CENTRO PAULA SOUZA
GOVERNO DE SÃO PAULO
FACULDADE DE TECNOLOGIA DE SÃO CAETANO DO SUL

FABIO ROGÉRIO FARIA BARBOSA

LUCAS LIMA GIACOMI

RAFAEL MARTINEZ SOUZA

RUANN OLIVEIRA MELGAÇO

# Implementação de um Sistema de *Blockchain* para Garantir a Integridade de Informações de Contexto Distribuídas em Brokers Federados

SÃO CAETANO DO SUL

2020

**FABIO ROGÉRIO FARIA BARBOSA**

**LUCAS LIMA GIACOMI**

**RAFAEL MARTINEZ SOUZA**

**RUANN OLIVEIRA MELGAÇO**

# Implementação de um Sistema de *Blockchain* para Garantir a Integridade de Informações de Contexto Distribuídas em Brokers Federados

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia de São Caetano do Sul, sob a orientação do Prof. Me. Fábio Henrique Cabrini, como requisito para a obtenção do diploma de Graduação no Curso de Segurança da Informação.

SÃO CAETANO DO SUL

2020

**FICHA DE AVALIAÇÃO**

FABIO ROGÉRIO FARIA BARBOSA

LUCAS LIMA GIACOMI

RAFAEL MARTINEZ SOUZA

RUANN OLIVEIRA MELGAÇO

Data de Aprovação ___/___/___

Nota: _____

Prof°. (a) Dr. (a) _____

_____

(Assinatura)

Prof°. (a) Dr. (a) _____

_____

(Assinatura)

Prof°. (a) Dr. (a) _____

_____

(Assinatura)

**AGRADECIMENTOS**

A Faculdade de tecnologia Fatec Antônio Russo e todo seu corpo docente e funcionários, por todo o carinho que nos deram durante todo o curso.

Em especial ao Prof° Ms Fábio Henrique Cabrini, por toda atenção dada nesses dois semestres e também pela confiança depositada em nosso trabalho.

Aos nossos colegas de curso que seguiram juntos do primeiro ao último dia de aula, para juntos formamos uma família dentro da Fatec.

**DEDICATÓRIA**

Primeiramente a Deus e em segundo lugar a nossas famílias, que sempre nos apoiaram e nunca deixaram de acreditar em nosso sucesso.

**Epígrafe**

“Que todos os nossos esforços estejam sempre focados no desafio à impossibilidade. Todas as grandes conquistas humanas vieram daquilo que parecia impossível”

(Charles Chaplin)

**RESUMO**

Este trabalho tem o objetivo de implementar uma arquitetura de segurança baseada em *Blockchain,* que possa garantir a integridade das informações no banco de dados do Helix SandBox NG. Por isso, foi criada uma instalação de *Blockchain,* sendo necessário analisar o sistema responsável pelo encaminhamento dos dados e gerenciamento do ciclo de vida da informação. Com seu desenvolvimento foi possível provar, através de uma federação de *context brokers*, como ela pode informar o responsável sobre tentativas de mudança de um dado na federação.

**PALAVRAS-CHAVE:** *Blockchain*. Helix Sandbox NG. Context brokers. Federação.

**ABSTRACT**

*This work aims to implement a blockchain-based security architecture, which can guarantee the integrity of the information in the Helix SandBox NG database. Therefore, a Blockchain installation was created, being necessary to analyze the system responsible for forwarding the data and managing the information life cycle. With its development, it was possible to prove, through a federation of context brokers, how it can inform the responsible about attempts to change a data in the federation.*

**KEYWORDS:** *Blockchain*-based. Helix Sandbox NG. *Blockchain*. Federation.

**LISTA DE FIGURAS**

**LISTA DE TABELAS**

**LISTA DE SÍMBOLOS**

API – Application Programing Interface

CSP – Cloud Service Provider

CPU – Central Processor Unit

GE – Generic Enabler

HDD - Hard Drive Disk

ID - Identification

IEEE – Institute of Electrical and Electronic Engineers

IoT – Internet of Things

IP – Internet Protocol

MvP – Minimum Value Product

P2P – Peer to Peer

PoC – Proof of Concept

RAM – Random Access Memory

SSD – Solid State DriveSSH - Secure Socket Layer

TLS – Transport Layer Security

**Sumário**

## INTRODUÇÃO

A Internet das Coisas (IoT) têm crescido de maneira expressiva nos últimos anos (ABNIC, 2019). Seu objetivo é realizar a interconexão digital entre dispositivos cotidianos (CONNER, 2010) ou sensores para diferentes fins, como por exemplo: monitoramento de smart bikes, patinetes, carros autônomos etc.

Segundo uma recente pesquisa (DBS, 2018), até 2030 serão cerca de 125 mil unidades de dispositivos IoT por quilômetro quadrado, destes, 75 mil são dispositivos de usuários comuns, o que representa 60% do total de dispositivos conectados à internet.

Estes dispositivos são extremamente passíveis de ataques (CIO, 2019) e demandam um sistema que possa garantir sua segurança e de seus usuários.

Uma *Blockchain* atua como um livro-razão, permitindo que as informações sejam gravadas e transmitidas de modo descentralizado (COINBASE, 2017), desta forma precisa validar as informações com base em um algoritmo de consenso.

Este algoritmo permite que todas as máquinas da rede, chamadas de “nó” entrem em um acordo sobre a veracidade de um dado adicionado à cadeia de informação.

Assim, ela consegue garantir que a informação adicionada por um nó foi a mesma repassada para os outros, ou seja, que não foi adulterada (LIN, 2017).

Segundo um projeto do MIT os semáforos, por exemplo, não serão mais necessários, tendo em vista que os carros autônomos se comunicarão por outro sistema mais inteligente, permitindo que os carros apenas desacelerem em cruzamentos, não sendo necessário parar totalmente (MIT, 2016).

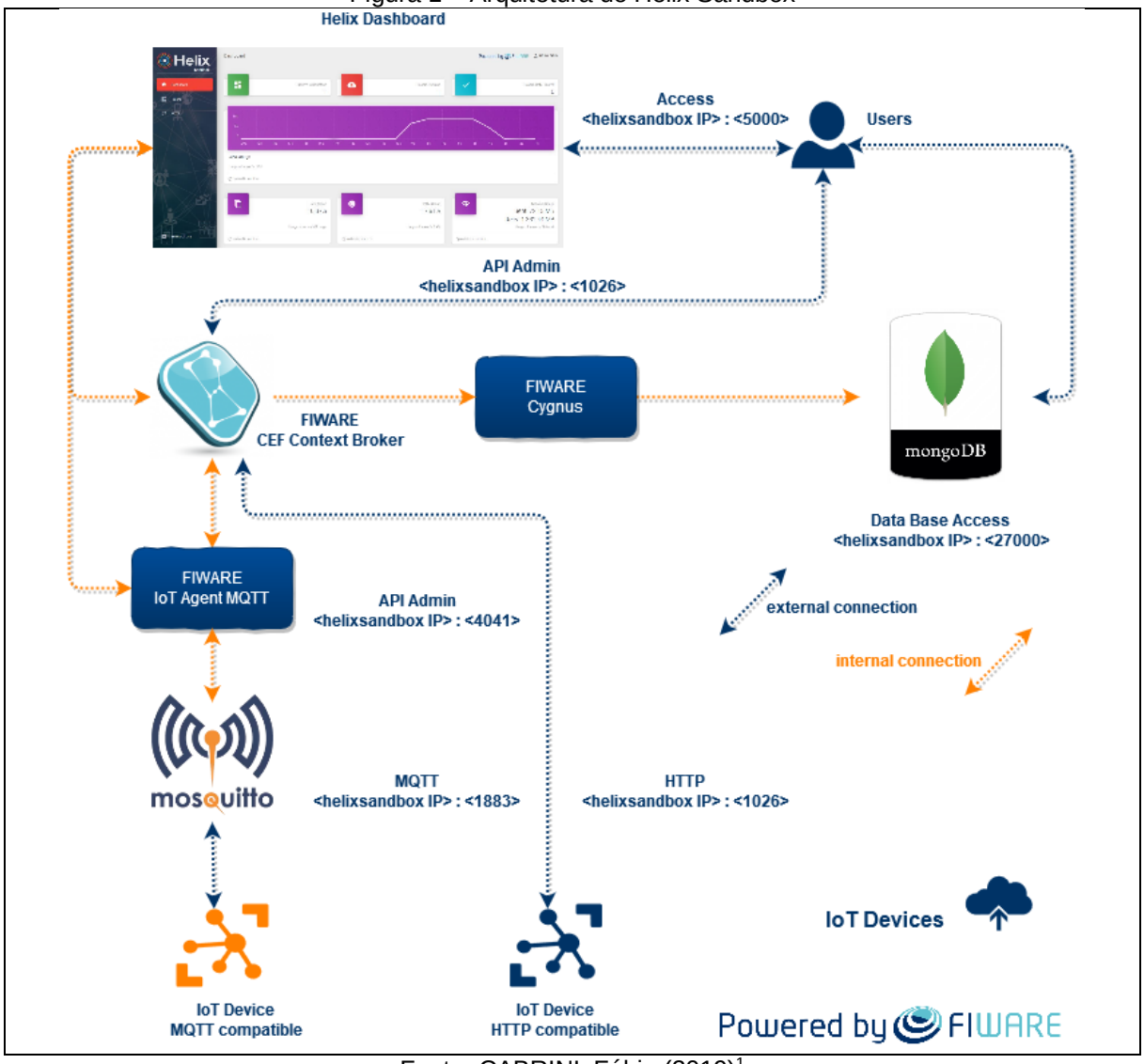
Essas tendências têm gerado a necessidade de utilização de sistemas de interconexão de dispositivos IoT, capazes de orquestrar diversos módulos para o funcionamento correto e eficiente desta grande rede de dispositivos, e o uso de um sistema que administre e forneça a segurança adequada para estes dispositivos é indispensável.

## 1 Helix SandBox

O Helix Sandbox NG (*Next Generation*) é uma plataforma de *back-end* desenvolvida para a prototipação rápida de aplicações para ambientes inteligentes e IoT, na forma de POC (*Proof of concept),* MVP *(Minimum Viable Product*) ou pesquisa acadêmica.

Ele facilita a instalação e a configuração dos principais GE (*Generic Enablers)* previstos na arquitetura FIWARE através do Helix Compose e sua arquitetura pode ser vista na Figura 1.

Figura 1 – Arquitetura do Helix Sandbox




Fonte: CABRINI, Fábio (2019)[1]

Ele apresenta uma curva de aprendizagem rápida para começar a explorar a ferramenta, tampouco um computador muito potente, ele foi pensado para ser uma

[1] Helix SandBox: An Open Platform to Fast Prototype *Smart* Environments Applications

solução bastante leve, podendo ser executado em diversos ambientes de nuvem ou local.

Nele é encontrado, por exemplo, o CEF Context Broker, um GE que gerencia o ciclo de vida das informações no Helix.

O Helix trabalha com sete camadas em sua arquitetura: *Infrastructure*; *Operating System*; *Container Platform*; *Services*; *Administration*; *Context Providers* e *Context Consumers*:

- **Infrastructure**: O Helix foi pensado para ter uma arquitetura agnóstica, trabalhando tanto nos virtualizadores mais conhecidos, quanto em plataformas *Openstack*, *Cloud Service Providers* (CSP) e em *Bare Metal*, reduzindo tempo e custo com infraestrutura e pessoal (CABRINI, Fábio, 2019);
- **Operating System**: Ele foi projetado para ser uma solução leve, precisa de recursos mínimos para seu funcionamento correto e permitindo sua utilização em qualquer distribuição Linux;
- **Container Platform**: Utiliza o Docker Engine e o Docker Compose, que permitem o gerenciamento e a comunicação de contêineres com o *Docker* Hub, responsável pela atualização dos componentes do Helix e o fornecimento e APIs para o gerenciamento e monitoramento dos Contêineres;
- **Services**: Módulos do Helix,: Helix Compose; CEF Context Broker, Cygnus IoT Agent MQTT, Eclipse-Mosquitto e MongoDB;
- **Administration**: Permite o acesso externo pela conexão SSH, pela interface gráfica do Helix Compose e pelas APIs fornecidas pelos GEs;
- **Context Providers** e **Context Consumers**: Demonstram a comunicação entre diversos dispositivos, como dispositivos IoT; agentes IoT; Dashboards; aplicativos mobile e APIs de terceiros.

Ele é uma plataforma oficial e homologada pela Fiware Foundation que certificou a plataforma como sendo Powered by FIWARE em 2018.

## 2 Federação

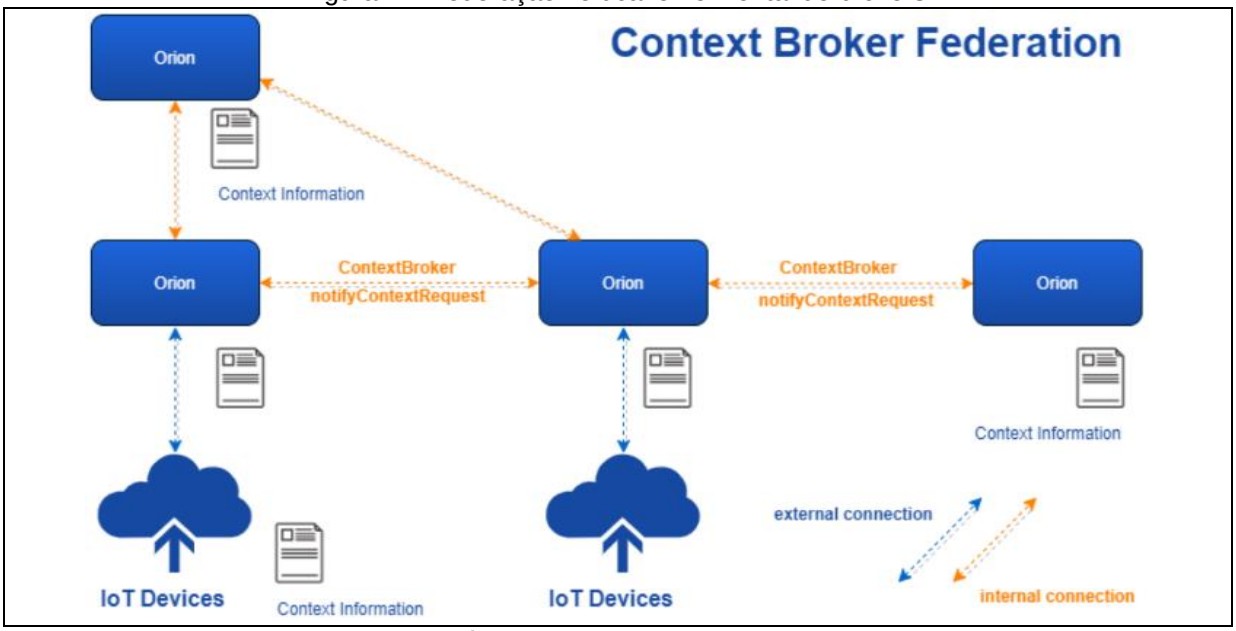
O termo pode ser usado ao descrever a interoperação de duas redes de telecomunicações distintas, formalmente desconectadas, que podem ter estruturas internas diferentes. (M. SERRANO, S. DAVY, M. JOHNSSON, W. DONNELLY, A. GALIS, 2011).

A federação é basicamente um encadeamento hierárquico de CEF Context Brokers e pode ser realizada através dos modos *push* e *pull* (FIWARE, 2019). Na federação do tipo *push*, as notificações de contexto enviadas por uma instância do CEF Context Broker são processadas e então enviadas para o próximo *broker*.

Já na federação do tipo *pull*, as notificações de contexto são repassadas para a próxima instância do Orion, codinome do CEF Context Broker, sem que a primeira instância precise manter registros do dado. O processo de federação é feito através de assinaturas utilizando-se os parâmetros adequados.

A Figura 2 apresenta o fluxo das informações dos dispositivos para a federação de *brokers* que possibilitam o trânsito de informações a todos os elementos da arquitetura.

Figura 2 – Federação vertical e horizontal de *brokers*.




**Fonte**: CABRINI, Fábio H; V. FILHO, Filippo; T. KOFUJI, Sergio.

Uma federação de *context brokers* pode ser criada "horizontalmente" no nível da nuvem ou do *fog*, ou ainda "verticalmente", interligando um ou mais nós de *fog*,

cada qual executando sua instância do CEF Context Broker, uns com os outros e com a instância central na nuvem. (CABRINI, Fabio H 2019).

Na arquitetura de múltiplas camadas prevista na arquitetura do Helix, a federação realiza o encaminhamento das informações entre *brokers* através dos métodos *push* e *pull*.

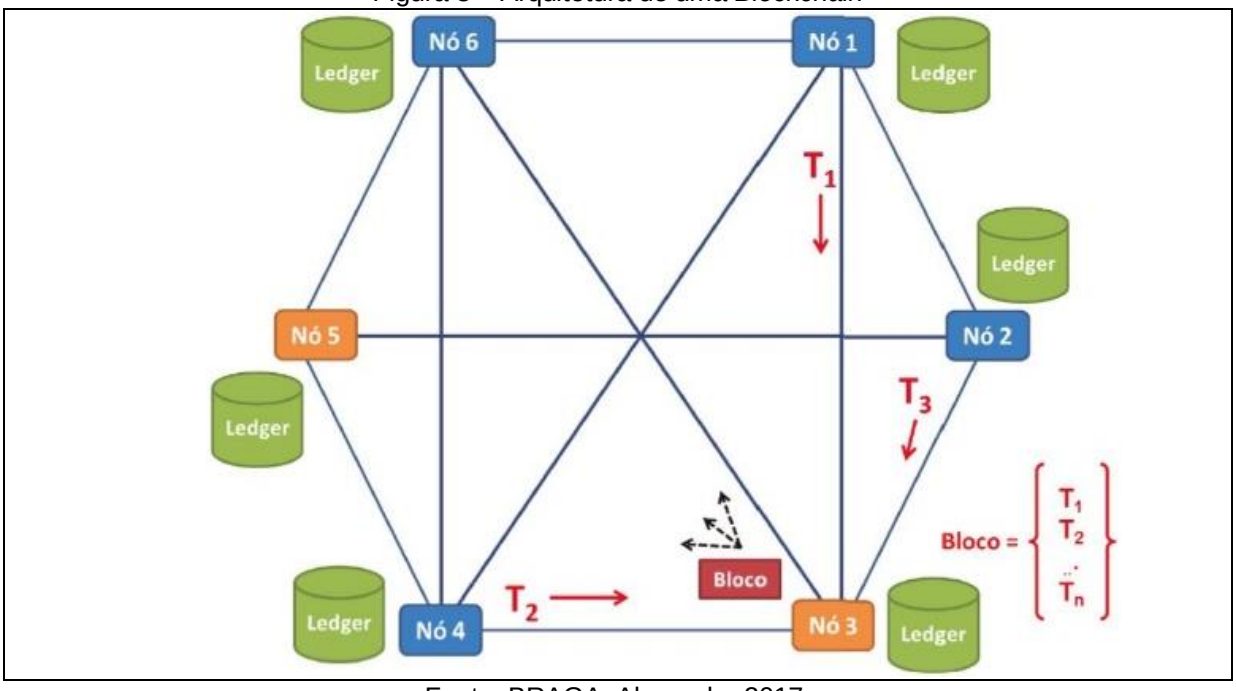
## 3. A Tecnologia *Blockchain*

Como visto anteriormente a *blockchain* é um livro-razão que garante registros confiáveis por serem imutáveis, ou seja, consiste em uma tecnologia que gera uma base de dados ordenados, formada por blocos de transações criptografadas e fortemente interligadas, que somente aceitam a inclusão de novos blocos e nunca a remoção ou modificação de blocos existentes, respeitando assim a sua cronologia.

Como solução para segurança e intervenções governamentais, a *Blockchain* trouxe um sistema descentralizado *Peer*-to-*Peer* (P2P) que preza o anonimato, garantindo assim, a integridade total dos dados.

A arquitetura de uma rede P2P é composta por *nodes*, onde todos exercem a função de cliente e/ou servidor, sem necessidade de um servidor central, conforme a Figura 3.

Figura 3 – Arquitetura de uma *Blockchain*




Fonte: BRAGA, Alexandre 2017.

Para uma melhor compreensão da estrutura de uma *Blockchain* é necessário contextualizar os *nodes* (nós) que são responsáveis por atuar como um ponto de

comunicação que apresentam as seguintes funcionalidades: criação, armazenamento e validação de dados.

A *blockchain* é mantida simultaneamente por todos os *nodes* da rede, não existindo local principal ou preferencial para armazenamento de uma base de dados originais. Todo node tem a sua réplica da base de dados, e todas são mantidas íntegras, consistentes e sincronizadas pelos protocolos de consenso.

Há diversos tipos de *nodes*, neste trabalho será colocada em evidência o *Full nodes* (nós completos), denominados também como, *Full validating nodes* (nós de validação total) que compreendem todas as funções supracitadas.

Ela também possui o *backup* de toda a cadeia de blocos, e tem como fim, realizar a verificação da integridade de todos os blocos, seguindo as normas dos algoritmos de consenso, conforme exposto na Figura 3.

### 3.1 Consenso Distribuído

Consenso distribuído é um termo da ciência da computação usado na disciplina de sistemas distribuídos e é um aspecto crítico do *Blockchain* e das criptomoedas. Consenso significa que quase todos (os envolvidos) concordam. Consenso é diferente de unanimidade, uma vez que nem todos tem que concordar, basta que a maioria concorde.

No *Blockchain*, o consenso ocorre entre os nós da rede P2P por meio de métodos compostos por protocolos específicos e regras bem definidas. Todos os nós da rede P2P são envolvidos de algum modo na tomada de decisão por consenso.

Opcionalmente, um nó centralizador (validador) pode coletar e propagar o consenso na rede P2P. O resultado de uma realização do protocolo de consenso deve ser confiável (determinístico) para toda execução.

Exemplos de métodos de consenso utilizados em *Blockchains* são os seguintes: o consenso bizantino (caracterizado pela necessidade de 3*n+1 nós na rede P2P para tolerar n divergências no consenso), a prova de trabalho utilizada no Bitcoin para mineração e a prova de participação utilizada pelo *Ethereum*.

## 3.2 Blocos e Transação

O bloco é formado por um cabeçalho contendo metadados, essa estrutura de dados que compreende as transações a serem incluídas na *blockchain* por intermédio dos *nodes* que validam os dados antes de incluírem, conforme mostra a Tabela 1.

Tabela 1 – Estrutura do Bloco.

| Campo | Tamanho | Descrição |
|---|---|---|
| Índice | 4 Bytes | Número do índice do bloco indica a posição em que ele foi criado |
| Hash Anterior | 32 Bytes | *Hash* do cabeçalho do bloco anterior a este na *blockchain* |
| Timestamp | 4 Bytes | Hora exata da criação deste bloco em segundos no padrão UNIX |
| Dados | Variável | Dado a ter sua integridade garantida |
| Hash | 32 Bytes | A função *hash* (resumo) deste bloco |
| Dificuldade | 4 Bytes | Dificuldade alvo do algoritmo de *proof-of-work* para este bloco |
| Nonce | 4 Bytes | Contator utilizado como *nonce* no algoritmo de *proof-of-work*. |

Fonte: Autoria nossa.

O bloco será salvo se for validado pelos *nodes* seguindo o algoritmo de consenso da rede, caso contrário, passam a não ter validade.

A estrutura de criptomoedas, por exemplo, possui uma estrutura que se parece com um balancete contábil de débito e crédito e é composta dos seguintes elementos:

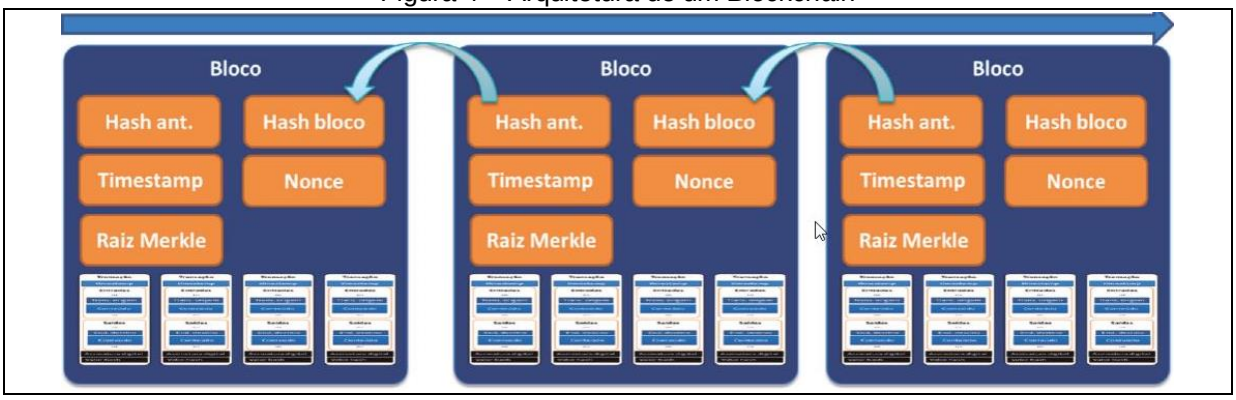
- ***Timestamp***; ***Hash***: O identificador da transação anterior de onde vemos valor de entrada (pode haver mais de um);
- **Valor de entrada**;
- **Valor de saída**;
- **Endereço de destino** (que vai receber o crédito);
- **Assinatura digital** feita com a chave privada do debitado.

## 3.3 A Cadeia de Blocos

Os *nodes* de uma rede *Blockchain* validam consensualmente a veracidade em que os dados trafegados são registrados na *ledger*. Este consenso se dá com a maioria simples (50% + 1) da rede concordando com a integridade do bloco, para que o mesmo seja adicionado à cadeia.

As transações são incluídas em blocos que estão ordenados em uma cadeia, formando uma estrutura de dados conhecida em computação como lista ligada, conforme a Figura 4.

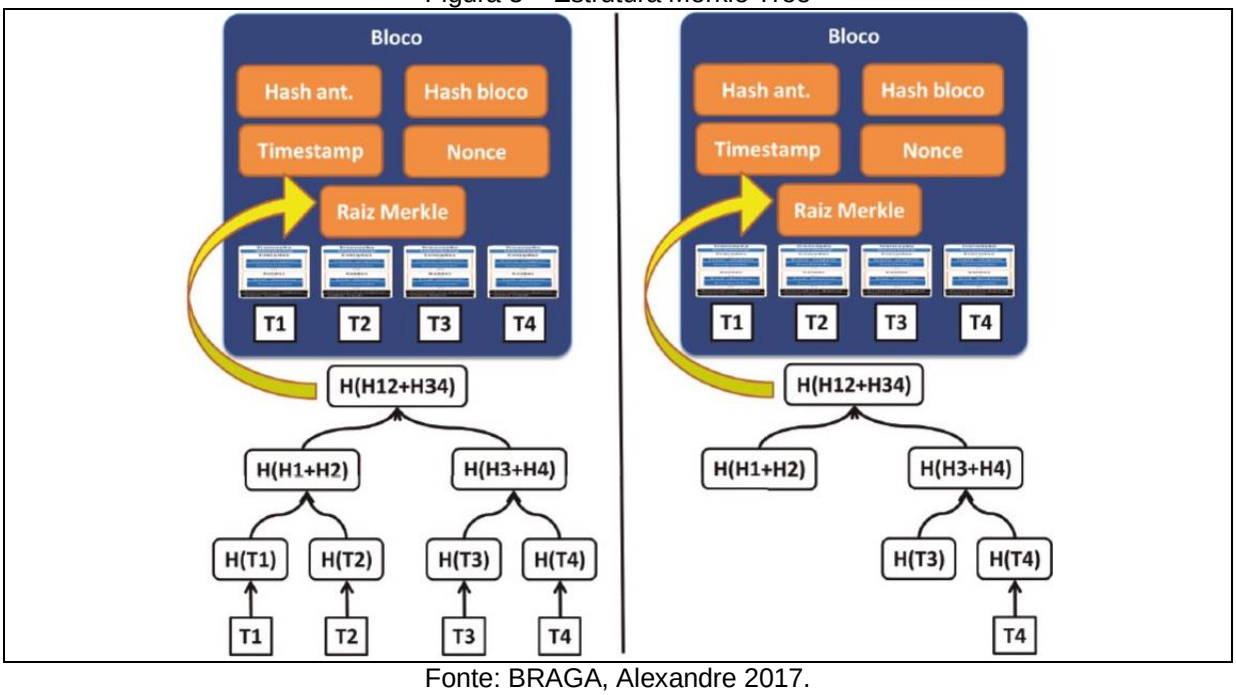
Figura 4 – Arquitetura de um *Blockchain*




Fonte: BRAGA, Alexandre 2017.

O bloco mais recente é a cabeça (*head*) da cadeia. Cada bloco contém um conjunto de transações e um cabeçalho composto dos seguintes itens: o *hash* do bloco anterior, um número pseudoaleatório único (*nonce*) e o *hash* da raiz da árvore de transações no bloco.

Figura 5 – Estrutura Merkle Tree




Fonte: BRAGA, Alexandre 2017.

Conforme a Figura 5 as transações dentro de um bloco estão ordenadas entre si de acordo em uma estrutura em árvore binária baseada em *hashes*, também conhecida como *Merkle Tree*. Ela ilustra a estrutura da árvore *Merkle* e a verificação de uma transação.

As folhas da árvore são os *hashes* das transações e os *hashes* dos pais são calculados com os *hashes* dos filhos.

Os *hashes* dos ramos imediatos, são calculados com os *hashes* das folhas, os *hashes* dos ramos intermediários são calculados com os *hashes* dos ramos imediatos, sucessivamente, até o cálculo do *hash* da raiz da árvore, que é incluído no bloco.

A estrutura em árvore acelera a operação de verificação se a transação pertence ao bloco, que pode ser feita em log(n) computações de *hash*, onde n é o tamanho da árvore.

A verificação do *hash* de uma transação só usa o ramo da árvore (*Merkle branch*) em que a transação está localizada, que é necessário para verificar o *hash* da transação. (BRAGA, Alexandre, 2017).

## 3 Detalhamento do Experimento

Esta seção apresenta os detalhes da implementação e testes realizados. O cenário utilizado será de um sensor de temperatura de um quarto, tendo em vista que a variação dos valores não é constantes, o que ajuda a demonstrar melhor o funcionamento da *blockchain*.

### 3.1 Instalação

Para a realização do experimento, foi instalado a última versão do Helix Sandbox NG, seguindo os passos de instalação disponíveis em sua página no Github.

As máquinas utilizadas precisam ter os requisitos mínimos descritos no *requirements* (1 vCPU, 1GB RAM e 16GB HDD ou SSD), tal máquina será virtualizada no Amazon Web Services.

Foi utilizada a versão do Ubuntu Server 18.04.4 Long Term Support (LTS[2]), recomendada pelos desenvolvedores.

O Postman foi *software* utilizado para a interação com o CEF Context Broker simulando os dispositivos de IoT.

[2] Versão do sistema a qual possui suporte a longo prazo.

Para o correto funcionamento do Helix, as portas abaixo devem ser liberadas no *firewall,* conforme tabela 2.

Tabela 2 – Portas de serviço do Helix

| Port | Transport | Protocol |
|---|---|---|
| 22 | TCP | SSH |
| 5000 | TCP | Helix Web Interface |
| 3030 | TCP | Helix Orchestrator |
| 22443 | TCP | Helix Hardware Monitor |
| 1026 | TCP | CEF Context Broker |
| 27000 | TCP | MongoDB |
| 5050 | TCP | Cygnus |
| 1883 | TCP | Eclipse-Mosquitto |
| 4041 | TCP | IoT Agent MQTT |

Fonte: CABRINI, Fábio (2019)

Depois de realizado o *download* execute o arquivo chamado *install.sh* localizado na pasta Sandbox-NG.

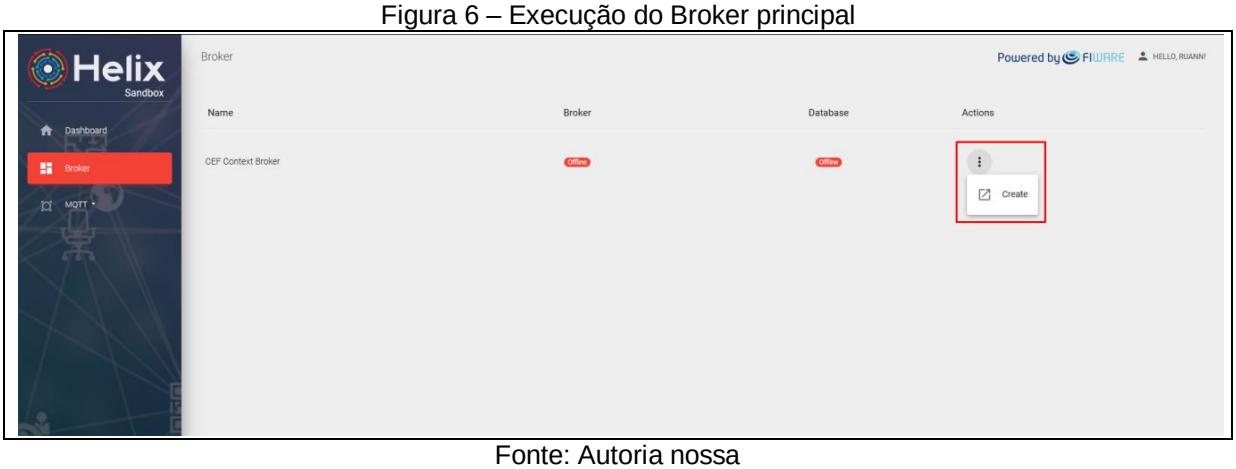
Para acessar o *dashboard* insira o endereço http:// <IP_DO_HELIX>:5000 e então realize a criação do usuário administrador. Em seguida, é necessário criar e executar o CEF Context Broker através do *dashboard*.

Todos os processos listados anteriormente estão disponíveis na página do github do Helix acessível pela página oficial da plataforma. (GETHELIX.ORG, 2019).

### 3.2 Criação do *broker*

Depois de realizado o login no *Dashboard* clique em *Broker* > *Nice, help me*! Assim poderemos realizar a criação do *Broker* clicando nos três pontinhos e em seguida em *Create*, conforme a figura 6

Figura 6 – Execução do Broker principal



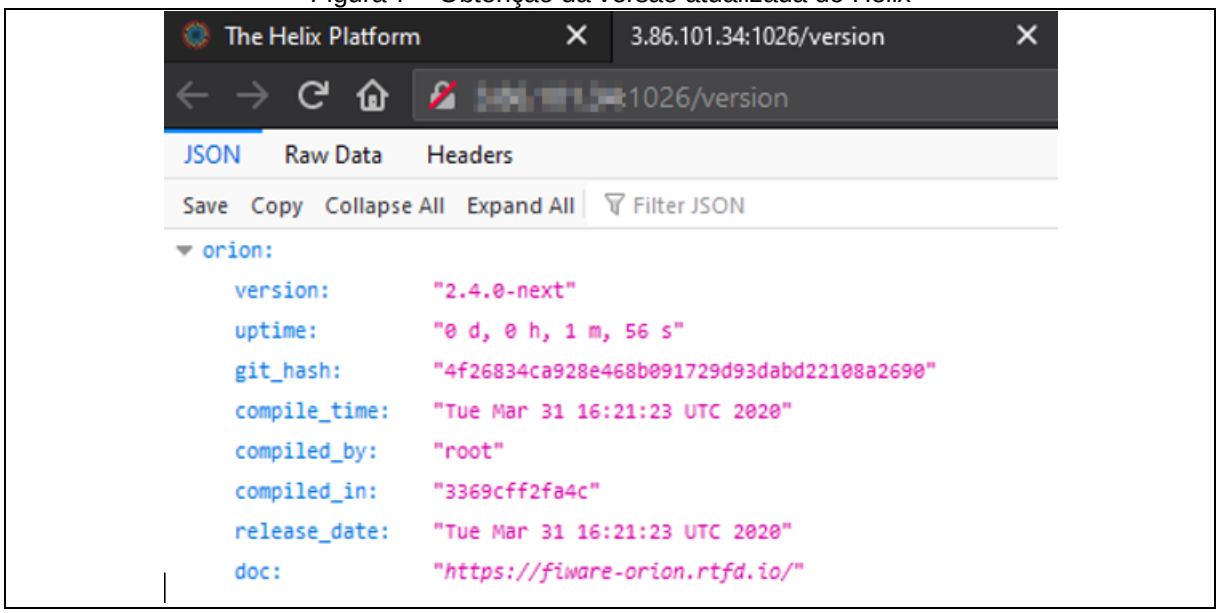

Fonte: Autoria nossa

Clique novamente nos três pontinhos e em seguida em *Run* para iniciar o *Broker*. Agora as informações poderão ser geridas pelo Helix.

### 3.3 Verificação da versão

Podemos verificar se este se encontra em sua última versão clicando novamente nos três pontinhos e em seguida em *Broker Status*, isto nos levará à uma outra página com as informações da versão instalada, conforme a figura 7.

Figura 7 – Obtenção da versão atualizada do Helix




Fonte: Autoria nossa

Neste ponto é importante notar o caminho descrito na barra de endereços (<[IP_HELIX]:1026/version>).

É através da porta 1026 que o usuário comunica os dispositivos com o *CEF Context Broker*, como anteriormente em sua arquitetura e como será analisado posteriormente.

### 3.4 Criação de uma entidade

O primeiro passo é criar a entidade de contexto, que é a representação lógica do dispositivo de IoT. O modelo de dado NGSIv2 será o mesmo utilizado pelo FIWARE.

As entidades têm um Id (*Identifier*) que as identificam no formato JSON, além de atributos, tipos de dados e metadados que a descrevem.

Para entender melhor a modelagem dos dados de uma entidade veja o Apêndice A.

A figura 8 expõe o código para a criação da entidade, está se trata de um sensor de temperatura com o valor 23.

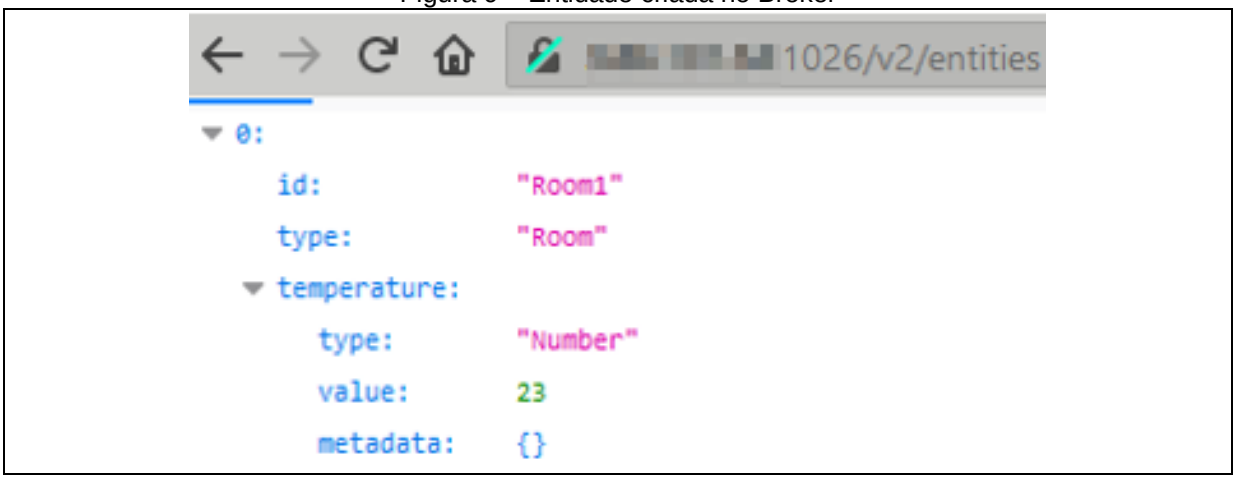
Figura 8 – Código para a criação da entidade

```
curl --location --request POST 'http://[redacted]:1026/v2/entities' \
--header 'Content-Type: application/json' \
--data-raw '{
  "id": "Room1",
  "type": "Room",
  "level": {
  "value": "23",
  "type": "integer"
 }
}'
```

Fonte: Autoria nossa

A figura 9 demonstra o resultado da criação da entidade dentro do *broker*.

Figura 9 – Entidade criada no Broker




Fonte: Autoria nossa

Os metadados desta entidade estão de acordo com seu sensor, as informações podem variar em cada tipo de sensor.

### 3.5 Federação por meio da subscrição

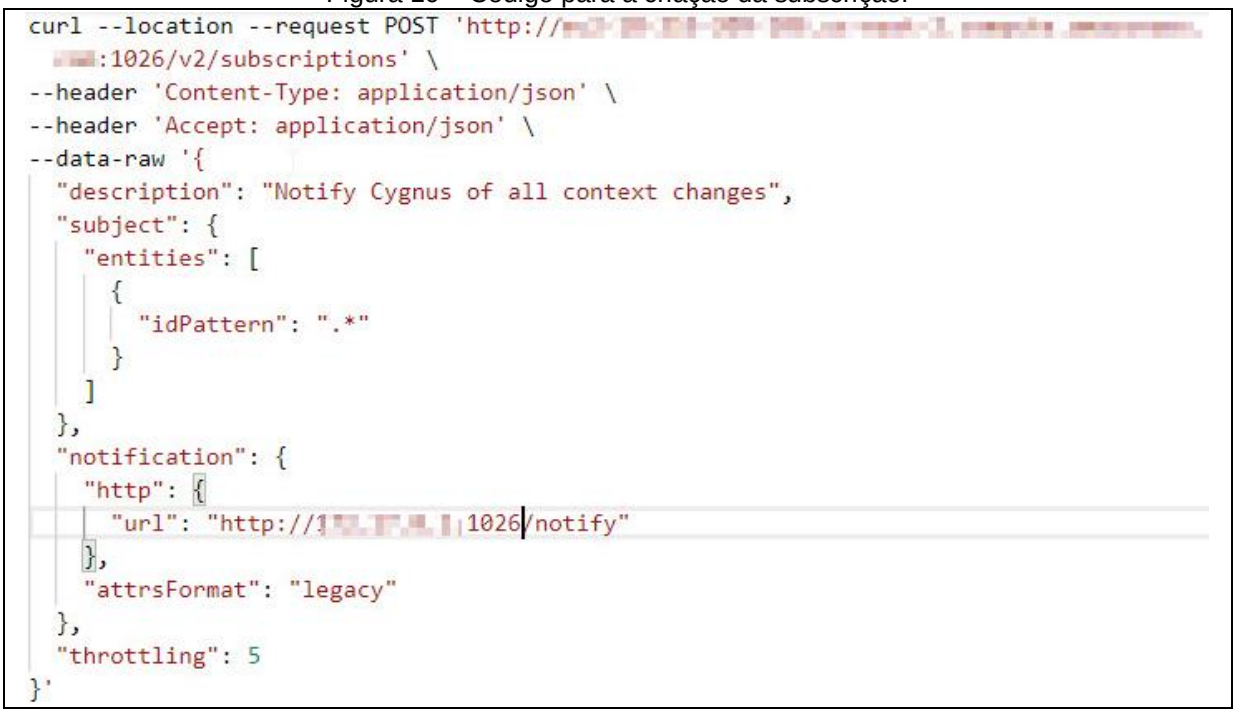
A federação no Helix é criada por meio da subscrição. Ela é criada após a entidade, assim, ela consegue verificar a entidade a qual se refere e consegue então notificar um *broker* configurado acerca de suas atualizações;

Seu código é dividido em duas partes, a primeira informa qual entidade será federada informando seu “*ID”* e “Type”, a segunda informa o *IP* do *broker* que receberá a atualização dos dados.

Conforme a figura 10, pode-se observar o código utilizado para criar a subscrição; Note que as informações na primeira parte do código são as mesmas da entidade criada anteriormente, ou seja, “*ID”* e “*Type”*.

Caso as informações da entidade estejam incorretas a subscrição ainda será criada, porém, a federação não funcionará.

Figura 10 – Código para a criação da subscrição.

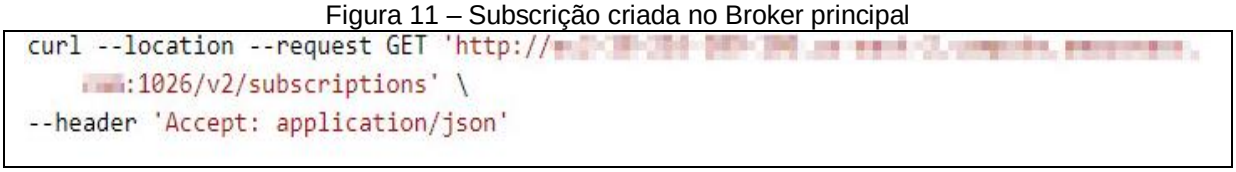


```
curl --location --request POST 'http://[redacted]
  :1026/v2/subscriptions' \
--header 'Content-Type: application/json' \
--header 'Accept: application/json' \
--data-raw '{
  "description": "Notify Cygnus of all context changes",
  "subject": {
    "entities": [
      {
        "idPattern": ".*"
      }
    ]
  },
  "notification": {
    "http": {
      "url": "http://[redacted]:1026/notify"
    },
    "attrsFormat": "legacy"
  },
  "throttling": 5
}'
```

Fonte: Autoria nossa

Conforme a figura 11, pode-se ver o resultado da criação da subscrição dentro do broker.

Figura 11 – Subscrição criada no Broker principal



```
curl --location --request GET 'http://[redacted]
    :1026/v2/subscriptions' \
--header 'Accept: application/json'
```

Fonte: Autoria nossa

Como visto nas entidades a subscrição aqui também é composta por um ID, porém no lugar do Tipo, há o s*tatus* da subscrição e os metadados.

Caso seja necessário federar o dispositivo para um terceiro *broker*, basta criar uma subscrição no segundo *broker* criado apontando para o terceiro.

### 3.6 Atualização das entidades

Após a criação da entidade, é possível atualizar os atributos através do método PUT, conforme Figura 12:

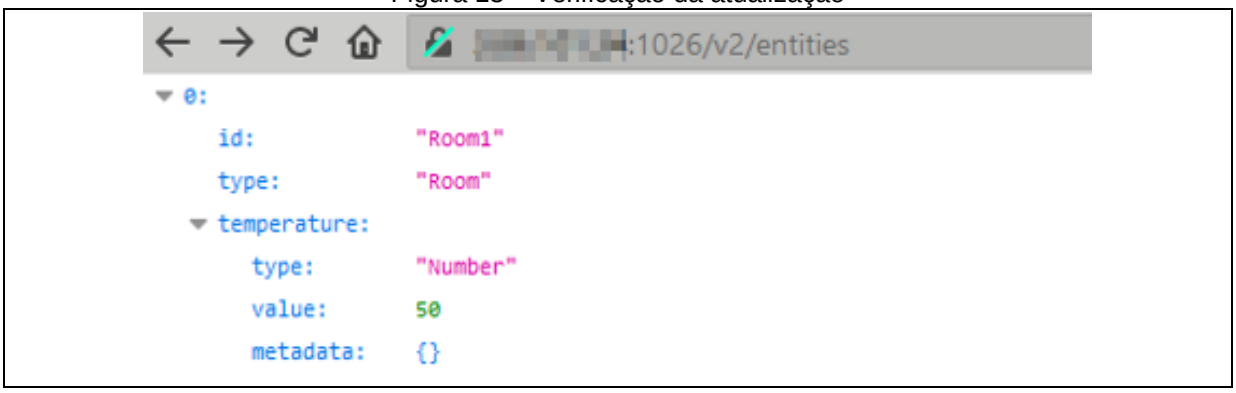
Figura 12 – Atualização do valor do sensor de temperatura da entidade

```
curl --location --request PUT 'http://[redacted]:1026/v2/entities/termometro/attrs/
    temperature/value' \
--header 'Content-Type: text/plain' \
--data-raw '1000'
```

Fonte: Autoria nossa

Esta atualização pelo Postman simula o envio real de dados pelo sensor. Conforme a figura 13 verifica-se o resultado da atualização da entidade no broker.

Figura 13 – Verificação da atualização




Fonte: Autoria nossa

Nesta seção foi simulada a atualização legítima de dados. Durante o uso da *blockchain* será simulado uma atualização ilegítima de dados, podendo assim verificar o resultado com a *blockchain* em funcionamento.

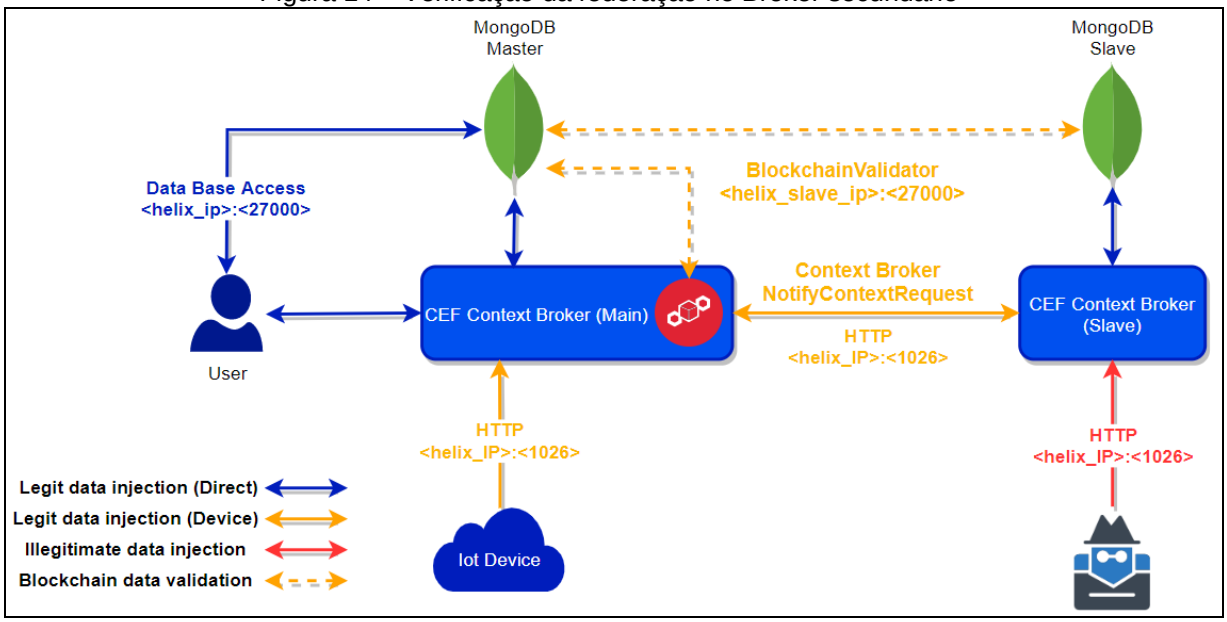
## 4 Utilizando a *Blockchain*

Para este experimento foi simulado o uso de dois brokers, sendo que no primeiro encontra-se instalada a *blockchain* para a validação dos dados. Uma atualização legítima e uma atualização ilegítima serão simuladas para que se possa verificar o resultado nos dois casos.

A *Blockchain* criada faz a validação dos dados contidos nos *brokers*. Após ser instalada no *broker* principal, representada pelo ícone vermelho de *blockchain* no diagrama da figura 13, ela verificará a primeira entidade cadastrada no *broker* e a existência da federação.

Caso a federação exista a *blockchain* tentará se conectar ao *broker* secundário através da porta 27000 e então iniciará a comparação de valores entre o *broker* principal e o secundário, conforme a figura 14.

Figura 14 – Verificação da federação no Broker secundário




Fonte: Autoria nossa

Caso a federação não exista ou a porta se encontre bloqueada, a *blockchain* continuará funcionando, porém a validação não será feita e consequentemente a mensagem de notificação no *prompt* da *blockchain* não aparecerá.

Note que cada Helix possui apenas um único MongoDB, por questões de otimização. É no MongoDB que são guardados os registros de entidades, subscrições, registros.

O armazenamento temporal dos dados é realizado com auxílio do Cygnus, caso o armazenamento temporal não seja ativado o CEF conversará diretamente com o MongoDB, conforme diagrama apresentado.

Realize o download da *blockchain* através do endereço <*https://github.com/Martinez1991/helixBlockchain.git*>. Acesse a pasta da *Blockchain* e execute o código "*./install.sh*".

Após realizado o download execute o script "Principal.py", para executar a blockchain.

## 5 Resultados obtidos

Conforme a Figura 14 têm-se a visão das primeiras informações obtidas pela blockchain.

O índice 2 representa a primeira atualização dos dados, tendo em vista que o índice 0 é o bloco gênesis e o índice 1 representa a blockchain obtendo os dados iniciais do sensor.

Figura 15 – Bloco gênesis

```
Índice: 0
Hash anterior:
Timestamp: 1589549954282
Dados: Genesis block
Hash: e3438d3c39801e2bf64073b988d12dd21efe61630b2242d2070242aba8c8eecf
Dificuldade: 2
Nonce: 0

Índice: 1
Hash anterior: e3438d3c39801e2bf64073b988d12dd21efe61630b2242d2070242aba8c8eecf
Timestamp: 1589549956138
Dados: {'_id': {'id': 'Room1', 'type': 'Room', 'servicePath': '/'}, 'attrs': {'temperature': {'value': 15.0}}}
Hash: 0003e57c0c1215330994c33d453f563d31e3814c2bd002039030a7067d08abef
Dificuldade: 2
Nonce: 199

Índice: 2
Hash anterior: 0003e57c0c1215330994c33d453f563d31e3814c2bd002039030a7067d08abef
Timestamp: 1589549957005
Dados: {'_id': {'id': 'Room1', 'type': 'Room', 'servicePath': '/'}, 'attrNames': ['temperature'], 'attrs': {'temperature
': {'value': 15.0, 'type': 'Number', 'mdNames': [], 'creDate': 1589539644, 'modDate': 1589539948}}, 'creDate': 158953964
4, 'modDate': 1589539948, 'lastCorrelator': '2b62ae4c-969a-11ea-900d-0242ac120004'}
Hash: 00a4ba579a01268b127d448e31e98023a006fa855d411689febd712c6d6f4a6d
Dificuldade: 2
Nonce: 138
```

Fonte: Autoria nossa.

Enquanto a figura 15 demonstra a atualização legítima do dado, a figura 16 demonstra a tentativa de atualização indevida do mesmo. Note que a informação não foi escrita pois foi impedida pela *blockchain*.

Figura 16 – *Blockchain* após alteração do dado

```
------------------------------------------------------------
                    Blockchain  Helix
------------------------------------------------------------



Índice: 0
Hash anterior:
Timestamp: 1589549954282
Dados: Genesis block
Hash: e3438d3c39801e2bf64073b988d12dd21efe61630b2242d2070242aba8c8eecf
Dificuldade: 2
Nonce: 0

Índice: 1
Hash anterior: e3438d3c39801e2bf64073b988d12dd21efe61630b2242d2070242aba8c8eecf
Timestamp: 1589549956138
Dados: {'_id': {'id': 'Room1', 'type': 'Room', 'servicePath': '/'}, 'attrs': {'temperature': {'value': 15.0}}}
Hash: 0003e57c0c1215330994c33d453f563d31e3814c2bd002039030a7067d08abef
Dificuldade: 2
Nonce: 199

Índice: 2
Hash anterior: 0003e57c0c1215330994c33d453f563d31e3814c2bd002039030a7067d08abef
Timestamp: 1589549957005
Dados: {'_id': {'id': 'Room1', 'type': 'Room', 'servicePath': '/'}, 'attrNames': ['temperature'], 'attrs': {'temperature
': {'value': 15.0, 'type': 'Number', 'mdNames': [], 'creDate': 1589539644, 'modDate': 1589539948}}, 'creDate': 158953964
4, 'modDate': 1589539948, 'lastCorrelator': '2b62ae4c-969a-11ea-900d-0242ac120004'}
Hash: 00a4ba579a01268b127d448e31e98023a006fa855d411689febd712c6d6f4a6d
Dificuldade: 2
Nonce: 138

['Room1']
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    Device Room1 Foi adulterado
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Fonte: Autoria nossa.

Note que ela descreve também o nome do dispositivo que foi adulterado, sendo um meio ainda rústico de notificar o usuário acerca da tentativa de adulteração.

# 6 Vulnerabilidades encontradas no ambiente Helix Sandbox NG

Nenhum sistema ou ferramenta é a prova de falhas, tanto humanas quanto lógicas por parte do sistema.

Nesse capítulo será abordado mais detalhes acerca de algumas das vulnerabilidades encontradas durante o desenvolvimento do trabalho, tendo em vista o ambiente com Helix Sandbox NG integrado com o banco de dados MongoDB assim como o cenário ideal de configuração.

## 6.1 Visualização de dados.

O *broker* funciona como uma estrutura de diretórios em árvore, desta forma é possível visualizar todas as informações dentro dele apenas acessando sua estrutura através de um navegador.

Isso pode fornecer informações valiosas para um agente malicioso, como informações do sensor de geolocalização de um dispositivo.

## 6.2 Injeção de dados através do Postman

O Helix não exige autenticação para inserção de dados, desta forma o próprio método de inserção de dados através do Postman utilizado neste experimento é uma vulnerabilidade.

O atacante não precisa possuir um conhecimento técnico muito extenso para adulterar as informações, basta que saiba o IP do helix.

## 6.3 Injeção de dados através do MongoDB

Como dito no início desse capítulo, a integração Helix Sandbox NG trabalha junto ao banco de dados MongoDB, para acessar tal banco de dados é comumente utilizado o **Mongo Compass**.

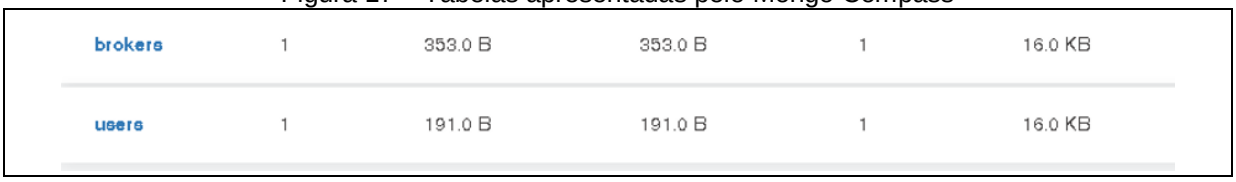
Para acessar o banco de dados, basta inserir o IP, porta e autenticar com um usuário e senha padrão, sendo o usuário “helix” e a senha “H3l1xNG”. Estes dados são de conhecimento comum entre os utilizadores do Helix.

Com ele são apresentados os dados e tabelas encaminhados para o banco de dados durante criação, atualização e deleção de dados, *brokers* e informações.

Através do MongoDB é possível realizar a alteração dos dados simplesmente selecionando o dispositivo desejado e inserindo uma informação qualquer. Para isto basta ter um conhecimento básico da estrutura dos dados no Helix.

A Figura 17 demonstra a visualização de duas tabelas, a ***main.brokers*** e a ***main.users*** que contêm os dados dos *brokers* e usuários respectivamente.

Figura 17 – Tabelas apresentadas pelo Mongo Compass




Fonte: Autoria nossa

Acessando tais tabelas, é apresentado uma estrutura de dados semelhantes às descritas na Figura 18.

A Figura 18 demonstra a estrutura de dados de um *broker* dentro da tabela *main.brokers*.

Figura 18 – Tabela *main.brokers*


Fonte: Autoria nossa

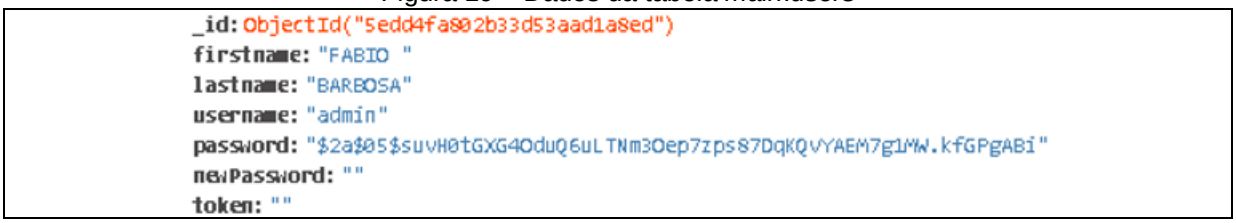
Nela, é possível observar diversos dados como ID; Nome; IP; porta usada pelo API do Cygnus; porta usada pelo container do banco de dados MongoDB e se os dados históricos dela estão habilitados.

### 6.4 Bruteforce no usuário administrador

Ainda é possível realizar um ataque *bruteforce* (força bruta) no usuário administrador do sistema, sendo possível descobrir sua senha.

Para tanto, é necessário entender como as senhas são criptografadas no Helix. Observando agora a figura 19 aonde é apresentada os dados dos usuários da plataforma Helix Sandbox NG que o banco de dados está conectado.

Figura 19 – Dados da tabela *main.users*




Fonte: Autoria nossa

Nela é possível identificar os campos de ID; Primeiro nome; Sobrenome; usuário e senha. Assim como as que foram criadas no primeiro acesso. É notável que no campo de senha está preenchido com dados que causam uma certa estranheza.

Esses dados são a senha criada pelo usuário, mas criptografada em formato de hash usando o algoritmo conhecido como ***eksblowfish***.

### 6.4.1 O Algoritmo Eksblowfish

O Eksblowfish é uma variante da cifra Blowfish, modificada para tornar a configuração da chave muito cara. ("Eks" significa “expensive key schedule” ou "programação de chave cara").

Isso não a torna significativamente mais forte, mas visa impedir ataques de força bruta. Também a torna inadequada para qualquer aplicativo que exija agilidade importante.

Foi projetado por Niels Provos e David Mazieres para hash de senha no OpenBSD. Eksblowfish é uma cifra parametrizada (com chave de família). É preciso um parâmetro de custo (Cost Parameter) que controla o quanto o agendamento de chaves é caro.

Também é preciso uma chave de família, conhecida como "sal" (salt). Parâmetros de custo e sal juntos definem uma família de cifras. Dentro de cada família, uma chave determina uma função de criptografia da maneira usual. (Main, Andrew em Crypt:Eksblowfish. Metacpan, 27 de abril de 2011).

### 6.4.2 Metodologia do ataque bruteforce

O Utilizando um simples *script* em Python se é possível extrair a *hash* da senha do usuário administrador do ambiente Helix Sandbox NG, converte-la em texto e assim conseguir o acesso à plataforma utilizando um usuário administrador.

Isto representa um risco gravíssimo à integridade da plataforma, visto que esse usuário tem o “poder” de alterar e deletar quaisquer dados, como usuários, registros e até o próprio *broker*.

No início desse *script*, deverá ocorrer uma importação do cliente MongoClient da distribuição PyMongo. A distribuição PyMongo contém ferramentas para interagir com o banco de dados MongoDB do Python.

O pacote bson é uma implementação do formato BSON para Python. O pacote pymongo é um driver Python nativo para o MongoDB. O pacote gridfs é uma implementação do gridfs sobre o pymongo. (PyPi, pymongo 3.10.1 project, 07/01/2020).

Uma importação do crypt e urllib.parse para o python. O Crypt é uma função hash unidirecional baseada em um algoritmo DES modificado; veja a página de manual do Unix para mais detalhes.

Os usos possíveis incluem o armazenamento de senhas com hash, para que você possa verificar as senhas sem armazenar a senha real ou tentar quebrar as senhas do Unix com um dicionário. (Python. crypt — Function to check Unix passwords).

Já o Urllib.parse define uma interface padrão para quebrar as strings de URL (Uniform Resource Locator) em componentes (esquema de endereçamento, local de rede, caminho etc.), para combinar os componentes novamente em uma string de URL e converter um "URL relativo" em um URL absoluto, dado um "URL base". (Python. urllib.parse — Parse URLs into components)

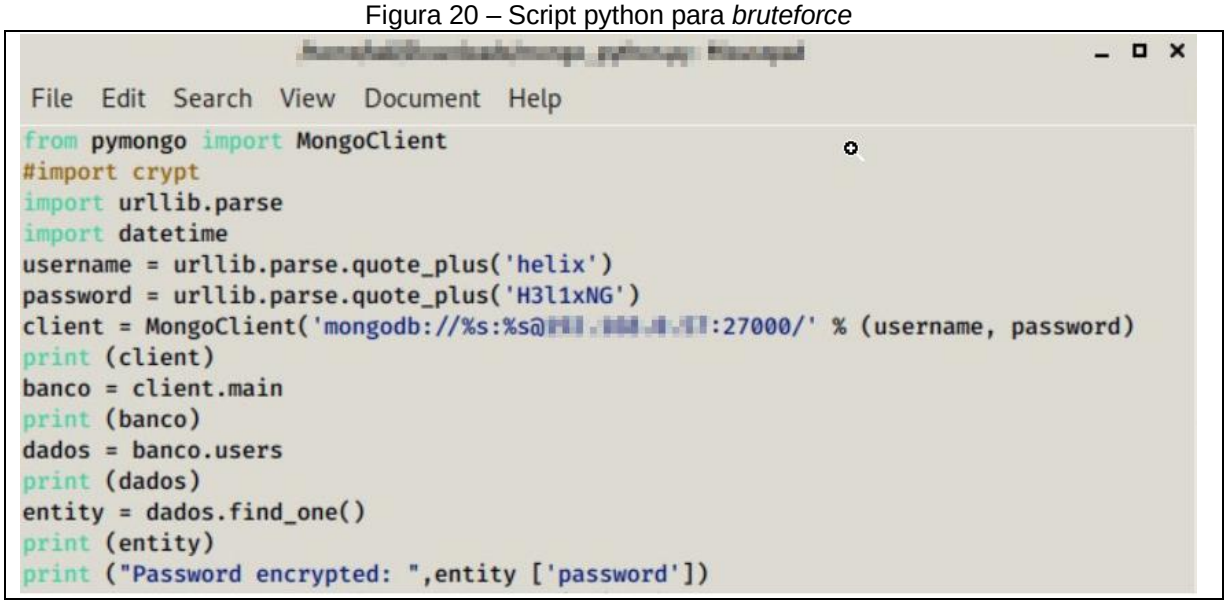
Então, após a importação, será inserido no script os comandos para acessar o bando de dados MongoDB alvo usando o nome de usuário e senha do mesmo, que como dito anteriormente, é de conhecimento comum e em um cenário aonde o

usuário não tenha alterado a senha de acesso ao MongoDB e tendo conhecimento do ip do servidor.

Com o acesso ao banco de dados concebido, o atacante teria acesso às tabelas contidas dentro do banco e assim à tabela que contêm dos dados de usuários, a main.users.

Assim, com acesso a ela, o atacante poderia realizar uma busca pela senha do usuário administrador, aonde seria apresentada o hash da senha. A Figura 20 demonstra um exemplo de como o script em python do atacante seria escrito, no final dele, seria apresentado o hash da senha do usuário alvo.

Figura 20 – Script python para *bruteforce*




Fonte: Autoria nossa

Utilizando esse dado, o atacante poderia utilizar o *crypt* para executar um ataque da metodologia *Bruteforce* utilizando uma *wordlist* afim de gerar uma hash identica à coletada e assim identificar a senha do usuário administrador.

Nesse caso, é exemplificado que o usuário alvo utilizou como senha a sequência númerica: 1234567890 e o atacante ao invés de utilizar uma wordlist, escolheu inserir esses mesmos dados para comparação.

Escrevendo então no script do python uma invocação do *crypt* e estabelecendo como comparação a senha e o salt do hash recolhido anteriormente. O salt é identificado como os 5 primeiros digitos do hash.

O *crypt* então criptografa os dois dados inserido e gera um hash, a Figura 21 exemplifica o mesmo *script* anterior mas sendo executado o *crypt* e gerando a hash.

Figura 21 – Script Python após utilizar o crypt


Fonte: Autoria Nossa

Nesse cenário o atacante tendo o hash do usuário de dentro do MongoDB que coletou anteriormente e o hash gerado agora utilizando o *crypt,* lógicamente saberia a senha de acesso do usuário e assim o acesso ao ambiente do Helix Sandbox NG.

## 7 Considerações Finais

Conforme o exemplo da entidade “Room1” e no exemplo dos semáforos inteligentes mostrados no início do documento, pequenas alterações indevidas em alvos sensíveis podem vir a gerar problemas terríveis.

A utilização de um sistema que possa mitigar o risco de quebra da integridade se mostra bastante necessário. A blockchain criada para o documento é instalada em um primeiro *broker*, e desta maneira consegue validar os demais conectados a ele. Ela consegue ainda impedir que a alteração indevida ocorra.

Apesar de ainda bastante rústica, a blockchain consegue notificar o usuário acerca das atualizações indevidas na própria tela de prompt onde ela é executada. Desta forma, buscar uma maneira de melhorar a notificação das atualizações é algo

a ser trabalhado posteriormente, junto ao fato que a comunicação entre os *brokers* conectados entre si não garante um encapsulamento de dados, que apesar de a *blockchain* não permitir a atualização indevida, não existe uma possibilidade de impedir que os dados da atualização sejam interceptados por um ataque de ***Man-In-The-Middle*** de um agente malicioso afim de identificar os tipos de dados contidos dentro do sistema por um meio de engenharia social.

A utilização da ferramenta para POCs e MVPs é uma solução bastante interessante, tendo em vista que esta não consome muitos recursos do computador. No caso de estudos acadêmicos, é uma solução didática e que ajuda a entender o funcionamento da troca de informações entre os *brokers* e o uso de *blockchain***.**

## REFERÊNCIAS

BANAFA, Ahmed; **IoT and *Blockchain* Convergence: Benefits and Challenges**. IEEE, 2017. Disponível em: <https://iot.ieee.org/newsletter/january-2017/iot-and-*blockchain*-convergence-benefits-and-challenges.html>.

BRAGA, Alexadre. 2017 **TECNOLOGIA *BLOCKCHAIN*: Fundamentos, Tecnologias de Segurança e Desenvolvimento de Software.** Disponível em: <https://www.cpqd.com.br/wp-content/uploads/2017/09/whitepaper_*Blockchain*_fundamentos_tecnologias_de_seguranca_e_desenvolvimento_de_softwar_FINAL.pdf>.

CABRINI, Fábio H; BARROS CASTRO FILHO, Albérico; V. FILHO,Filippo; T. KOFUJI, Sergio. **Helix: An Open Platform to Fast Prototype *Smart* Environments Applications***.* São Paulo, Brasil 2019.

CABRINI, Fábio H; V. FILHO, Filippo; T. KOFUJI, Sergio. **Arquiteturas de Fog Computing para Internet das coisas nas plataformas Fiware e Helix**. São Paulo, Brasil 2019.

CABRINI, Fabio, 2019. **Helix.** Disponível em: <https://github.com/fabiocabrini/helix-SandBox/blob/master/README.md>.

FREDERIC, Paul. **10 principais vulnerabilidades da Internet das Coisas**. CIO, 2019. Disponível em: <https://cio.com.br/10-principais-vulnerabilidades-da-internet-das-coisas/>

FIWARE, ***Context Broker* Federation**, 2019. Disponível em: < https://fiware-orion.readthedocs.io/en/master/user/federation/index.html >. Acesso em: 13 de Março de 2020.

FIWARE DATA MODELS, Disponível em: <https://fiware-datamodels.readthedocs.io/en/latest/index.html>, Acesso em: 27 de Maio de 2020.

HELIX, GetHelix.Org, Disponível em: <https://github.com/Helix-Platform/Sandbox-NG>, Acesso em: 7 de Maio de 2020.

IBM, 2011. **Introduction to InfoSphere Federation Server,** Disponível em: <https://www.ibm.com/support/knowledgecenter/en/SSZJPZ_8.7.0/com.ibm.swg.im.i

is.productization.iisinfsv.overview.doc/topics/cisofedintro.html>. Acesso em: 22 de Outubro de 2019.

ONGARO, Diego; OUSTERHOUT, John. **In Search of an Understandable Consensus Algorithm**. USENIX, Stanford University, 2014.

M. Serrano, S. Davy, M. Johnsson, W. Donnelly, A. Galis - "Review and Designs of Federated Management in Future Internet Architectures" parte do livro "The Future Internet - Future Internet Assembly 2011: Achievements and Technological Promises" Vol. 6656, pg 465, 4 Maio 2011.

Metacpan. **Crypt::Eksblowfish**. Disponível em: <https://metacpan.org/pod/Crypt::Eksblowfish:>. Acesso em: 25 de Junho de 2020.

PyPi – Pymongo 3.10.1. Disponível em: <https://pypi.org/project/pymongo/>. Acesso em: 25 de Junho de 2020.

Python - crypt — **Function to check Unix passwords**. Disponível em: <https://docs.python.org/3/library/crypt.html>. Acesso em 25 de Junho de 2020.

Python - urllib.parse — **Parse URLs into components**. Disponível em: <https://docs.python.org/3/library/urllib.parse.html>. Acesso em: 25 de Junho de 2020.

RODRIGO SANCHES, André. **Fundamentos de Armazenamento e Manipulação de Dados**. USP 2005. Disponível em: <https://www.ime.usp.br/~andrers/aulas/bd2005-1/aula7.html>. Acesso em 18 de Junho de 2020.

RF Wireless World. **IoT Sensors**. Disponível em: <https://www.rfwireless-world.com/Terminology/IoT-sensors.html>. Acesso em: 18 de Junho de 2020.

**Apêndice A – Estrutura de uma entidade**

Uma **entidade** (*entity*) é um objeto que existe e é distinguível dos outros objetos. Por exemplo, Paulo Silva com número de CPF 123.456.789-00 é uma entidade, visto que isso identifica unicamente uma pessoa particular do universo.

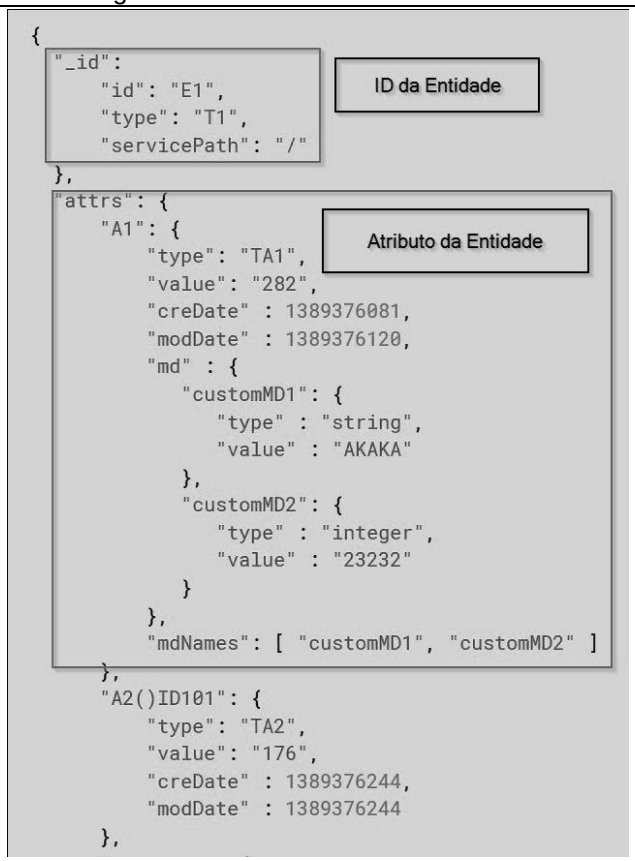
Assim a conta número 40167-9 na agência Lapa é uma entidade que identifica unicamente uma conta corrente particular. Uma entidade pode ser **concreta**, como uma pessoa ou um livro, ou pode ser **abstrata**, como um feriado ou um conceito (SANCHES, 2005).

Ao criar uma coleção de entidades, por padrão, é estruturada seguindo o seguinte formato:

- _**id** armazena o EntityID, incluindo o próprio ID e tipo. Como usamos _id para isso, garantimos que os EntityIDs sejam únicos.
- **attrs** é um mapa-chave dos diferentes atributos que foram criados para essa entidade. A chave é gerada com o nome do atributo
- **attrNames:** uma matriz de strings. Seus elementos são os nomes dos atributos da entidade (sem IDs). Nesse caso, o "." para "=" a substituição não está concluída.
- **creDate:** o registro de data e hora (como número inteiro) correspondente à data de criação da entidade (como conseqüência do anexo).
- **modDate:** o registro de data e hora (como número inteiro) correspondente à última atualização da entidade. Observe que ele costuma ser o mesmo que um modDate correspondente a pelo menos um dos atributos (nem sempre: não será o mesmo se a última atualização for uma operação DELETE). Corresponde a creDate se a entidade não tiver sido modificada após a criação.
- **local (opcional):** local geográfico da entidade
- **lastCorrelator:** valor do cabeçalho Fiware-Correlator na última solicitação de atualização na entidade. Utilizado pela lógica de proteção do loop de auto notificação.
- **expDate (opcional):** registro de data e hora de expiração (como um objeto Data) para a entidade (FIWARE, 2020).

A figura 17 exemplifica a modelagem de uma entidade com diversos dos atributos citados.

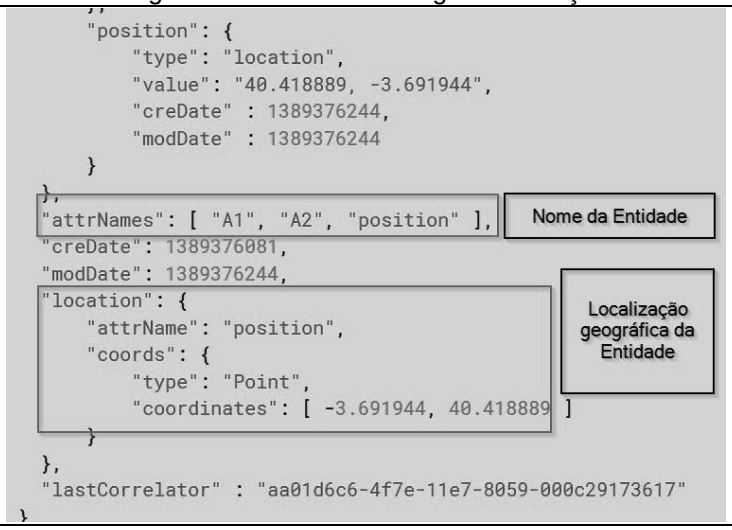
Figura 22 – Estrutura de uma entidade




Fonte: FIWARE, 2019

A figura 18 exemplifica a modelagem de dados de uma entidade com sensores de geolocalização.

Figura 23 - Entidade com geolocalização




Fonte: FIWARE, 2019

Note que as informações difere em cada tipo de sensor.

**Sensores que compõem uma Entidade.**

Os “tópicos” que estruturam uma entidade são conhecidos como sensores. Como sabemos, o mercado da Internet das coisas está crescendo e o principal componente que terá um enorme potencial de mercado é o sensor.

Existem duas divisões principais no back-end digital e no front-end da rede da IOT, principalmente composto por sensores da IOT. O principal benefício desse grande mercado será capturado pelas empresas de semicondutores.

Veremos sensores IoT que são usados para aplicações ópticas, de luz ambiente, temperatura, pressão, inércia, umidade, proximidade, gesto, toque e sensor de impressão digital. Esses sensores são baseados na tecnologia MEMS. MEMS é a forma abreviada de Microssistemas Eletromecânicos.

Se os sensores são digitais, é fácil interagir com o microcontrolador usando SPI. Se os sensores forem analógicos, é necessário o conversor de analógico para digital, conforme mencionado posteriormente em dispositivos analógicos.

Caso contrário, o modulador Sigma Delta pode ser usado para converter dados analógicos na saída SPI. É fácil fazer a interface de sinais SPI com qualquer dispositivo de microcontrolador (RFWORLD, 2012).

**Sensor de pressão**

Esse tipo de sensor detecta a pressão ao redor da terra, o que é muito útil para desenvolver o altímetro e o barômetro. O sensor de pressão LPS25H da ST-Microelectronics é um desses dispositivos sensores de IoT. É usado para navegação baseada em GPS.

**Acelerômetro e giroscópio**

O acelerômetro é usado para detectar vibração, inclinação e aceleração linear. É usado para implementação de pedômetro, nivelamento, alerta de vibração, antirroubo e muito mais. O giroscópio é usado para medir a velocidade angular. O giroscópio é usado principalmente em mouse 3D, jogos e treinamento de atletas.

A InvenSense Inc. desenvolveu o ICM-20628, que é uma solução combinada de acelerômetro e giroscópio de 6 eixos e baixa potência.

**Sensor de temperatura**

É usado para controlar o desempenho do dispositivo IoT em temperaturas variadas. Um exemplo de sensor de temperatura é o ADT7320. É sensor de temperatura digital com faixa de -40 graus C a +150 graus C. Ele tem interface direta com o microcontrolador em um dispositivo IoT.

**Sensor de umidade**

Semelhante ao sensor de temperatura, também é usado para controlar o desempenho do dispositivo. O HTU21D da Measurement Specialities é um sensor digital de umidade. A série NPA-700 de sensor de umidade da GE sensing fornece dados de saída digital usando o protocolo I2C padrão da indústria.

A série Si70xx da Silicon Laboratories Inc. também é usada como sensores de umidade e temperatura.

Sensor de proximidade

A família de sensores ópticos da Silicon Labs permite detectar o Índice UV, luz ambiente, proximidade de longo alcance, oximetria de frequência cardíaca / pulso e movimento com gestos 2D ou 3D. As séries de modelos Si114x, Si1132, Si1120 e Si1102 são sensores da Silicon Labs usados como sensores de proximidade por infravermelho.

**Sensor de toque**

O Silicon Labs possui microcontroladores de 8 e 32 bits, que são usados como aplicações de sensor de toque capacitivo. viz. controles deslizantes, rodas, botões de toque, sensor de nível de líquido e sensor de proximidade capacitivo.

**Conversor analógico para digital**

O AD7176-2 da Analog Devices é usado como conversor analógico para digital. Ele funciona em sinais de entrada de largura de banda muito baixa originados por sensores. Tem um tempo de acomodação mais rápido. É muito preciso e possui ADC sigma-delta de alta resolução.