens enviadas.

Com capacidade de comunicação com múltiplos clientes, implementação de QoS (*Quality of Service*) na comunicação entre cliente e *broker*, o MQTT controla a manipulação de suas mensagens através do conceito de *topics*, onde cada *topic* possui um identificador e uma prioridade (QoS) na comunicação, conforme a Figura 3.

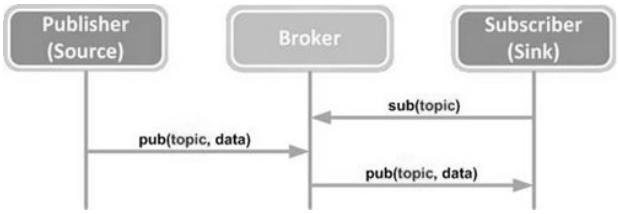

Figura 3: Exemplo de comunicação do protocolo MQTT. Fonte: https://tessel.io/blog/98339010407/mqtt-on-tessel (Acessado em 12/12/2016).

O HTTP e o MQTT estão presentes nas bibliotecas das principais linguagens de programação (C, Java, PHP, Python, Ruby, Javascript). Isto permitiu a estes protocolos tornarem-se um padrão nas principais soluções e arquiteturas de Internet das Coisas.

2 O Manuscrito

O termo Internet das Coisas foi citado pela primeira vez em 1999 **[3]**. Esta primeira definição aplicava-se a sensores de objetos que se comunicavam com sistemas através de protocolos de Internet e enfrentou várias mudanças a longo do tempo. As primeiras mudanças neste conceito surgiram junto com a adoção de protocolos abertos em larga escala como as redes sem fio, o Bluetooth e o RFID (*Radio Frequency Identifier*). A utilização destas tecnologias permitiu a primeira expansão significativa de equipamentos conectados **[3]**.

Com crescimento deste paradigma, o conceito anterior de dispositivos conectados através de redes de sensores e atuadores WSAN's (*Wireless Sensor and Actuators Networks)* começou a ser substituído **[8]**

http://dx.doi.org/10.25286/repa.v2i2.547

gradativamente.

Contudo, a Internet das Coisas herdou as principais características das WSAN's: dispositivos físicos, com conexão através de uma rede sem fio e envio de dados para um *gateway*, onde tal *gateway* é responsável por apresentar as informações dos sensores em algum sistema local. Desta vez, os dispositivos físicos estão diretamente conectados à Internet através das redes de banda larga das residências e empresas, enviando dados para um *gateway* hospedado na computação em nuvem, onde estes dados são acessados por aplicações de qualquer sistema na Web, conforme a Figura 4.

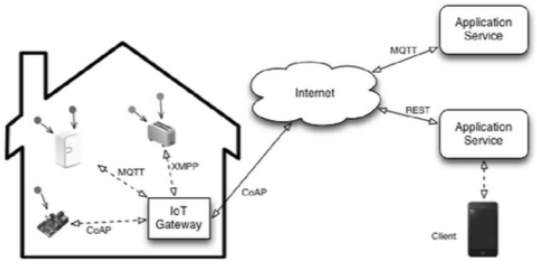

Figura 3: Exemplo de uma arquitetura IoT em uma residência.
Fonte: (Acessado em 12/12/2016).

Logo, os principais elementos que caracterizam a Internet das Coisas são o hardware (com componentes embarcados), um middleware com capacidade de manipular os dados enviados pelos objetos e uma camada de apresentação, em formato de software ou sistema.

O Xively (http://www.xively.com) é um exemplo desta arquitetura que conecta diversos dispositivos em uma plataforma de computação em nuvem que consolida os dados enviados em formato de gráficos e agrupamentos de dados.

É possível definir dentro da aplicação qual tipo de protocolo (MQTT, HTTP, etc) pode ser utilizado pelo dispositivo. Contudo, ainda existe a necessidade de ajustar configurações tanto no equipamento quanto na própria aplicação. Além disso, a aplicação apresenta os dados apenas na visão dos dispositivos e não em visão centralizada de uma residência ou empresa.

Com a necessidade de manter um visão completa dos recursos de uma Casa Inteligente ou Prédio Inteligente, algumas abordagens de centralização foram desenvolvidas. Desde *gateways* residenciais que consolidam os dados de todos os sensores e enviam para um servidor na Nuvem, até técnicas de hibridização de protocolos **[5]**, fazendo com que o mesmo serviço possa receber dados de sensores independentes e interpretar estas informações em um único ponto de manipulação de mensagens.

O desafio na arquitetura de Internet das Coisas persiste na definição de um modelo que possa ser repetível e escalável sem a necessidade de muitas configurações ou ajustes tanto em *gateways* residenciais como em servidores na Nuvem.

Uma estratégia viável pode ser desenvolvida criando serviços simplificados baseados em perfis pré-existentes. Desta forma, este conjunto de microserviços seria capaz de replicar e escalar a maior quantidade de cenários de Casas Inteligentes apenas com acesso a repositórios destes perfis.

3 Microserviços

Os Microserviços **[9]** representam um estilo de arquitetura de software onde uma aplicação com uma suíte de pequenos serviços interage com seu próprio processo e seus mecanismo simplificados de comunicação. Os benefícios desta arquitetura estão na capacidade de trabalhar em ambientes de produção de software de forma simplificada, fácil manutenção e migração rápida, resultando no menor impacto em toda infraestrutura da aplicação.

Uma das motivações para o desenvolvimento dos Microserviços é quando comparado a estrutura de um software monolítico onde um ciclo de versão com falhas pode deixar indisponível toda aplicação exatamente porque todo o sistema está compartilhando recursos. Na infra-estrutura de Microserviços é possível manter uma horizontalidade de micro instâncias do software monolítico da aplicação, onde caso uma destas instâncias falhe qualquer outra instância assume o acesso deste sistema, como apresentado na Figura 5.

Diferente do conceito de máquinas virtuais, onde uma aplicação é executada sobre um sistema operacional específico do hóspede, os microserviços utilizam-se do mesmo sistema operacional, segmentando-se através do conceito de LXC (*Linux Containers*) **[10]**.

Com este recurso de isolamento de aplicações dentro de containers, o benefício da aplicação não se restringe apenas a disponibilidade do sistema, mas também a escalabilidade e manutenabilidade do código do programa devido a eliminação de dependências.

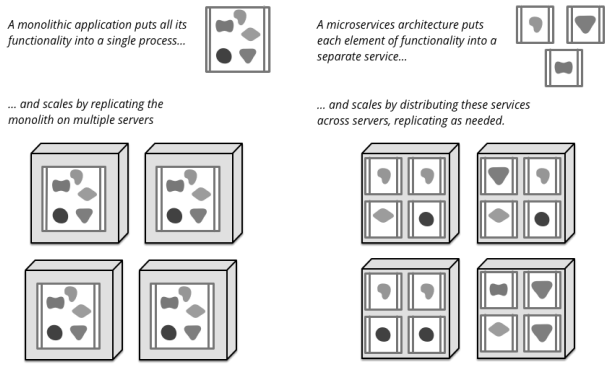

Figura 5: Diagrama comparativo Monolítico x Microserviços Fonte:http://www.martinfowler.com/articles/microservices.html (Acessado em 12/12/2016).

Existem algumas *engines* atuais para utilização de microserviços. Dentre as mais utilizadas está o Docker **[10][11]**. O Docker é uma plataforma de microserviços que utiliza-se do conceito de *containers*, imagens e fórmulas, conhecidas por Dockerfiles. De forma simplificada, o Docker cria uma API de comunicação entre o *container* e o sistema operacional anfitrião, isolando a aplicação inserida no *container* de qualquer interação com o sistema operacional. A criação dos *containers* pode ser realizada através de imagens disponibilizadas pela comunidade do Docker ou através da criação das fórmulas a partir dos Dockerfiles. O Dockerfile é uma fórmula para a criação do container que contempla todas configurações, características e dependências necessárias para a aplicação contida na instância.

A *engine* do Docker está disponível para os principais sistemas operacionais, além de possuir um serviço de repositório de *containers* em computação na nuvem, onde é possível executar os Dockerfiles de forma remota, como apresentado na Figura 6.

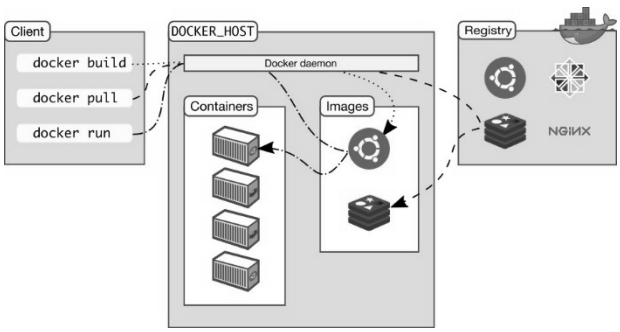

Figura 6: Relação do Dockerfiles com Client e Registry Fonte:https://docs.docker.com/engine/understanding-docker/ (Acessado em 12/12/2016)

O Docker é portável para outras arquiteturas de computadores como, por exemplo, para processadores ARM. Com esta característica, a *engine* pode funcionar em computadores com poucos recursos ou dispositivos simples como, por exemplo, Raspberry Pi ou Beaglebone. A combinação de microserviços com Docker, dispositivos portáteis como o Raspberry Pi e repositórios em computação na nuvem, permite a configuração de uma arquitetura distribuída de Internet das Coisas que pode ser aplicada em Casas Inteligentes para gerenciamento de recursos remotos.

4 Arquitetura de Microserviços para Internet das Coisas

A arquitetura proposta neste artigo consiste em validar a viabilidade técnica da utilização de microserviços distribuídos em um *gateway* residencial e em um servidor na nuvem, para o consumo de informações de sensores instalados em uma residência, conforme apresentado na Figura 7.

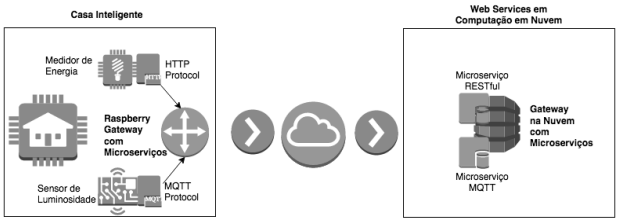

Figura 7: Arquitetura de Microserviços de Internet das Coisas para Casas Inteligentes.

Para este projeto foi levado em consideração o gerenciamento de energia de um cômodo de uma residência através de um medidor inteligente e a análise de luminosidade do mesmo cômodo utilizando um sensor de luminosidade.

Os dados destes dois sensores serão coletados pelo Raspberry *Gateway* através de duas aplicações clientes utilizando microserviços e as informações consumidas serão enviadas para um servidor na Nuvem da AWS (*Amazon Web Services*). Estes dados serão consolidados em uma única aplicação e apresentados em uma plataforma Web. Não foram levados em consideração para este projetos aspectos de segurança como autenticação e criptografia de dados.

http://dx.doi.org/10.25286/repa.v2i2.547

4.1 Implementação

O primeiro passo para o desenvolvimento da arquitetura foi a definição dos sensores a serem utilizados. O medidor inteligente proposto nesta arquitetura é o Emon Pi. O Emon Pi é um medidor inteligente desenvolvido pela equipe do projeto *Open Energy Monitor* (http://openenergymonitor.org). Utiliza como tecnologia um Raspberry Pi 2 B+ adaptado com um sensor de corrente que se conecta através de transformadores de corrente para realizar as medições e entregar dados de consumo de energia.

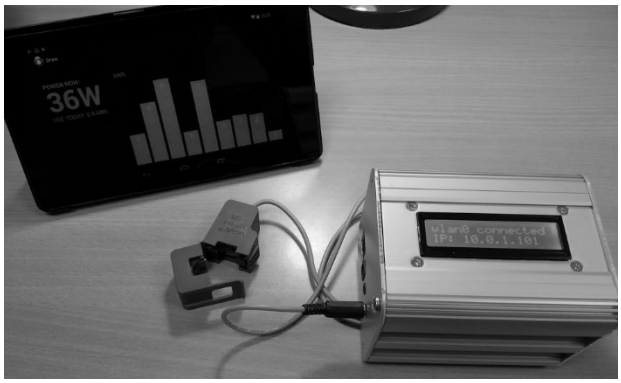

Figura 8: Fotos do Emon Pi

O Emon Pi possui uma aplicação web embarcada para acesso aos dados de consumo, corrente e tensão. Estes dados também são acessados através de uma API RESTful desta aplicação web. As requisições a esta API retornam valores no formato de um JSON(*Javascript Object Notation*).

O sensor de luminosidade foi desenvolvido com o Nodemcu, um *firmware* de código aberto para projetos de Internet das Coisas utilizando o módulo sem fio SOC ESP8266. Este firmware suporta *scripts* desenvolvidos na linguagem de programação Lua. O módulo ESP8266 permite a rápida prototipação e a leitura analógica de dados de outros sensores, neste caso, dados de uma fotocélula, conforme a Figura 9.

Como o módulo ESP8266 não possuía um código nativo para medição de luminosidade, um *script* na liguagem Lua foi desenvolvido para coletar os dados medidos pela fotocélula e enviar os dados utilizando o protocolo MQTT para um *broker*.

O *gateway* para a centralização dos dados coletados dos dois sensores foi construído utilizando a placa do Raspberry Pi 3. O Raspberry Pi é computador de dimensões reduzidas(8,5cm x 4,9cm), desenvolvido no Reino Unido utilizado para fins educacionais e para projetos de Internet das Coisas.

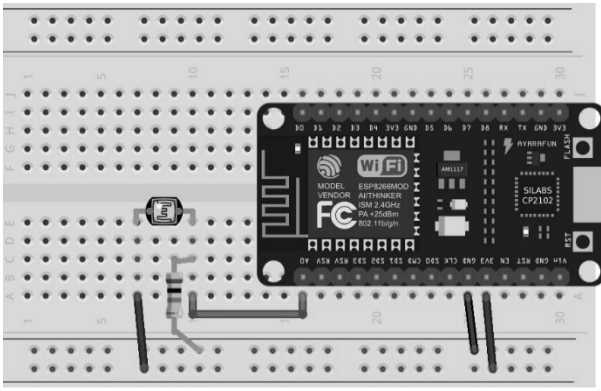

Figura 9: Ilustração do diagrama de interligação do Módulo ESP8266 conectado a uma fotocélula

A arquitetura do processador do Rapsberry Pi é baseada em ARM e está na versão 7. Com 01(um) GB de memória RAM e cartão SSD de 08(oito) GB para armazenamento de memória de massa. Possui um sistema operacional padrão baseado em Linux, chamado Raspbian.

Para a construção do *gateway* com microserviços foi necessário compilar uma versão específica do Docker 1.12.3 para sistema operacional Raspbian na arquitetura ARMv7.

Após a configuração do *gateway* foi desenvolvido duas aplicações para dois microserviços distintos. O primeiro microserviço possui uma aplicação na linguagem de programação Python, na versão 2.7.4, responsável por efetuar uma requisição a cada minuto para o medidor de energia, coletar o JSON com os dados de consumo e enviar para o webservice na AWS.

O segundo microserviço embarcado no *gateway* do Raspberry Pi possui uma aplicação na linguagem Python, na versão 2.7.4, responsável por levantar um servidor MQTT, receber os dados enviados pelo sensor de luminosidade através deste protocolo e envia-lós para um broker MQTT configurado no servidor da AWS. A construção dos *containers* foi baseado no exemplo do Dockerfile apresentado na Figura 10.

Para validação da arquitetura foram criados mais dois microserviços locais no *gateway* simulando o papel do servidor hospedado na AWS. Estes dois microserviços contemplaram um *container* com o MongoDB versão 3.4, banco de dados não relacional baseado na manipulação de objetos em formato de documentos, solução adequada para armazenar os dados no formato JSON. E um segundo *container* contemplando um serviço web RESTful desenvolvido em Python, na versão 2.7.4, com o framework Web Flask, na versão 0.11, responsável por consolidar os dados enviados pelos dois sensores e apresentar no formato JSON as duas informações em uma aplicação Web.

https://raw.githubusercontent.com/rfapo/smar

```
FROM resin/rpi-raspbian:wheezy

RUN apt-get update && apt-get install -y \
    python \
    python-dev \
    python-pip \
    python-virtualenv \
    --no-install-recommends && \
    rm -rf /var/lib/apt/lists/*

RUN pip install pymongo

RUN pip install paho-mqtt

COPY ./app /app

WORKDIR /app

CMD ["python", "./mqtt-client-v2.py"]
```

Figura 10: Módulo ESP8266 conectado a uma fotocélula

Os quatro microserviços foram executados localmente no *gateway* do Raspberry Pi e não foi registrado nenhum problema de performance durante a execução.

Com os microserviços executando baseados nos Dockerfiles construídos para eles, o próximo passo realizado foi a migração dos dois últimos *containers* para o servidor na AWS. Para isto, os Dockerfiles dos *containers* foram salvos no repositório de *containers* da própria Docker no endereço https://hub.docker.com/r/rfapo/.

Por fim, os microserviços foram executados no servidor da AWS. O servidor na Amazon possui a seguinte configuração: processador intel Quad Core de 2GHz, 01(um) GB de memória RAM e disco SSD de 40GB, executando o Ubuntu Linux, na versão 16.04.

A instalação do Docker no servidor utilizou o repositório do próprio sistema operacional. Após esta configuração, os dois últimos *containers* foram executados a partir do repositório da Docker.

Para verificar o funcionamento da arquitetura, os dois últimos *containers* foram desligados do *gateway* local do Raspberry Pi e o tráfego foi redirecionado para os microserviços hospedados no servidor da AWS, conforme as Figuras 11 e 12.

```
pi@raspberrypi:~ $ sudo docker run -tip 1883:1883 -p 9001:9001 pascaldevink/rpi-mosquitto
1480004202: Error: Unable to open log file /mqtt/log/mosquitto.log for writing.
1480004202: mosquitto version 1.4.4 (build date Thu, 17 Sep 2015 16:11:28 +0100) starting
1480004202: Config loaded from /mqtt/config/mosquitto.conf.
1480004202: Opening websockets listen socket on port 9001.
1480004202: Opening ipv4 listen socket on port 1883.
1480004202: Opening ipv6 listen socket on port 1883.
1480004281: New connection from 192.168.25.168 on port 1883.
1480004281: New client connected from 192.168.25.168 as ESP8266esp03 (c1, k180).
```

Figura 11: Microserviço MQTT executado no Raspberry Pi

```
ubuntu@ip-172-30-0-34:~$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATU
S                   PORTS                                        NAMES
be93f4f83727        rfapo/flask-app     "/usr/bin/supervisord"   12 minutes ago      Up 12
minutes             0.0.0.0:80->80/tcp, 443/tcp   flask-app
c48c9ed5dc2e        rfapo/mongodb       "/usr/bin/mongod"        2 days ago          Up 2
days                0.0.0.0:27017->27017/tcp      mongo_instance_001
ubuntu@ip-172-30-0-34:~$
```

Figura 12: Microserviço RESTful e Banco de Dados executados na AWS.

Com a comunicação dos dois sensores utilizando os microserviços fim a fim, a consulta das informações no formato JSON podem ser acessadas no mesmo endereço da aplicação Web com as seguintes URI's: htttp://<endereço_da_aplicação>/energyData e htttp://<endereço_da_aplicação>/luminosity, conforme apresentado nas Figuras 13 e 14.

Todos os códigos do projeto e configurações dos *containers* estão disponíveis no seguinte repositório: https://github.com/rfapo/smarthome_microservices. Além disso as próprias imagens dos *containers* estão disponíveis no repositório do Docker no seguinte endereço: https://hub.docker.com/r/rfapo/.

34.192.244.29

[{"timestamp": 1480002775.985475, "unit": "W", "_id": {"$oid": "58370f0c7d59a3000ef67d8e"}, "name": "power1", "value": 7}, {"timestamp": 1480004389.183184, "unit": "W", "_id": {"$oid": "583713257d59a3000ff67d8e"}, "name": "power1", "value": 18}, {"timestamp": 1480004392.866678, "unit": "W", "_id": {"$oid": "583713297d59a3000ef67d8f"}, "name": "power1", "value": 16}, {"timestamp": 1480004396.323933, "unit": "W", "_id": {"$oid": "5837132c7d59a3000ff67d8f"}, "name": "power1", "value": 16}, {"timestamp": 1480004399.719, "unit": "W", "_id": {"$oid": "5837132f7d59a3000ef67d90"}, "name": "power1", "value": 16}, {"timestamp": 1480004403.301194, "unit": "W", "_id": {"$oid": "583713337d59a3000ff67d90"}, "name": "power1", "value": 17}, {"timestamp": 1480004406.759843, "unit": "W", "_id": {"$oid": "583713367d59a3000ef67d91"}, "name": "power1", "value": 24}, {"timestamp": 1480004410.162686, "unit": "W", "_id": {"$oid": "5837133a7d59a3000ff67d91"}, "name": "power1", "value": 24}, {"timestamp": 1480004413.61656, "unit": "W", "_id": {"$oid": "5837133d7d59a3000ef67d92"}, "name": "power1", "value": 24},

Figura 13: Dados do medidor de consumo

34.192.244.29

[{"luminosity": "42", "_id": {"$oid": "5835e73b77edca67a1af15fd"}, "timestamp": 1479927552.759442}, {"luminosity": "84", "_id": {"$oid": "5835ed9177edca6b88b278bd"}, "timestamp": 1479929212.9624}, {"luminosity": "305", "_id": {"$oid": "58371331bfeafebcd8b0bf12"}, "timestamp": 1480004401.962838}, {"luminosity": "366", "_id": {"$oid": "5837136dbfeafebcd8b0bf13"}, "timestamp": 1480004461.967825}, {"luminosity": "409", "_id": {"$oid": "583713aabfeafebcd8b0bf14"}, "timestamp": 1480004522.285504}, {"luminosity": "476", "_id": {"$oid": "583713e5bfeafebcd8b0bf15"}, "timestamp": 1480004581.98321}, {"luminosity": "459", "_id": {"$oid": "58371421bfeafebcd8b0bf16"}, "timestamp": 1480004641.789246}]

Figura 14: Dados do sensor de luminosidade

http://dx.doi.org/10.25286/repa.v2i2.547

5 Conclusões

A utilização de microserviços para a arquitetura proposta mostrou-se promissora quanto a reprodutibilidade e manutenabilidade. Com o recurso de geração e execução de novos *containers* a partir de uma fórmula pré-definida com os Dockerfiles permitiu a migração de uma aplicação de ambiente com arquitetura distinta para um novo ambiente sem maiores dificuldades. A existência de um repositório da própria ferramenta permite a criação de um *Marketplace* de instâncias que podem atender os mais diversos protocolos e serviços de comunicação com dispositivos de Casas Inteligentes.

Mesmo compreendendo que o Docker executou os quatros *containers* dentro do Raspberry Pi, não houve um teste de stress para verificar o custo de processamento da ferramenta, o que limitaria a execução em plataformas mais simples que o próprio Raspberry Pi.

A possibilidade de manter cenários distribuídos com microserviços faz desta ferramenta indispensável para cenários de missão crítica onde a disponibilidade da aplicação é um fator crucial.

Apesar de existir um processo de instalação diferente entre as arquiteturas do Raspberry Pi e do servidor Linux na AWS, não houve necessidade de algum tipo de customização das configurações do Docker.

Ainda existe a necessidade de avaliar de forma mais aprofundada a performance das máquinas utilizadas como anfitriãs dos *containers* e qual o impacto da execução delas tanto para os *containers* quanto para máquina hospedeira.

Outra análise a ser realizada nos trabalhos futuros é quanto aos requisitos de segurança dos containers e aspectos de segurança de rede, visto que o *container* compartilha da mesma interface física e das bibliotecas do sistema operacional.

Um estudo sobre a orquestração de microserviços em ambientes de produção com uma grande quantidade de *containers* também será necessário, além da análise de composição de múltiplos Dockerfiles simultâneos.

Referências

[1] "Number of connected devices worldwide form 2012 to 2020", Statista, 2016[Online]. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[2] Melis, A., Mirri, S., Prandi, C., Prandini, M., Salomoni, P., & Callegati, F. (n.d.). CrowdSensing for Smart Mobility through a Service Oriented Architecture.

[3] GUBBI, Jayavardhana et al. Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems, v. 29, n. 7, p. 1645-1660, 2013.

[4] Chan, M., Estève, D., Escriba, C., & Campo, E. (2008). A review of smart homes-Present state and future challenges. Computer Methods and Programs in Biomedicine, 91(1), 55–81. http://doi.org/10.1016/j.cmpb.2008.02.001

[5] M. Collina, G. E. Corazza and A. Vanelli-Coralli, "Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST," Personal Indoor and Mobile Radio Communications (PIMRC), 2012 IEEE 23rd International Symposium on, Sydney, NSW, 2012, pp. 36-41.

[6] A. Al-Fuqaha, A. Khreishah, M. Guizani, A. Rayes and M. Mohammadi, "Toward better horizontal integration among IoT services," in IEEE Communications Magazine, vol. 53, no. 9, pp. 72-79, September 2015.

[7] ISO/IEC 20922 Information Technology – Message Queue Telemetry Transport (MQTT) v3.1.1.

[8] P. Desai, A. Sheth and P. Anantharam, "Semantic Gateway as a Service Architecture for IoT Interoperability," Mobile Services (MS), 2015 IEEE International Conference on, New York, NY, 2015, pp. 313-319.

[9] Fowler, M., & Lewis, J. (2014). Microservices. Viittattu, 28, 2015.

[10] Stubbs, J., Moreira, W., & Dooley, R. (2015). Distributed Systems of Microservices Using Docker and Serfnode. Proceedings - 7th International Workshop on Science Gateways, IWSG 2015, 34–39. http://doi.org/10.1109/IWSG.2015.16

[11] Boettiger, C. (2015). An introduction to Docker for reproducible research. ACM SIGOPS Operating Systems Review, 49(1), 71–79. http://doi.org/10.1145/2723872.2723882

[12] A. Krylovskiy, M. Jahn and E. Patti, "Designing a Smart City Internet of Things Platform with Microservice Architecture," Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on, Rome, 2015, pp. 25-30.

[13] U. Hunkeler, H. L. Truong and A. Stanford-Clark, "MQTT-S — A publish/subscribe protocol for

Wireless Sensor Networks," Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on, Bangalore, 2008, pp. 791-798.

[14] N. De Caro, W. Colitti, K. Steenhaut, G. Mangino and G. Reali, "Comparison of two lightweight protocols for smartphone-based sensing," 2013 IEEE 20th Symposium on Communications and Vehicular Technology in the Benelux (SCVT), Namur, 2013, pp. 1-6.

[15] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[16] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120, March 2011, <http://www.rfc-editor.org/info/rfc6120>.

[17] V. Karagiannis, "A survey on application layer protocols for the internet of things.Transaction on IoT and Cloud Computing", v. 1, n. 1, 2015.

[18] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," in IEEE Communications Surveys & Tutorials, vol. 17, no. 4, pp. 2347-2376, Fourthquarter 2015.

[19] Yashiro, T., Kobayashi, S., Koshizuka, N., & Sakamura, K. (2013). An Internet of Things (IoT) architecture for embedded appliances. 2013 IEEE Region 10 Humanitarian Technology Conference, R10-HTC 2013, 314–319.

[20] Newman, S. (2015). Building Microservices. " O'Reilly Media, Inc.".

[21] Familiar, B. (2015). Microservices, IoT, and Azure. Apress.

[22] Datta, S. K., Bonnet, C., & Nikaein, N. (2014). An IoT Gateway Centric Architecture to Provide Novel M2M Services, 514–519.

[23] https://iot-analytics.com/top-10-iot-project-application-areas-q3-2016/

[24] Chan, M., Estève, D., Escriba, C., & Campo, E. (2008). A review of smart homes-Present state and future challenges. Computer Methods and Programs in Biomedicine, 91(1), 55–81. http://doi.org/10.1016/j.cmpb.2008.02.001

[25] Melis, A., Mirri, S., Prandi, C., Prandini, M., Salomoni, P., & Callegati, F. (n.d.). CrowdSensing for Smart Mobility through a Service Oriented Architecture.

[26] Boettiger, C. (2015). An introduction to Docker for reproducible research. ACM SIGOPS Operating Systems Review, 49(1), 71–79. http://doi.org/10.1145/2723872.2723882

[27] Chan, M., Campo, E., Estève, D., & Fourniols, J. Y. (2009). Smart homes - Current features and future perspectives. Maturitas, 64(2), 90–97. http://doi.org/10.1016/j.maturitas.2009.07.014

http://dx.doi.org/10.25286/repa.v2i2.547