
Universidade Federal de Pernambuco

Centro de Informática

Graduação em Engenharia da Computação

# KNoTPy: Uma biblioteca para acessar dados de devices conectados na plataforma KNoT

Ramon Henrique Pereira Ribeiro

Trabalho de Graduação

Recife, Pernambuco

Outubro de 2018

Universidade Federal de Pernambuco
Centro de Informática

Ramon Henrique Pereira Ribeiro

# KNoTPy: Uma biblioteca para acessar dados de devices conectados na plataforma KNoT

Trabalho apresentado ao Curso de Graduação em Engenharia da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Engenharia da Computação.

**Orientador:** Kiev Gama

Recife, Pernambuco
Outubro de 2018

*Dedico este trabalho a Deus e minha família.*

# Agradecimentos

Quero agradecer primeiramente aos meus pais por sempre me incentivaram a perseguir o que eu queria para minha carreira sempre acreditando em mim e no meu esforço tanto para entrar quanto para concluir esse curso.

Gostaria de agradecer ao Centro de Informática (CIn) da UFPE e todas as pessoas que conheci ao passar por ele. Tanto professores e amigos foram importantes para tornar a pessoa que sou hoje.

Também gostaria de agradecer ao CESAR (Centro de Estudos e Sistemas Avançados do Recife), onde faço estágio atualmente, pela oportunidade de trabalhar num projeto tão amplo onde pude aprender várias tecnologias novas e que deu origem a esse trabalho de graduação.

*"Most good programmers do programming not because they expect to get paid or get adulation by the public, but because it is fun to program."*

—Linus Tovalds

# Resumo

A Internet of Things (IoT - *Internet of Things*) tem o objetivo de conectar objetos e fazer com que os mesmos troquem informações que permitam ajudar usuários a tomar decisões com os dados gerados por esses dispositivos conectados. Com o montante de dados que a IoT irá gerar, não será possível, para dispositivos com baixo poder de processamento e armazenamento, lidarem com todos esses dados. Para acessar esses dados, que ficarão armazenado em uma cloud, é preciso usar as API dessa cloud, e hoje em dia há muitos serviços de cloud voltados para IoT como AWS IoT, Google cloud IoT, Konker, FIWARE, entre outros. A meta-plataforma de IoT, KNoT, promete fornecer interoperabilidade para abstrair os dispositivos conectados para um conjunto de serviços de cloud. Porém o desenvolvedor ainda precisa gastar tempo para estudar a API daquela cloud para conseguir acessar quais operações é possível se fazer nos dispositivos conectados, e se por alguma decisão de projeto for necessário mudar de uma cloud para outra esse tempo de estudo terá sido perdido. Este trabalho visa criar um biblioteca de funções canônicas para acessar dados de um dispositivo conectado nas plataformas de cloud suportadas pelo KNoT.

**Palavras-chave:** IoT, Cloud, Plataforma, KNoT, API.

# Abstract

The Internet of Things (IoT) aims to connect objects and get them to exchange information that will help users make decisions with the data generated by those connected devices. With the amount of data that IoT will generate, it will not be possible for devices with low processing and storage power to handle all of this data. To access this data, which will be stored in a cloud, you need to use the API's of this cloud, and nowadays there are many IoT cloud services such as AWS IoT, Google cloud IoT, Konker, FIWARE, among others. The IoT meta-platform, KNoT, promises to provide interoperability to abstract connected devices for a set of cloud services. But the developer still needs to study the API of that cloud to get access to what operations it is possible to do in devices, and if for some design decision it is necessary to change from one cloud to another this time of study has been lost. This work aims to create a library of canonical functions to access data from a connected device in the cloud platforms supported by KNoT.

**Keywords:** IoT, Platform, API.

# Lista de Tabelas

# Lista de Figuras

# Siglas

**API** Application Program Interface. 2

**DSL** Domain Specific Language. 3

**IoT** Internet of Things. 1, 2

**KNoT** Network of Things. 2

**NFC** Near Field Communication. 1

**RFID** Radio Frequency IDentification. 1

# Sumário

# Capítulo 1

# Introdução

Neste capítulo é apresentada a motivação para a escrita deste trabalho, assim como o objetivo e a estrutura de tópicos do mesmo.

## 1.1 Motivação e contextualização

A internet das coisas, ou seu acrônimo em inglês *Internet of Things (IoT)*, promete ser a terceira onda de tecnologia de informação depois da internet e a rede de dispositivos móveis. A tecnologia de Internet of Things pode facilitar a integração do mundo físico com o mundo digital [4]. A IoT tem constantemente crescido na indústria[5], tem o potencial de mudar nossas vidas e como nós vemos e usamos a internet tendo várias visões para isso[6], tais como o uso de redes Radio Frequency IDentification (RFID) e Near Field Communication (NFC) além de todos os dispositivos conectados rodarem sobre protocolo IP. De fato, espera-se que com IoT, bilhões de novos tipos de dados e eventos serão gerados e explorá-los será a chave fundamental para a construção de um mundo mais inteligente [7] revolucionando o modo como vivemos, trabalhamos e cada aspecto da nossa vida.

Porém a granularidade de informações que um dispositivo conectado em $IoT$ pode fornecer é muito grande, gerando um problema para os dados serem compartilhados entre dispositivos, já que os dispositivos não sabem que tipo de dados podem receber. Essa granularidade nos dados é vista, pois podemos ter uma quantidade enorme de dispositivos e das mais diversas formas possíveis. Essa heterogeneidade dos dispositivos são um dos grandes desafios de $IoT$, que algumas plataformas estão tentando resolver.

Uma plataforma de $IoT$ nada mais é do que um *middleware* que fornece a infraestrutura

para permitir e facilitar que usuários e desenvolvedores interajam com objetos inteligentes. Hoje existem várias plataformas, para resolver problemas de *IoT*, nas seguintes categorias [8], que serão mais discutidas no capítulo 2:

- Integração de sensores e atuadores
- Propriedade dos dados
- Processamento e compartilhamento de dados
- Suporte a desenvolvedores de aplicações
- Formação de ecossistemas de IoT

No entanto, interações entre plataformas de *IoT* são limitadas e custosas. E a interoperabilidade entre plataformas é difícil devido a falta de um consenso de padrões para que elas possam se comunicar. Na tentativa de minimizar esses problemas há plataformas como o KNoT, Network of Things [3], um projeto desenvolvido pelo Centro de Estudos e Sistemas Avançados do Recife (CESAR), que vem com o objetivo de fornecer uma infraestrutura para dar interoperabilidade entre plataforma *clouds-based* e protocolos de rádio sem fio. Com isso o desenvolvedor pode usar a API da plataforma *cloud-based* que ele selecionar para poder acessar os dados dos dispositivos conectados que ele conectou na sua infraestrutura.

Porém o KNoT mostra uma lacuna em uma das categorias listadas acima que é o suporte a desenvolvedores de aplicações, fazendo com que os desenvolvedores ainda precisem consultar as Application Program Interface (API)'s da plataformas que eles estão usando. Mineraud et al. [8] considera que o ideal para o desenvolvedor da aplicação é o uso de uma API comum para facilitar o desenvolvimento entre aplicações *cross-platform*.

## 1.2 Objetivos

O principal objetivo deste trabalho é resolver um das lacunas do trabalho de Mineraud. et. al. [8], na categoria de suporte ao desenvolvedor de aplicação. Para isso é usado a meta plataforma de IoT, KNoT, que permite facilitar a integração de plataformas de IoT, possibilitando que um grande número de plataformas possa ser integrado, na figura 1.1 é mostrada a contribuição para o conjunto de bibliotecas do KNoT.

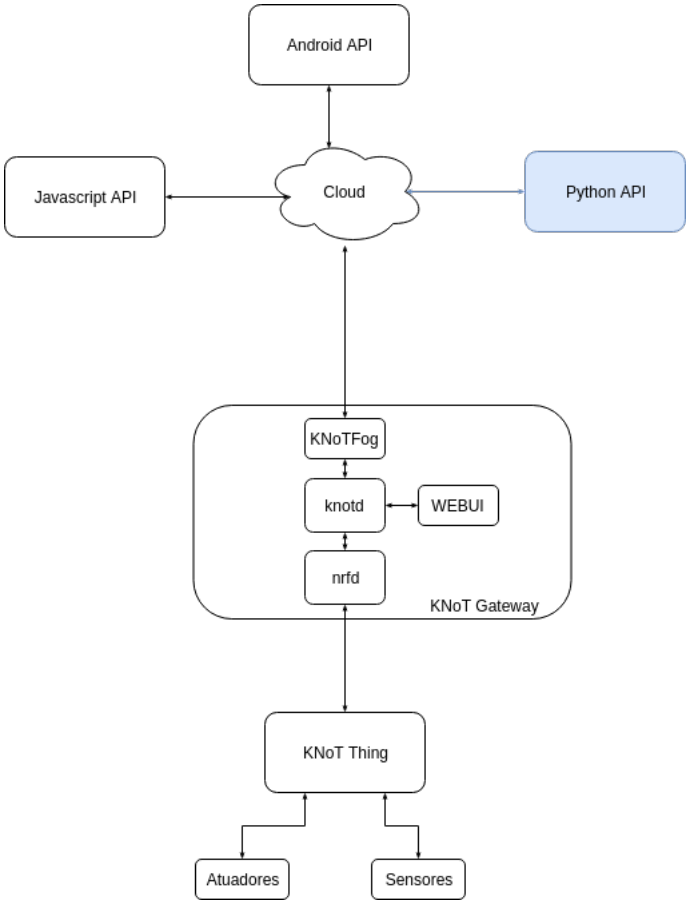


Figura 1.1: Contribuição para conjunto de bibliotecas do KNoT

É desenvolvido então nesse trabalho uma biblioteca chamada KNoTPy, onde o principal objetivo é fornecer ao desenvolvedor um conjunto de funções que permita a ele ter uma aplicação cross-platform, para acessar e configurar os dispositivos conectados a plataformas de clouds suportadas pelo KNoT. Fornecendo ao desenvolvedor métodos mais simples em Python para acessar os dados dos dispositivos e também prover uma arquitetura de software multi cloud para bibliotecas KNoT, fornecendo assim uma Domain Specific Language (DSL) que Mineraud. et. al. considera ser umas das recomendações para as plataformas de IoT.

Python foi escolhida pois, é uma linguagem open-source que é amplamente usada em aplicação de tratamento de dados e aprendizagem de máquina. Como IoT gera muitos dados dos devices conectados esses dados ainda precisam ser consultados e tratados para as diversas aplicações necessárias.

Esse trabalho tem os seguintes objetivos:

- Realizar um estudo das plataformas IoT, e como é seu acesso de dados
- Propor uma arquitetura de software multi-cloud, para permitir interoperabilidade entre

plataformas de cloud

- Prover uma biblioteca open-source para a comunidade e encorajar o uso da plataforma KNoT
- Implementar e demonstrar um caso de uso da biblioteca

## 1.3 Estrutura do documento

Os capítulos que seguem estão estruturados da seguinte maneira:

- *Capítulo 2*: Apresenta contextualização deste trabalho , onde mostra a situação atual da internet das coisas, a solução para a interoperabilidade na internet das coisas, plataforma KNoT e a API da Meshblu.
- *Capítulo 3*: Mostra a proposta e as contribuições deste trabalho além de apresentar as funcionalidades e as ferramentas usadas para a implementação do mesmo.
- *Capítulo 4*: Mostra um caso de uso para biblioteca desenvolvida e faz um comparativo entre seu uso.
- *Capítulo 5*: Apresenta a conclusão, incluindo dificuldades encontradas e trabalhos futuros.

# Capítulo 2

# Contextualização

Neste capítulo serão abordados os assuntos necessários para o entendimento do presente trabalho. São considerados assuntos relacionados à Internet das coisas. Devido a granularidade do assunto e o foco do trabalho, Internet das coisas não será abordado em grande profundidade. Em vez disso, será falado sobre internet das coisas e suas plataformas.

Este capítulo está organizado do seguinte modo: A seção 2.1 apresenta uma visão geral sobre Internet das coisas. A seção 2.2 descreve o conceito de plataformas de IoT, já a seção 2.5, fala sobre a meta-plataforma KNoT e na seção 2.3 fala sobre a Meshblu e na seção 2.4 fala sobra a plataforma FIWARE.

## 2.1 Internet das coisas (IoT)

A definição de internet das coisas vem circulando por um tempo e existem diferentes entendimentos do que esse termo significa. Não há uma definição unificada, dependendo de qual pessoa ou organização estiver trabalhando[7]. No geral há 3 perspectivas[4] que podem focar:

- Na internet,
- nos dispositivos conectados ou
- nos dados

A definição mais unificada é a de que ela é *uma rede de objetos inteligentes que tem a capacidade de se auto organizar, compartilhar informação, dados e recursos, reagindo e atuando diante de situações e mudanças no ambiente*[9].

Espera-se que 75% dos carros produzidos até 2020 saiam prontos para conexão, cidades devem investir 133 bilhões em IoT até 2019, na agricultura quase 75 milhões de dispositivos devem estar conectados até 2020[10].

Os ambientes de internet das coisas são caracterizados por alto grau de heterogeneidade por conta da especificidade de cada solução que pode variar desde de comunicação, taxa de transmissão, consumo de energia e custo [3]. Tratar essa heterogeneidade é trabalho de uma plataforma de IoT.

## 2.2 Plataformas de IoT

Nada mais é do que um middleware e a infraestrutura que possibilita que o usuário final interaja com objetos inteligentes [8]. Uma plataforma de IoT tem como principal objetivo conectar e unificar inúmeros objetos e sistemas heterogêneos [11].

Essa heterogeneidade gera um grande desafio para a criação e definição de padrões de comunicação para as coisas na internet, similar ao que já existe hoje na World Wide Web(WWW). Ainda é difícil falar em padrões dentro desse universo, evidenciando a importância das plataformas, que nada mais são do que uma infraestrutura que visa facilitar a interação dos dispositivos (hardware, as coisas) com as aplicações (software). Uma plataforma pode ser de software – que é um conjunto de serviços responsáveis pelo tráfego e armazenamento dos dados – ou de hardware–microcontroladores com seu conjunto de sensores, atuadores e rádios de comunicação [10].

No trabalho de Mineraud. et. al. [8], é feita uma analise de 39 plataformas de IoT e o que falta para elas resolverem o cenário atual de IoT. As plataformas são avaliadas nos seguintes quesitos: arquitetura, suporte a dispositivos heterogêneos, controle de acesso aos dados, serviços de descoberta entre outros.

Dentre as plataformas mostradas por Mineraud. et. al. [8] podemos ver uma classificação de plataformas de IoT como:

- descentralizada, como por exemplo OpenIoT e LinkSmart
- centralizada, como exemplo temos IFTTT, EveryAware e EvryThng
- cloud-based, como Carriot, ThingWorx e Xively

Há também outras formas de classificar plataformas de IoT como diz R. Quinnel [12], de acordo com o poder de processamento e a capacidade de comunicação, uma plataforma de IoT pode se comportar entre um dessas três formas:

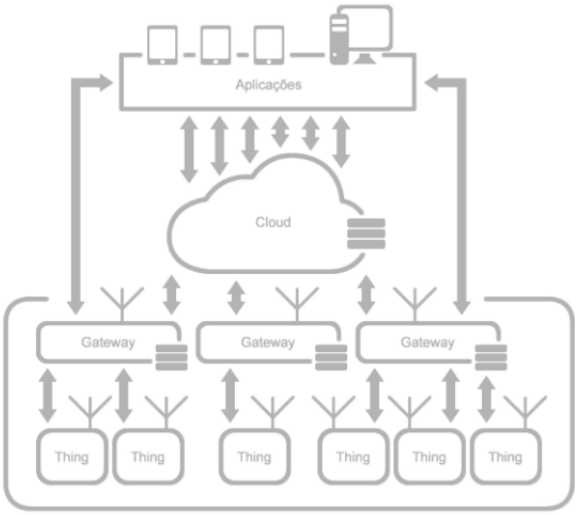


Figura 2.1: Arquitetura Hub and Spoke [1]

- *Hub and Spoke*: Essa arquitetura considera dispositivos com baixo poder de processamento e sem IP. Logo, esses dispositivos não tem capacidade de tomar decisões ou rodar algoritmos complexos. Geralmente eles são dispositivos de baixo custo com microcontroladores limitados. Como por exemplo lâmpadas inteligentes. Os dispositivos usam um rádio barato para se conectar com um gateway que irá encaminhar seus dados para a cloud. Mostrada na figura 2.1.

- *Front Loaded*: Essa arquitetura considera dispositivos com baixo poder de processamento mas com capacidade de conexão o suficiente para rodar um pilha IP. Por essa razão eles não precisam de um gateway para tradução de protocolo nem conectividade com a internet. Nesse cenário os dispositivos podem se comunicar diretamente com as aplicações ou serviços de cloud. Mostrada na figura 2.2.

- *Smart Objects*: Essa arquitetura considera que os dispositivos tem alto poder de processamento e também capacidade de conexão para se ter uma pilha IP. Por essa razão eles podem rodar algoritmos complexos como inteligência artificial por exemplo, se comunicar com outros dispositivos e tomar decisões autonomamente. Eles também podem coordenar outros dispositivos. Por isso embarcar um microcontrolador poderoso é economicamente viável desde que o custo possa ser diluído com o produto. Mostrada na figura 2.3.

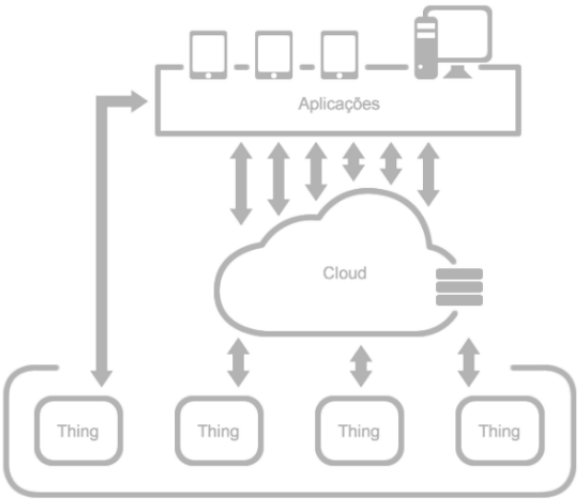


Figura 2.2: Arquitetura Front Loaded [1]

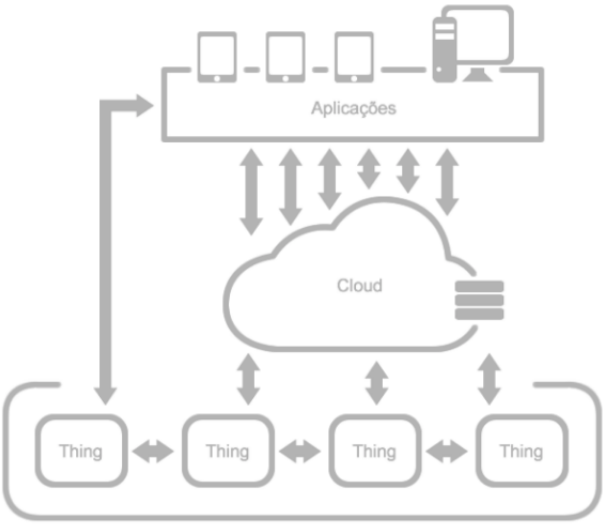


Figura 2.3: Arquitetura Smart Objects [1]

## 2.3 Meshblu API

Meshblu é uma camada de comunicação segura, escalável e multi protocolo habilitando comunicação entre dispositivos inteligentes, sensores e recursos de clouds. A Meshblu permite comunicação entre redes protocolos e outras plataformas de IoT [13].

Cada dispositivo conectado só precisa saber como enviar e receber mensagens Meshblu. Eles não precisam implementar uma API especial para todo o consumidor de informação.

A meshblu oferece os seguintes protocolos HTTP, Socket.io, Websocket, MQTT, CoAP, AMQP, and XMPP. A forma como um dispositivo conectado é identificado é pelo seu UUID e Token que são criados no momento do seu registro. A Meshblu fornece acesso seguro aos

Tabela 2.1: Meshblu

| Requisição | Caminho | Uso |
|---|---|---|
| POST | /devices | Registra um device na meshblu |
| GET | /devices?key=value& key=value | Retorna um array de devices seguindo o critério da query |
| GET | /devices/{uuid} | Retorna todas a informações do device específico |
| PUT | /devices/{uuid} | Atualiza o device |
| DELETE | /devices/{uuid} | Remove o device |
| GET | /mydevices | Retorna todas as informações de todos os devices que pertencem a um device com propriedade 'owner' |
| GET | /subscribe | É uma API de streaming que retorna mensagens de devicesassim que são enviadas ou recebidas |
| GET | /data/{uuid} | Retorna dados que o device está armazenando |
| POST | /data/{uuid} | Publica um dado para o device específico |

dispositivos registrados se eles estiverem num lista de permissão (whitelist). Essa lista conterá os UUIDs dos dispositivos que terão acesso garantido de se comunicar de maneira segura. Para acessar o dados e dispositivos a Meshblu oferece uma API REST para o desenvolvedor como é vista na tabela 2.1.

## 2.4 FIWARE

FIWARE é uma plataforma aberta e extensível focada primeiramente em suportar o desenvolvimento de aplicações em *smart cities*. Ela provê um conjunto de especificações disponíveis através da API aberta e é estruturada através de componentes de software, chamados de GEs, *generic enables*. Esses GEs seguem o modelo de especificação NGSI para padronizar troca de informações e permitir interoperabilidade entre os componentes.No modelo NGSI, a informação é estruturada de uma forma genérica através de entidades de contexto, *context entities*, que podem ser usadas para virtualmente representar elementos do mundo físico como lugares, pessoas, objetos e etc. No FIWARE, o contexto de dispositivos (devices) é provido pelo *Orion Context broker* (o simplesmente Orion), que é o GE responsavel pelo gerenciamento de contexto, como mostrado na figura 2.4. Um vez que os dados estejam no Orion, uma plicação pode realizar consultas sobre os estados ou usar outros GEs.

O FIWARE provê alguns GEs para o desenvolvimento de aplicações IoT. Os componentes

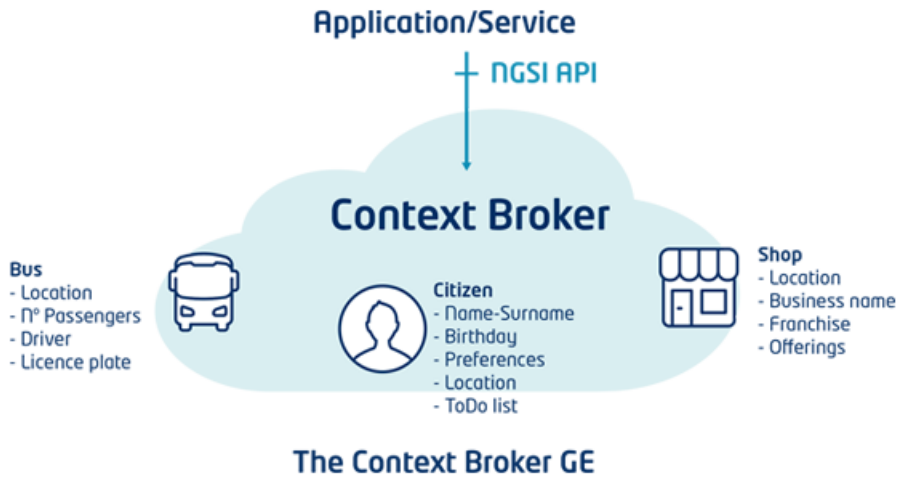


Figura 2.4: Orion Context Broker [2]

de IoT oferecem um conjunto de features e abstrações para facilitar o consumo e compartilhamento de dados dentro da plataforma. O componente de gerenciamento de devices suporta diversos tipos de protocolos de IoT através de *IoT agents* ilustrado na figura 2.5, cada um sendo responsável por tornar os devices compatíveis com o protocolo NGSI. Esses módulos são responsáveis por encaminhar e converter redes baseadas no protocolo IP para o *context broker* Orion [2].

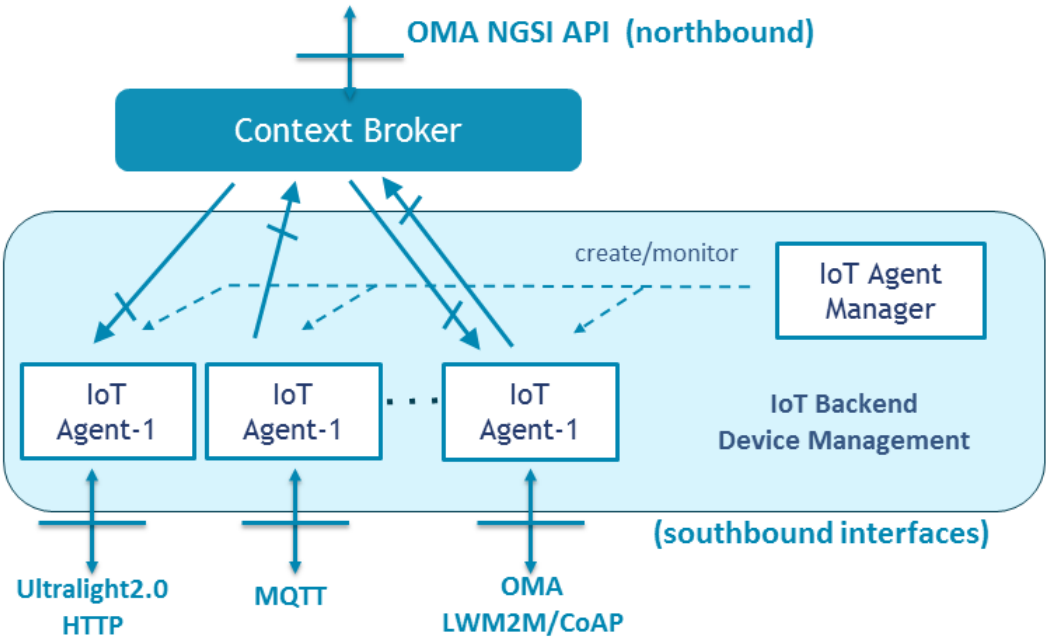


Figura 2.5: IoTAgents se comunicando com context broker [2]

## 2.5 KNoT - Network of Things

O KNoT (Network of Things) é um meta-plataforma de IoT porque ela foi construída usando outras plataformas open source para endereçar interoperabilidade. O principal objetivo é facilitar a integração, possibilitando que uma grande número de plataformas de hardware e software de IoT possa ser integrada, permitindo assim que elas se comuniquem umas com as outras plataformas [3].

Para alcançar esse objetivo o KNoT, atualmente, trabalha na arquitetura 1.0 (Hub and Spoke) comentada no capítulo 2.2, onde os dispositivos conectados não tem conexão direta com a internet mas se conectam a ela através de um gateway como mostrado na figura 2.6. Nessa arquitetura os dispositivos conectados são compostos por um microcontrolador, uma fonte de energia e um módulo RF onde o desenvolvedor pode conectar seus sensores e atuadores e criar um dispositivo nomeado de KNoT Thing.

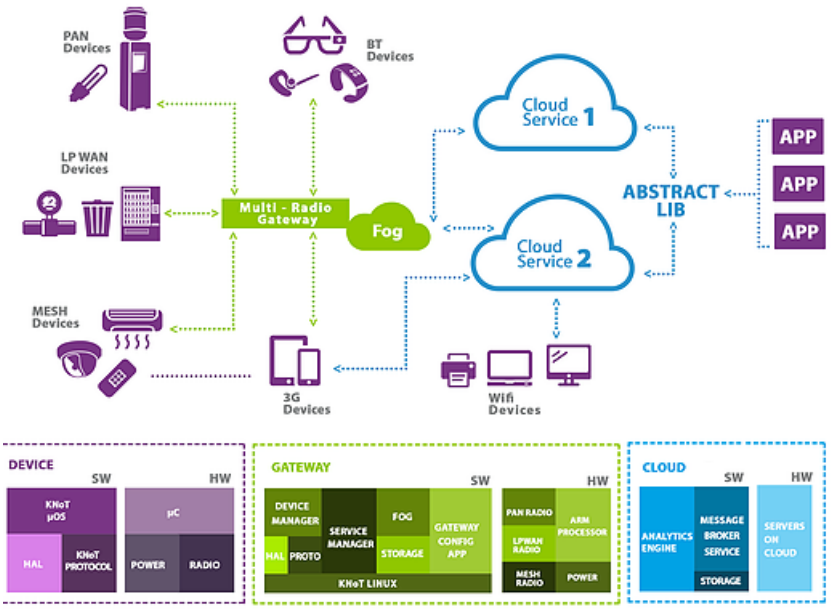


Figura 2.6: Arquitetura da meta-plataforma KNoT [3]

A figura 2.6 mostra os três atores do KNoT: dispositivo conectado (geralmente micro-controlado), gateway (dispositivos com maior poder de processamento e com maior conectividade) e cloud(servidor para processamento e armazenamento de grande volume de dados). Os sensores , presentes em cada dispositivos, leem e geram dados que são enviados para o gateway via rádio transceptor encapsulados em um protocolo específico. O gateway armazena

esses dados em um banco local (fog) que é sincronizado com a cloud. Uma vez disponibilizados em nuvem, os dados podem ser acessados por aplicações através da internet. As aplicações também podem demandar ações para os dispositivos que possuem atuadores. Essas ações são dados que trafegam no caminho inverso: da cloud até o dispositivo [10].

A versão utilizada do KNoT nesse trabalho é a 01.03, ilustrada na figure 2.7, tendo como os principais componentes:

- *KNoT Thing*: Representa o dispositivo e é composto pelo hardware (microcontrolador e rádio) e um pilha de software que gerencia os sensores e atuadores e as mensagens enviadas e recebidas entre o gateway. Nesta versão há apenas o suporte para placa Arduino com suporte ao rádio nrf24l01+.

- *KNoT Gateway*: Age como um proxy conectando múltiplos dispositivos com diferentes protocolos de rádio para a internet, traduz mensagens do dispositivo para um protocolo comum e envia o dado para a cloud. Ele também tem uma pequena instância da cloud chamada fog, para processamento e armazenamento local.Nesta versão há apenas o suporte para placa *Raspberry Pi*.

- *KNoT Cloud*: É um servidor que provê uma API para troca de dados/mensagens entre dispositivos e apps. Ele também armazena dados do dispositivo para realizar operações temporais nele. Nessa versão é um fork da plataforma Meshblu [13].

- *KNoT Libs*: Bibliotecas para Android, iOS e JavaScript que abstraem a cloud e ajudam o desenvolvimento da aplicação.

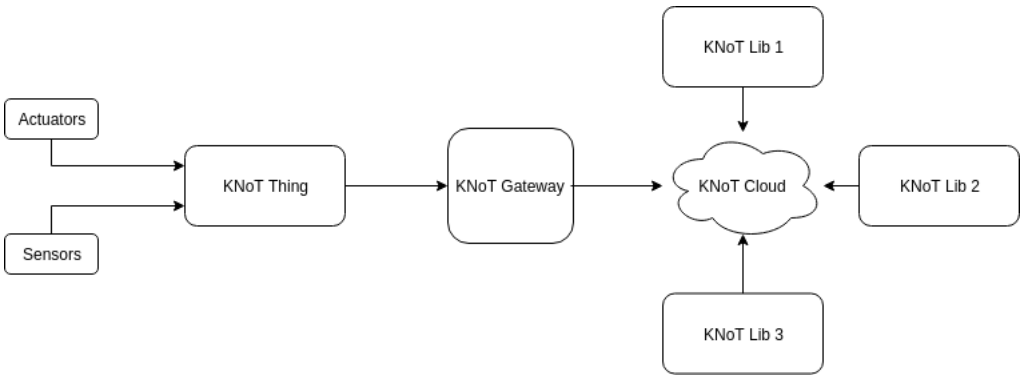


Figura 2.7: Arquitetura KNoT 01.03

Como o KNoT é uma plataforma em desenvolvimento e como o KNoT tem suporte a apenas duas plataformas clouds-based, Meshblu[13] e FIWARE, o desenvolvedor ainda precisar usar

as bibliotecas ou *frameworks* fornecidos por essa plataformas de cloud. Assim o desenvolvedor poderá fazer requisições a plataforma para ter acesso aos dados e criar aplicações para o usuário. Porém existem conjuntos de funções que o KNoT pode modelar de diferentes maneiras dependendo de cada plataforma de cloud. Por exemplo, na cloud Meshblu a operação de enviar um dado para um dispositivo é fazendo uma atualização da representação JSON do dispositivo, enquanto essa mesma operação no dispositivo da FIWARE é implementada como a criação de um comando, que não existe na Meshblu, esse comando pode ser feito a partir de uma requisição do dispositivo e assim os dispositivos podem fazer a chamada daquele determinado comando.

Essa mudança de modelagem de plataforma para plataforma é usada para aproveitar as particularidades de cada uma, fazendo com que a mesma seja aproveitada com suas particularidades. Fica a cargo então de cada biblioteca fazer a implementação correta daquela requisição na forma como foi modelada.

# Capítulo 3

# KNoTPy

Este capítulo apresenta um visão geral de implementação da biblioteca KNoTPy. Primeiramente há uma sessão discutindo as proposta da biblioteca enquanto em outra mostras as tecnologias usadas no processo de implementação e para a abordagem técnica da solução. Todo o código está disponível no GitHub como uma biblioteca *open source*.

## 3.1 Proposta

Este trabalho propõe uma biblioteca chamada KNoTPy. O principal objetivo é permitir não só acessar dados dos dispositivos conectados a plataforma KNoT mas também interagir com sensores e atuadores que tem nesses dispositivos de forma simples como também fazer uso de ferramentas de *data analytics* com a linguagem Python.

Essa biblioteca foi escrita em Python pois é uma da linguagens mais utilizadas para *data analysis* [14] [15]. A arquitetura de software é modelada em alto nível na figura 3.1. As entidades tracejadas na figura são APIs e protocolos que não foram implementados e que constam em trabalhos futuros que será comentado no capitulo de conclusão.

O desenvolvedor da aplicação irá fazer as chamadas das funções onde irá passar pela camada de abstração do KNoT que usa elementos definidos no KNoT de como "organizar"um dispositivo. Após isso a biblioteca identificará qual a cloud o usuário está usando junto com KNoT e irá selecionar qual a sua determinada API, cada API tem associada a ela um protocolo de comunicação especifico que aquela cloud implementa. Quando este trabalho foi feito o KNoT oferecia suporte para duas clouds, Meshblu e FIWARE.

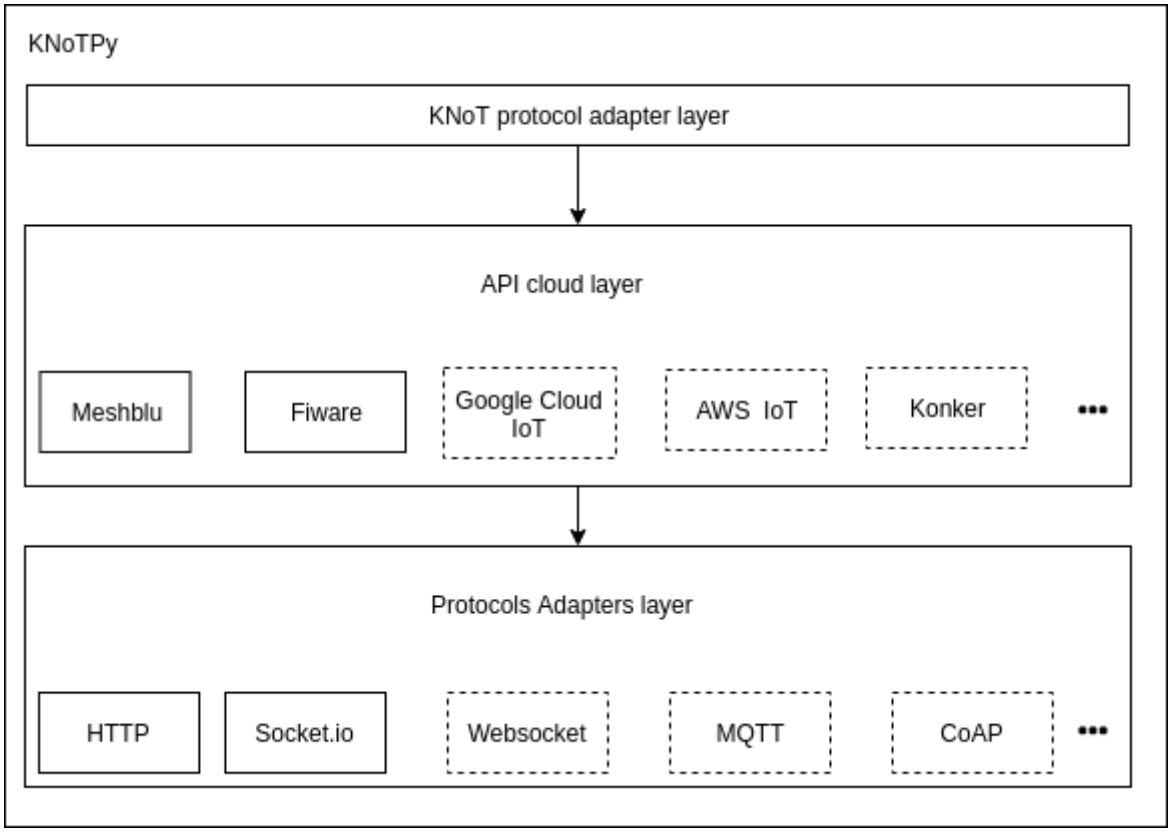


Figura 3.1: Arquitetura de alto nível KNoTPy

## 3.2 Implementação

Este capítulo apresenta um visão geral de implementação da biblioteca KNoTPy. Primeiramente há uma sessão discutindo as tecnologias usadas no processo de implementação e outra para abordagem técnica da solução. Todo o código está disponível no GitHub como uma biblioteca *open source*.

### 3.2.1 Tecnologias

Todo o código da biblioteca KNoTPy foi escrito para ser compatível com as versões de Python 2.x e Python 3.x, pois apesar de Python 3 vir como substituto do Python 2 muitas bibliotecas ainda fazem uso de Python 2. Internamente a biblioteca foi usada com as dependências de *socket_io_client* e *requests*. A IDE usada para implementação foi o Visual Studio Code, desenvolvida como projeto open source pela Microsoft, que tem vários plugins que permitem um melhor desempenho quando se programa em Python. E também durante a fase de implementação foi usado o GitHub para o versionamento do código e para hospedar os arquivos fontes.

### 3.2.2 Abordagem Técnica

A biblioteca KNoTPy consiste de duas principais classes: KnotConnection e as classes que herdam de Cloud, no caso a Meshblu e a FIWARE. Há também outras classes que herdam da classe Protocol que são: ProtoSocketIo e ProtoHttp. Que representam os protocolos suportados pela plataforma KNoT, socketio e HTTP. A figura 3.2 mostra o diagrama UML das classes.

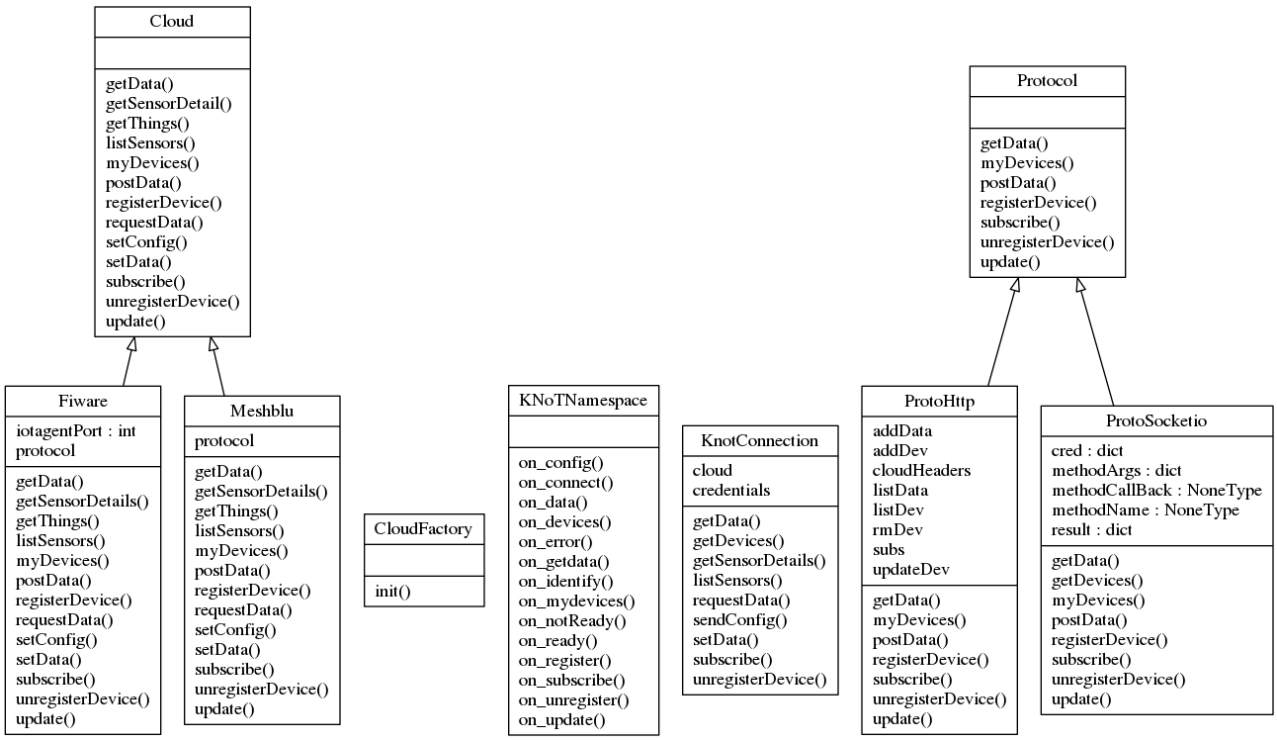


Figura 3.2: Diagrama de classes UML (gerado pelo pacote pyreverse)

#### KnotConnection

Essa é a classe que o usuário pode interagir para se comunicar com a cloud a partir do protocolo passado. Ela é responsável por criar uma conexão dependendo da operação que o usuário quer fazer com seu dispositivo. Ela também irá usar as funções definidas pelo ErrorHandler para dizer se é preciso gerar uma Exception de python. Dentre as operações possíveis estão setar o sensor de um dispositivo, ou pegar os dados desse determinado sensor. Essa classe segue o padrão de projeto Adapter. Para começar a utilizar suas funções basta indicar qual protocolo deseja utilizar e as credenciais do seu usuário e sua cloud.

### Abstração de clouds

A ideia de abstração de clouds é permitir que essa biblioteca seja multicloud e para quando o KNoT implementar outro suporte a uma plataforma de cloud seja de fácil modificação. A Cloudfactory é uma classe genérica que segue o padrão de projeto Abstract Factory. As classe Meshblu e FIWARE fazem o papel de implementar o métodos específicos de cada cloud. Nesse trabalho a parte de autenticação e de segurança não foram habilitados devido ao suporte da plataforma *KNoT* ainda estar em desenvolvimento.

### Abstração de protocolos

Para uma plataforma de IoT como há uma grande heterogeneidade de dispositivos no mercado é preciso habilitar uma gama de protocolos sabendo das limitação que esses dispositivos terão. Nesse trabalho foram implementados duas classes abstraindo os protocolos para que as classes de cloud possam usar, ProtoHttp e ProtoSocketio. No caso da ProtoHttp é necessário passar campos com os headers e endpoints na criação classe ProtoHttp. Já na ProtoSocketio é preciso passar o namespace do socket. Além disso um serviço de gerenciamentos de erros vindos desses protocolos foi implemtado como uma classe que gera exceções em python para serem tratadas pelo desenvolvedor da aplicação.

# Capítulo 4

# Casos de uso

Este capítulo descreve dois casos de uso da biblioteca KNoTPy. O primeiro, mais simples, é uma aplicação onde iremos acender uma lâmpada conectada ao KNoT Thing, um arduino. O outro é um pouco mais complicado pois é uma aplicação que pode consultar todos os dispositivos pertencentes a um usuário e configurar os sensores desses devices e alterar os parâmetros de envio de dados dos sensores como o tempo, limite superior ou inferior ou por mudança. Ao final do capítulo será feita uma discussão comparando como seriam feitas esses casos de uso usando a API de JavaScript da Meshblu e da FIWARE e será mostrado a simplicidade para o desenvolvedor da aplicação sem precisar saber todo o funcionamento da API da cloud do KNoT.

## 4.1 Ligando uma lâmpada remotamente

O problema nesse caso de uso é simples: conseguir controlar uma lâmpada através de uma aplicação simples. A lâmpada está ligada a um relé que é controlado pelo microcontrolador. O Arduíno está com firmware do protocolo KNoT para enviar e receber comandos da cloud.

No código mostrado na figura 4.1, vemos uma comparação de como seria feito o caso de uso usando a API da Meshblu (a esquerda) e (a direita) podemos ver como é usando a biblioteca desenvolvida nesse trabalho. Na API da Meshblu, como ela só opera usando o UUID como o identificador único dos dispositivos conectados é preciso fazer uma requisição GET a partir do KNoT ID, que é um ID conhecido por todos os processos da plataforma KNoT. Após termos o UUID do dispositivo conectado que queremos operar podemos enviar o comando KNoT de enviar um dado, nesse caso como é uma lampada enviamos um valor booleano para

true indicando que queremos acender a lâmpada e o sensor ao qual queremos enviar o dado. Note que para fazer isso o desenvolvedor, precisaria saber como o KNoT modelou o comando de enviar o dado para cloud da Meshblu, que nesse caso é fazendo uma requisição para se atualizar o dispositivo, através de um PUT. Porém ao ver o código a direta, as operações com UUID são abstraídas internamente pela biblioteca logo ela só opera usando o KNoT ID. A figura 4.2 faz agora a comparação de como é uma requisição para a API do FIWARE (a esquerda) com a biblioteca desenvolvida no presente trabalho, mostrando que apenas é preciso colocar o parâmetro CLOUD para fazer a biblioteca mudar o contexto de como é feita a requisição e as configurações necessárias como a porta do servidor e as credenciais.

sem_knot.py

```python
from requests import request
from json import dumps, loads

THING_ID = '7e133545550e496a'
uuid = "4c591297-b2b8-42a9-ad1d-eabc86f10000"
token = "add6f4dfb9846502dc68596170107f6e851982c7"
servername = 'knot-test.cesar.org.br'
port = 3000
url = 'http://%s:%d/devices/%s?id=%s' %(servername, port, uuid,THING_ID)

headers = {
    'Content-Type': "application/json",
    'meshblu_auth_uuid': uuid,
    'meshblu_auth_token': token,
}

response = request("GET", url, headers=headers)
thing = loads(response.text)

url = 'http://%s:%d/devices/%s' %(servername, port, thing.uuid)

payload = {
    'set_data': [{
        'sensor_id': 1,
        'value': True
    }]
}

response = request("PUT", url, data=dumps(payload), headers=headers)

print(response.text)
```

com_knot.py

```python
from knotpy import KnotConnection

THING_ID = '7e133545550e496a'
conn = KnotConnection({
    'uuid': '4c591297-b2b8-42a9-ad1d-eabc86f10000',
    'token': 'add6f4dfb9846502dc68596170107f6e851982c7',
    'servername': 'knot-test.cesar.org.br',
    'port': 3000
})
try:
    conn.setData(THING_ID, 1, True)
except Exception as err:
    print('Something went wrong with the following error')
    raise err
```

Figura 4.1: Comparação entre requisição MESHBLU e KNoTPy, para acender uma lâmpada remotamente

## 4.2 Configuração de sensores dos dispositivos conectados

O problema nesse caso de uso é: configurar o dispositivo para que todos os sensores dos dispositivos enviem um dado a cada mudança do sensor ou a cada hora. A biblioteca do KNoT Thing possui uma funcionalidade que é a de configuração remota pois podemos alterar as configurações padrões contanto que o dispositivo esteja conectado. O dispositivo irá receber a request do KNoT Gateway que veio da Cloud atualizando a representação do device virtual.

sem_knotfy.py

```python
from requests import request
from json import dumps, loads

THING_ID = '7e133545550e496a'
servername = 'knot-test.cesar.org.br'
port = 1026
url = 'http://%s:%d/v1/updateContext' %(servername, port)

headers = {
    'fiware-service': "knot",
    'fiware-servicepath': "/device/%s" %THING_ID,
}

payload = {
    "contextElements": [{
        "type": "sensor",
        "isPattern": "false",
        "id": 1,
        "attributes": [{
            "name": "setData",
            "type": "command",
            "value": "true"
        }]
    }],
    "updateAction": "UPDATE"
}

response = request("POST", url, data=dumps(payload), headers=headers)

if loads(response.text).get('contextResponses'):
    print(response.text)
else:
    print('Something went wrong with the following error')

```

com_knotfy_2.py

```python
from knotpy import KnotConnection

THING_ID = '7e133545550e496a'
conn = KnotConnection({
    'servername': 'knot-test.cesar.org.br',
    'port': 1026
}, cloud= 'FIWARE')
try:
    conn.setData(THING_ID, 1, True)
except Exception as err:
    print('Something went wrong with the following error')
    raise err
```

Figura 4.2: Comparação entre requisição FIWARE e KNoTPy, para acender uma lâmpada remotamente

A principal ideia é usar a API por exemplo caso se deseje monitorar o uso de uma lâmpada num recinto, admitindo-se uma lâmpada como o sensor que apenas envia um sinal booleano e um dispositivo como sendo responsável por dar conectividade através de rádio para o gateway, no caso do KNoT esse rádio é o nrf24l01+ [16]. O código ilustrado na figura 4.3, implementa esse caso de uso fazendo a mesma comparação da seção anterior. Como é mostrado nas figuras 4.2 e 4.4 apenas é necessário modificar um parâmetro para fazer a mudança de contexto da cloud.

A primeira linha é para carregar as classes do biblioteca KNoTPy, e a segunda para importar o arquivo de configuração do usuário. A linha seguinte é gerada uma instância da classe KnotConnection para podermos utilizar os conjuntos de funções para se comunicar com a cloud.

A função getDevices é utilizada para obter as credenciais dos dispositivos pertencentes ao usuário. Após isso basta iterar sobre os dispositivos e usar a função sendConfig que irá enviar a requisição a cloud, os parâmetros usados nessa função são o UUID do dispositivo, o id do sensor (valores de 1 à 255) o valor 252 já conhecido como sendo o da lâmpada, o outro parâmetro é a event flag que segue o padrão da tabela 3, como descrito na tabela o parâmetro time_sec é setado para enviar dados dos sensores de 20 em 20 segundos. A FLAG_TIME é usada em conjunto com a flag FLAG_CHANGE possibilitando assim mesclar eventos na configuração do

sem_knot_2.py

```python
from requests import request
from json import dumps, loads

THING_ID = '7e133545550e496a'
uuid = "4c591297-b2b8-42a9-ad1d-eabc86f10000"
token = "add6f4dfb9846502dc68596170107f6e851982c7"
servername = 'knot-test.cesar.org.br'
port = 3000
url = 'http://%s:%d/devices/%s?owner=%s' %(servername, port, uuid, uuid)

headers = {
    'Content-Type': "application/json",
    'meshblu_auth_uuid': uuid,
    'meshblu_auth_token': token,
}

response = request("GET", url, headers=headers)
mythings = loads(response.text)

for thing in mythings:
    sensors = thing.get('schema')
    for sensor in sensors:
        url = 'http://%s:%d/devices/%s' %(servername, port, thing.uuid)

        payload = {
            'config': [{
                'sensor_id': 1,
                'event_flags': 9,
                'time_sec': 3600,
            }]
        }

        response = request("PUT", url, data=dumps(payload), headers=headers)

        print(response.text)

```

com_knot_2.py

```python
from knotpy import KnotConnection, FLAG_CHANGE, FLAG_TIME
conn = KnotConnection({
    'uuid': '4c591297-b2b8-42a9-ad1d-eabc86f10000',
    'token': 'add6f4dfb9846502dc68596170107f6e851982c7',
    'servername': 'knot-test.cesar.org.br',
    'port': 3000
})
try:
    mythings = conn.getDevices()

    for thing in mythings:
        sensor_ids = conn.listSensors((thing['id'])
        sensors = [conn.getSensorDetails((thing['id'], i) for i in sensor_ids]
        for sensor in sensors:
            if sensor.get('name') == 'Temperature':
                conn.sendConfig(thing['id'], sensor.get('sensor_id'),
                FLAG_CHANGE+FLAG_TIME, timeSec=3600)
except Exception as err:
    raise err
```

Figura 4.3: Comparação entre requisição MESHBLU e KNoTPy, para configurar dispositivos remotamente

sem_knotfy_2.py

```python
from requests import request
from json import dumps, loads

servername = 'knot-test.cesar.org.br'
port = 1026
url = 'http://%s:%d/v1/queryContext' %(servername, port)

headers = {
    'fiware-service': "knot",
    'fiware-servicepath': "/device/%s" %THING_ID,
}
payload = {
    'entities': [{
        'type': 'device',
        'isPattern': 'true',
        'id': '/*'
    }]
}

response = request("POST", url, headers=headers)
mythings = loads(response.text).get('contextResponses')
for thing in mythings:
    url = 'http://%s:%d/v1/updateContext' %(servername, port, thing.uuid)
    payload = {
        "contextElements": [{
            "type": "sensor",
            "isPattern": "false",
            "id": 1,
            "attributes": {
                "sensor_id": 1,
                "event_flags": 9,
                "time_sec": 3600
            }
        }],
        "updateAction": "UPDATE"
    }
    response = request("PUT", url, data=dumps(payload), headers=headers)
    print(response.text)

```

com_knotfy.py

```python
from knotpy import KnotConnection, FLAG_CHANGE, FLAG_TIME
conn = KnotConnection({
    'servername': 'knot-test.cesar.org.br',
    'port': 1026
}, cloud='FIWARE')
try:
    mythings = conn.getDevices()

    for thing in mythings:
        sensor_ids = conn.listSensors((thing['id'])
        sensors = [conn.getSensorDetails((thing['id'], i) for i in sensor_ids]
        for sensor in sensors:
            if sensor.get('name') == 'Temperature':
                conn.sendConfig(thing['id'], sensor.get('sensor_id'),
                FLAG_CHANGE+FLAG_TIME, timeSec=3600)
except Exception as err:
    raise err
```

Figura 4.4: Comparação entre requisição FIWARE e KNoTPy, para configurar dispositivos remotamente

dispositivo.

Quando a requisição chegar a cloud ela irá usar a conexão socketIO com o gateway para atualizar o dispositivo na fog. Lá o gateway irá traduzir o dispositivo atualizado e enviar as novas configurações para o dispositivo.

## 4.3 Limitações

Uma das limitações é que a biblioteca admite que a plataforma cloud-based já está sendo suportada pelo KNoT. A outra é que KNoT ainda precisa de gateway, uma raspberry, para poder funcionar corretamente. Ou seja antes de se poder usar a biblioteca é preciso antes fazer o deploy de todos os componentes da infraestrutura do KNoT. Outra limitação é de que se alguma plataforma de cloud oferece um serviço que apresenta alguma funcionalidade a mais para o desenvolvedor, como por exemplo, um serviço para processamento estatístico, ou machine learning a biblioteca desenvolvida nesse trabalho ainda não poderá acessar esses serviços.

## 4.4 Discussão

Neste capítulo dois casos de uso da biblioteca KNoTPy foram mostrados. O mais simples liga uma lâmpada remotamente. Mesmo sendo uma aplicação simples a biblioteca torna ela mas clara sem precisar tratar as callbacks de credencias inválidas ou dispositivos inexistentes, tornando assim a implementação poluída. Depois, um segundo caso de uso foi apresentado para fazer uma configuração remota dos dispositivos que possuem um sensor de temperatura. Esse caso de uso foi mais complicado do que o primeiro pois é preciso ter todos os dispositivos do owner e depois filtra-los a partir do nome do sensor ser 'Temperature' e então enviar a configuração a partir da função sendConfig. Como previamente mostrado, a solução usando apenas a API da cloud não permite que seja passado as event flags para o microcontrolodar, o desenvolvedor deveria saber previamente os valores dessas flags que são armazenados no sistemas embarcado, além disso, o desenvolvedor precisa ter o conhecimento de como a cloud organiza os sensores no formato JSON para poder acessa-los. Na biblioteca KNoTPy ela unifica a estruturação dos sensores independente de cloud e também define as macros para o desenvolvedor saber que configuração deve ser enviada.

Além da contribuição para o conjunto de bibliotecas do KNoT, essa biblioteca foi utilizada por alunos da disciplina de Engenharia de Software e Sistemas do Centro de Informática da UFPE 2018.2, onde puderam criar aplicações para o projeto final usado na avaliação da disciplina. Ao fazer entrevistas com os alunos da disciplina, os alunos afirmam que uma das vantagens ao utilizar a biblioteca é fornecer uma melhor legibilidade para o desenvolvedor e que por ser Python ela está muito aberta para adaptações para cada tipo de problema mas que possíveis desvantagens podem surgir com aplicações de tempo real.

# Capítulo 5

# Conclusão

Este trabalho teve como objetivo o projeto e a implementação de uma biblioteca em python, que preenche o gap descrito pelo trabalho de Mineraud. et. al. [8], permitindo não só acessar dados dos dispositivos conectados mas também interagir com sensores e atuadores que tem nesses dispositivos, usando a meta-plataforma de IoT, KNoT. Neste trabalho, foram discutidos os principais conceitos sobre o atual cenário de Internet das Coisas, que incluem plataformas de IoT e ferramentas como o KNoT para permitir a interoperabilidade entre dispositivos e plataformas de cloud. Assuntos relacionados a plataforma KNoT também foram abordados, focando em sua funcionalidade, que colaborou com a biblioteca desenvolvida.

Portanto, a biblioteca foi desenvolvida com todas as funcionalidades suportadas pelas clouds da plataforma KNoT seguindo a API da Meshblu e da FIWARE discutidas. As aplicações criadas com essa biblioteca foram testadas através de exemplos como leitura de sensores, interação com atuadores e geração de dados.

## 5.1 Dificuldades encontradas

A maior dificuldade deste trabalho foi encontrar uma biblioteca em python que permitisse uma interação com a API de socketIO da Meshblu, de forma que pudesse lidar com o retorno das funções da API. Ao encontrar a biblioteca *socketio_client* que fornece uma API para gerenciar namespaces e callbacks de retorno foi possível uma implementação flexível para a biblioteca desenvolvida. Além disso a criação de uma arquitetura que permita multiplas clouds serem adicionadas a biblioteca desenvolvida neste trabalho.

## 5.2 Trabalhos Futuros

A biblioteca desenvolvida, KNoTPy irá crescer à medida que o KNoT for crescendo, pois o KNoT tem o objetivo de unir todas as plataformas de IoT. Então os seguintes tópicos devem ser trabalhados futuramente:

- *Habilitar múltiplas clouds*: Quando este trabalho foi desenvolvido, só há duas plataformas de cloud suportadas, a Meshblu e FIWARE. O KNoT vem trabalhando para habilitar outros serviços de cloud como Konker e AWS IoT. A arquitetura de abstração de cloud usada nesse trabalho permite a implementação de outras API de cloud que o KNoT suportará.

- *Permitir que a biblioteca tenha automação de testes*:A biblioteca poderá usar testes de integração para permitir que quando novas funcionalidades forem sendo adicionadas ao KNoT, elas possam ser automatizadas como por exemplo fazer todas as operações básicas do KNoT.

- *Integrar funcionalidades de análise de dados*: Quando este trabalho foi proposto não havia nenhuma forma de fazer análise de dados ou uma ferramenta para gerar gráficos que possa se extrair informações dos dados do KNoT. Como python é uma das linguagens mais usadas para esse feito a biblioteca desenvolvida pode ser usada para esse feito.

- *Implementar serviço de descoberta*: Como o KNoT é a partir de um gateway afim de salvar dados temporais e locais. Um mecanismo de descoberta de gateways deve ser implemtado para fazer um requisição na fog ao invés da cloud. Esse é um dos principios da Edge Computing (computação de borda). Que ainda fornece um serviço mesmo sem a disponibilidade de internet.

- *Implementar autenticação e autorização para a cloud FIWARE*: Não foi possivel implementar as parte de autenticação e autorização fornecidas pelos GEs do FIWARE, Wilma e AuthZForce. Eles permitem que apenas o usuário possa acessar os dados do device ao quais ele pertence.

# Referências Bibliográficas


[1] "Imagens removidas do knot user guide." `http://knot-devel.cesar.org.br/resources/knot_users_guide.pdf`, journal=KNoT. Acessado em 03/11/2018.

[2] "Fiware getting started." `https://fiware-tutorials.readthedocs.io/en/latest/getting-started/index.html`.

[3] "Knot website." `https://www.knot.cesar.org.br/`.

[4] Q. Zhu, R. Wang, Q. Chen, Y. Liu, and W. Qin, "Iot gateway: Bridgingwireless sensor networks into internet of things," in *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*, pp. 347–352, Ieee, 2010.

[5] P. Middleton, P. Kjeldsen, and J. Tully, "Forecast: The internet of things, worldwide, 2013," *Gartner Research*, 2013.

[6] A. Shah, H. Acharya, and A. Pal, "Cells in the internet of things," *arXiv preprint arXiv:1510.07861*, 2015.

[7] M. Lenders, "Analysis and comparison of embedded network stacks," *Analysis*, no. 2/34, 2016.

[8] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, "A gap analysis of internet-of-things platforms," *Computer Communications*, vol. 89, pp. 5–16, 2016.

[9] S. Madakam, R. Ramaswamy, and S. Tripathi, "Internet of things (iot): A literature review," *Journal of Computer and Communications*, vol. 3, no. 05, p. 164, 2015.

[10] E. Dornelas and S. Campello, "Monitoramento de consumo doméstico de água utilizando uma meta-plataforma de iot," *Revista de Engenharia e Pesquisa Aplicada*, vol. 2, no. 2, 2017.

[11] E. C. G. Silva, M. I. S. Oliveira, E. Oliveira, K. S. da Gama, and B. F. Lóscio, "Um survey sobre plataformas de mediaç ao de dados para internet das coisas," *researchgate*, 2015.

[12] R. Quinnell, "3 architectures dominate the internet of things," Nov 2014. EDN, Network.

[13] "Repositório do GitHub da Meshblu." `https://github.com/octoblu/meshblu/tree/legacy-meshblu`. Acessado em 22/08/2018.

[14] P. F. Dubois, "Guest editor's introduction: Python–batteries included," *Computing in Science & Engineering*, vol. 9, no. 3, 2007.

[15] K. J. Millman and M. Aivazis, "Python for scientists and engineers," *Computing in Science & Engineering*, vol. 11, no. 2, 2011.

[16] "nrf24l01+ specification." `https://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01`.

[17] "Repositório do GitHub da biblioteca KNoTPy." `https://github.com/ramonhpr/knot-lib-python`. Acessado em 22/08/2018.