

Trabalho Final de Internet das Coisas

Bruno Macena, Elenice Costa e Rodrigo Oliveira

SmartLock



Sumário

1. Introdução.....	3
2. Objetivos.....	3
2.1 Requisitos do sistema.....	4
2.2. Escopo do projeto.....	5
3. Projeto.....	6
3.1 UltraLight 2.0.....	7
3.2 Tráfego no sentido sul [Southbound] - Comandos.....	7
3.3 Comando Push usando HTTP POST.....	7
3.4. Tráfego sentido norte [Northbound] - Medições.....	8
3.5. Medição usando HTTP GET.....	9
3.6. Medição usando HTTP POST.....	9
4. Arquitetura.....	10
4.1. Componentes utilizados da Plataforma Fiware.....	10
4.2 Monitor de dispositivos.....	10
5. Aplicação.....	13
5.1. Pré-requisitos:.....	13
5.2 Instalação do ambiente Simulado.....	13
5.3. Instalação e configuração dos Parâmetros VM.....	14
5.4. Instalando Orion via Tutorial:.....	18
6. Teste da Aplicação.....	22
6.1. Abertura da Porta.....	22
6.2. Fechamento da Porta.....	23
6.3. Trancamento da Porta.....	24
7. Documentações de Setup.....	25
7.1. Setup: [Docker-compose.yml].....	25
7.2. Setup: [Iot Agent Ultralight 2.0].....	27
7.3. Criando e inicializando os containers e serviços.....	30
7.4 Provisionando uma porta inteligente.....	32
8. Testando IoT Agent via Ultralight 2.0.....	34
8.1. Abrindo a porta inteligente.....	34
8.2. Fechando a porta inteligente.....	36
8.3. Trancando a porta inteligente.....	37
9. Processamento em Python.....	39
Referências.....	42

1. Introdução

O setor elétrico é um alvo atraente para ataques cibernéticos e acesso indevido, ao ser essencial para o funcionamento da sociedade. As redes de energia elétrica são complexas e interconectadas, o que dificulta proteger contra acessos indevidos (Maia 2023). O setor elétrico brasileiro tem sido alvo de uma série de invasões cibernéticas nos últimos anos. Em 2021, por exemplo, a empresa de energia elétrica Light foi alvo de um ataque que causou uma interrupção no fornecimento de energia para mais de 2 milhões de pessoas no Rio de Janeiro. Segundo Maia (2023) o governo brasileiro tem tomado medidas para melhorar a segurança cibernética do setor elétrico. Em 2021, o governo criou o Centro Nacional de Coordenação de Incidentes Cibernéticos (CNCI), responsável por coordenar a resposta a ataques cibernéticos em infraestruturas críticas, incluindo o setor elétrico.

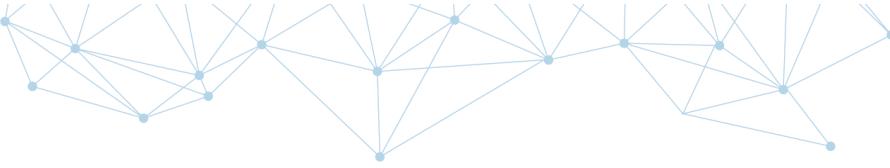
São medidas de segurança cibernética no setor elétrico, segundo Maia (2023):

- **Proteção de Infraestrutura crítica:** empresas elétricas devem identificar e proteger infraestruturas críticas, como usinas, subestações e sistemas de transmissão, contra ataques cibernéticos ou acesso indevido.
- **Monitoramento e Detecção:** sistemas de monitoramento em tempo real são fundamentais para identificar comportamentos anormais nos sistemas, sinalizando possíveis intrusões ou ameaças.
- **Segurança de Rede e Comunicação:** é crucial proteger as redes de comunicação para evitar acesso não autorizado e interceptação de informações sensíveis.
- **Gestão de Acesso:** controles rigorosos de acesso garantem interações apenas por pessoal autorizado

2. Objetivos

O projeto SmartLock tem como objetivo desenvolver um **sistema de controle de acesso autônomo para proteger equipamentos de self-healing em redes elétricas**, que pode ser utilizado por distribuidoras para manobras de automação na rede elétrica, que permite retomar o fornecimento de energia em uma área interrompida num curto período.

O SmartLock busca prevenir acessos não autorizados, mitigar riscos de falhas catastróficas e promover a segurança pública. Ele utilizará um sofisticado sensor de alarme combinado com um atuador para controle automatizado de aberturas de porta e um sistema de identificação por RFID, assegurando manutenções seguras e autorizadas. A implementação deste sistema não só eleva a confiabilidade operacional das infraestruturas críticas, mas também estabelece



um novo padrão de segurança operacional, contribuindo significativamente para a resiliência das smart cities.

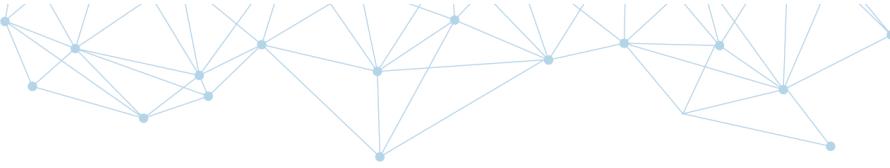
2.1 Requisitos do sistema

- **Requisitos Funcionais**

- a. Consultar o estado da porta para detectar e responder a acessos não autorizados.
- b. Processar informações da tag RFID de identificação para autenticar o acesso dos técnicos.
- c. Atuar mecanismos de travamento/destravamento de porta com base na autenticação.
- d. Projetar uma interface de usuário para monitoramento operacional, incluindo no mínimo status (open, closed, locked).
- e. Estabelecer a comunicação do sistema de controle de acesso com o centro operacional.

- **Requisitos não Funcionais**

- a. **Protocolos de Comunicação Padrão** - O sistema deve aderir a protocolos de comunicação padrão da indústria para garantir interoperabilidade com diferentes dispositivos e sistemas [HTTPS e REST].
- b. **Tolerância a Falhas** – A abertura de portas deve continuar operando corretamente na presença de falhas de hardware ou software. [Utilização de chave física em situação de contingência]
- c. **Registro de Atividades** - Deve haver repositório que registra as atividades para facilitar a análise de falhas.
- d. **Tempo de Resposta em milissegundos** - O sistema deve ter um tempo de resposta de controle de acesso e sensoriamento. [Pacotes de dados minimalistas]
- e. **Robustez Contra Falhas** - O sistema deve ser robusto contra falhas, com capacidade de manter operações críticas mesmo em condições de desastres climáticos.
- f. **Disponibilidade do Sistema** - O sistema deve manter alta disponibilidade, minimizando interrupções. [Arquitetura distribuída com contingenciamento].

- 
- g. **Recuperação Rápida** - Em caso de falhas, o sistema deve ser capaz de se recuperar rapidamente. [Arquitetura com Alta disponibilidade].
 - h. **Capacidade Expansível** - O sistema deve ser projetado para permitir a inclusão fácil de novos componentes.
 - i. **Modularidade** - A estrutura do sistema deve ser modular para facilitar ajustes e expansões. [Hardware de baixo custo e compartilhamento de acessos para dados].
 - j. **Padrões de Codificação** - O software deve ser desenvolvido seguindo padrões de codificação e boas práticas de mercado para facilitar a manutenção.
 - k. **Documentação** - Deve haver uma documentação com Plano do Projeto, Canvas e códigos do software.
 - l. **Modularidade do Código** - O código deve ser modular para permitir atualizações e modificações com mínimo esforço e custo.
 - m. **Suporte a diferentes navegadores** - O sistema deve funcionar de maneira consistente em vários navegadores web populares. [utilização de sistema – Dashboard em HTML HTTPS].
 - n. **Comunicação Segura** - Toda comunicação de dados entre o sistema SmartLock e outros sistemas deve ser realizada de forma segura. [utilização de HTTPS]
 - o. **Interface Intuitiva** - A interface do usuário deve ser intuitiva, fácil de navegar e acessível para todos os usuários.
 - p. **Armazenamento de Dados** - O sistema deve utilizar mecanismos de armazenamento de dados que sejam adequados para um ambiente IOT.

2.2. Escopo do projeto

O projeto SmartLock é implementado na plataforma Fiware tem como escopo proporcionar ao setor de energia um controle autônomo em infraestrutura críticas de energia. O desenvolvimento da aplicação respeitará as limitações de tempo e orçamento. Portanto, irá se restringir às funcionalidades essenciais e o foco será manter o protótipo limitado às funcionalidades necessárias para validar o conceito, evitando complexidades adicionais.

Dos requisitos funcionais serão implementados os itens “a, b e c” com a integração da plataforma limitada apenas aos componentes essenciais da plataforma Fiware e testados

com dados simulados. O requisito funcional “d” se utilizará da visualização própria do sistema Fiware para monitoramento dos dados. Já o requisito funcional “e” será implementado em momentos futuros uma vez que este projeto se trata da construção de um mínimo produto viável (MVP) e parcerias futuras com mais de um sistema próprio de empresas da rede elétrica possam surgir.

Dos requisitos não funcionais, o foco será nos itens: “a, b, d, j, n e p” que são respectivamente: Protocolos de Comunicação Padrão, Tolerância a Falhas, Tempo de Resposta, Padrões de Codificação, Comunicação Segura e Armazenamento de Dados. Todos foram tratados e definidos a partir das prioridades do projeto com as partes interessadas.

3. Projeto

O SmartLock é implementado na Plataforma Fiware, com protocolo UltraLight 2.0, utilizando o Orion Context Broker e o Agente IoT. Ele apresenta uma tranca (Porta) inteligente exibido em um navegador e permitindo que o usuário interaja com ela. Uma tranca inteligente é um componente de uma porta eletrônica à qual podem ser enviados comandos para ser trancada ou destravada remotamente. Também pode informar sobre seu estado atual (*open*, *close* ou *locked*).

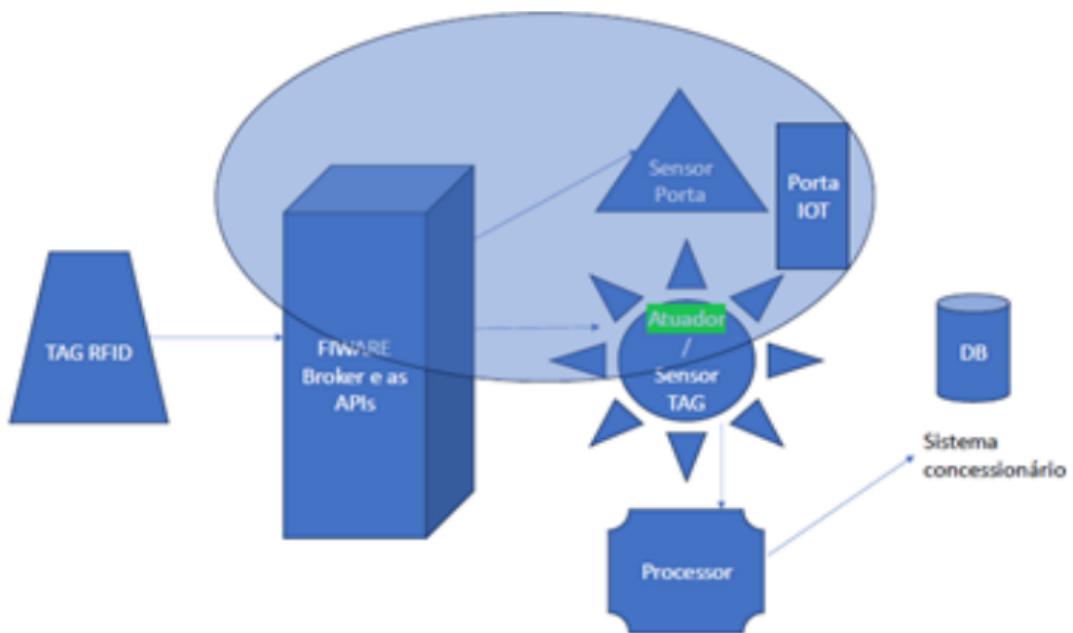


Figura 1 – Esquema dos principais componentes do Projeto. A área circular representa o foco de implementação neste MVP. Fonte: autores

Como o projeto tem por base a plataforma Fiware, os estados da informação no **Context Broker** são representados a seguir:

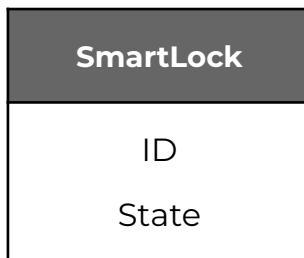


Figura 1 – Estados. Fonte: autores

3.1 UltraLight 2.0

É um protocolo leve baseado em texto para dispositivos e comunicações restritas onde a largura de banda e os recursos de memória do dispositivo são limitados. A carga útil para solicitações de medição é uma lista de pares de valores-chave separados pela barra vertical "|". Por exemplo:

Unset

```
<chave>|<valor>|<chave>|<valor>|<chave>|<valor>
```

Por exemplo, uma carga útil como:

Unset

```
t|15|k|abc
```

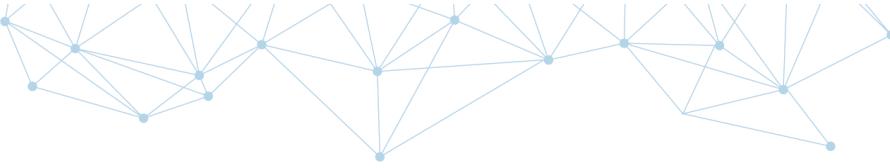
Neste caso, temos dois atributos, um denominado "t" com valor "15" e outro denominado "k" com valor "abc" são transmitidos. Os valores no Ultralight 2.0 são todos tratados como uma string.

3.2 Tráfego no sentido sul [Southbound] - Comandos

As solicitações HTTP geradas pelo Context Broker e passadas para baixo em direção a um dispositivo IoT (por meio de um Agente IoT) são conhecidas como **tráfego sul**. O tráfego no sentido sul consiste em comandos dados a dispositivos atuadores que alteram o estado do mundo real por meio de suas ações. Por exemplo, um comando para alterar o estado da tranca inteligente para OPEN abrirá a porta na vida real. Isto, por sua vez, pode alterar as leituras de outros sensores próximos.

3.3 Comando Push usando HTTP POST

A configuração da comunicação sul entre um Agente IoT e dispositivos IoT é conhecida como provisionamento. Isso garante que o Agente IoT retenha informações suficientes para poder entrar em contato com cada dispositivo IoT. Em outras palavras, ele sabe para onde enviar



comandos e quais comandos são suportados. Para enviar um comando a um dispositivo, o Agente IoT envia uma solicitação POST ao endpoint fornecido pelo dispositivo. O corpo da solicitação POST contém o comando.

A carga útil para comandos Ultralight tem o seguinte formato:

Unset

```
<device_name>@<comand> | <param> | <param>
```

Onde **<device_name>** é o **ID** da entidade mantida no intermediário de contexto, **<command>** é um dos comandos suportados e quaisquer valores adicionais necessários são passados em parâmetros subsequentes, por exemplo:

Unset

```
urna:ngsi-1d:Door:001@open
```

Dirá a um dispositivo "Sou conhecido como id="urn:ngsi-Id:Door:001" dentro do Context Broker. Gostaria que o dispositivo escutando neste endpoint executasse o comando open. A resposta *Northbound*, ou seja, do dispositivo IoT para o Context Broker é a seguinte:

Unset

```
urna:ngsi-1d:Door:001@Open|Open ok
```

O que significa "**Cumpri uma solicitação da entidade conhecida como id="urn:ngsi-Id:Robot:001**" dentro do Context Broker. O comando que executei foi um comando turn. O resultado foi Turn ok".

Como você pode ver, como o comando *Southbound* define o id usado na interação e os mesmos dados também são retornados, cada resposta sempre pode ser associada à entidade apropriada mantida no Context Broker.

Os comandos push só podem ser usados se o dispositivo for capaz de fornecer um endpoint separado para escutar o tráfego em direção ao sul. Um mecanismo de pesquisa alternativo pode ser usado quando todas as interações são iniciadas no próprio dispositivo, mas isso está além do escopo deste tutorial.

3.4. Tráfego sentido norte [Northbound] - Medições

As solicitações geradas a partir de um dispositivo IoT e repassadas para o Context Broker (por meio de um agente IoT) são conhecidas como tráfego norte. O tráfego no sentido norte



consiste em medições feitas por dispositivos sensores e transmite o estado do mundo real nos dados de contexto do sistema. Por exemplo, uma medição de um sensor de abertura de porta (não comandada) poderia ser retransmitida de volta ao intermediário de contexto para indicar que o estado da mesma mudou sem que tenha havido participação do atuador. Uma assinatura poderia ser feita para ser informado de tais mudanças e provocar outras ações (como ligar um alarme)

3.5. Medição usando HTTP GET

Um dispositivo pode relatar novas medidas a um Agente IoT usando uma solicitação HTTP GET para um endpoint "conhecido" (o caminho `/iot/d`) junto com os seguintes parâmetros de consulta:

- **i** (ID do dispositivo): ID do dispositivo (exclusivo para a chave API).
- **k** (API Key): Chave API do serviço no qual o dispositivo está registrado.
- **t** (timestamp): Timestamp da medida. Substituirá o carimbo de data/hora automático do IoTAgent (opcional).
- **d** (Dados): Carga útil Ultralight 2.0.

É importante citar que os parâmetros **i** e **k** são obrigatórios.

Vamos a um exemplo de solicitação:

```
Unset
<agente-iot>/iot/d?i= Door:001&d=c|ForcedOpen
```

Indicaria que o dispositivo **id=motion001** deseja informar ao Agente IoT que fez uma medição do mundo real **c** com o valor **ForcedOpen**. Isso eventualmente seria repassado ao Context Broker.

3.6. Medição usando HTTP POST

HTTP POST também pode ser usado. Novamente o caminho será `/iot/d`, mas neste caso, d (Data) não é necessário - os pares chave-valor da medição são passados como o corpo da solicitação. Os parâmetros de consulta i e k ainda são obrigatórios:

- **i** (ID do dispositivo): ID do dispositivo (exclusivo para a chave API).
- **k** (API Key): Chave API do serviço no qual o dispositivo está registrado.

- **t** (timestamp): Timestamp da medida. Substituirá o carimbo de data/hora automático do IoTAgent (opcional).

4. Arquitetura

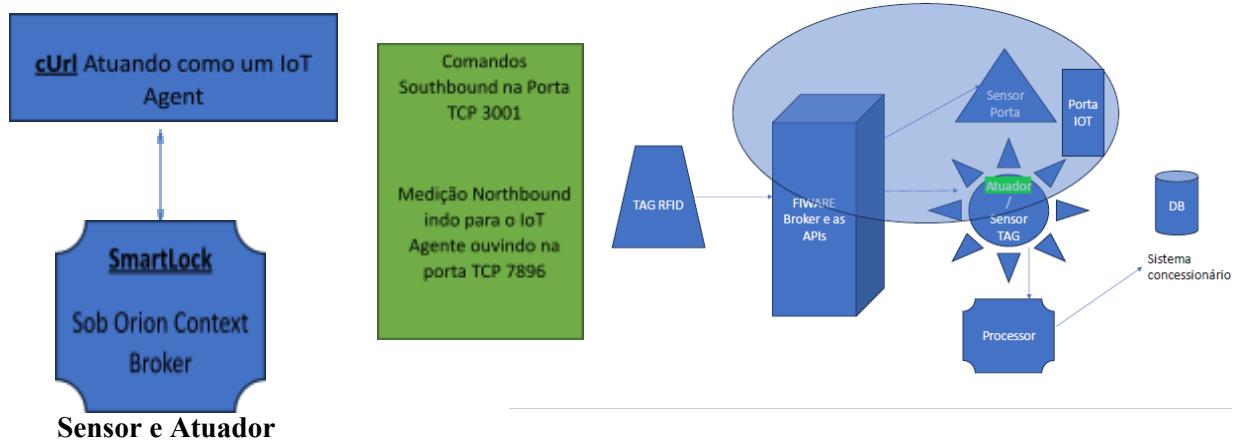


Figura 2 – Arquitetura do projeto. Fonte: autores

O aplicativo de demonstração usará apenas um único componente personalizado que atua como um conjunto de dispositivos IoT. Cada dispositivo IoT usará o protocolo UltraLight 2.0 executado em HTTP. Como todas as interações são iniciadas por solicitações HTTP, as entidades podem ser containerizadas e executadas a partir de portas expostas.

4.1. Componentes utilizados da Plataforma FIWARE

Para a base do projeto foram utilizados alguns container's iniciais:

- **Projeto FIWARE: FIWARE 201:** Introduction to IoT Sensors [tutorials.IoT-Sensors]
- **Tutorial Principal:** <https://fiware-iotagent-ul.readthedocs.io/>
- **Imagen GIT:** <https://github.com/FIWARE/tutorials.IoT-Sensors.git>

4.2 Monitor de dispositivos

Para os fins deste trabalho, foram utilizados dois dispositivos IoT fictícios, que eventualmente serão anexados ao Context Broker. O estado de cada dispositivo pode ser visto na página da web do monitor de dispositivos UltraLight, encontrada em:

Unset

<http://localhost:3000/device/monitor>



IoT Devices (Ultralight Over HTTP)

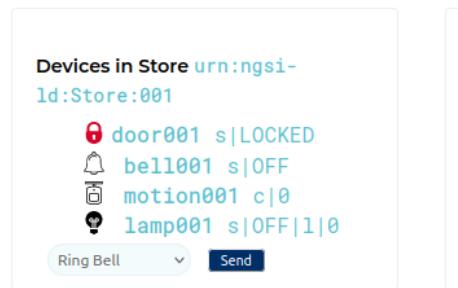


Figura 3 – Devices. Fonte: Fiware

As informações de configuração necessárias podem ser vistas na seção de serviços do arquivo **docker-compose.yml** associado:

```
Unset
image: quay.io/fiware/tutorials.context-provider
  hostname: iot-sensors
  container_name: fiware-tutorial
  networks:
    - default
  expose:
    - '3000'
    - '3001'
  ports:
    - '3000:3000'
    - '3001:3001'
```

environment:

```
- 'DEBUG=tutorial:*
- 'PORT=3000'
- 'IOTA_HTTP_HOST=iot-agent'
- 'IOTA_HTTP_PORT=7896'
- 'DUMMY_DEVICES_PORT=3001' # Porta usada pelos dispositivos IoT
fictícios para receber comandos
- 'DUMMY_DEVICES_API_KEY=4jggokgpepnvsb2uv4s40d59ov'
```

O contêiner está escutando em duas portas:

- A porta 3000 é exposta para podermos ver a página da web exibindo os dispositivos Dummy IoT.
- A porta 3001 é exposta apenas para que cUrl ou Postman possam executar comandos UltraLight sem fazer parte da mesma rede.
- O contêiner é orientado por variáveis de ambiente conforme mostrado no tutorial base do projeto

Tabela 1 - Descrição dos componentes.

Chave	Valor	Descrição
DEBUG	tutorial:*	Sinalizador de depuração usado para registro
WEB_APP_PORT	3000	Porta usada pelo aplicativo da web que exibe os dados fictícios do dispositivo
IOTA_HTTP_HOST	iot-agent	O nome do host do Agente IoT ausente
IOTA_HTTP_PORT	7896	A porta em que o Agente IoT ausente estará escutando. 7896 é um padrão comum para UltraLight sobre HTTP
DUMMY_DEVICES_PORT	3001	Porta usada pelos dispositivos IoT fictícios para receber comandos

Chave	Valor	Descrição
DUMMY_DEVICES _API_KEY	4jggokgpepn vsb2uv4s40d 59ov	Chave de segurança aleatória usada para interações UltraLight - será usada em um tutorial posterior para garantir a integridade das interações entre os dispositivos e o Agente IoT

Os outros valores de configuração do contêiner descritos no arquivo YAML não são usados neste projeto (são oriundos do tutorial base). O Orion Context Broker é usado para armazenar os dados de contexto da solução inteligente. Todas as interações com o context broker devem ser feitas usando NGSI-v2

Um Agente IoT atua como um componente de middleware convertendo solicitações NGSI-v2 (do intermediário de contexto) em um protocolo (como UltraLight 2.0) utilizável pelos próprios dispositivos IoT.

5. Aplicação

Nesta seção listamos um passo a passo para criar o ambiente e executar as funções relacionadas ao projeto.

5.1. Pré-requisitos:

- **Docker:** tecnologia de contêineres que permite isolar diferentes componentes em seus respectivos ambientes.
- **Docker Compose:** é uma ferramenta para definir e executar aplicativos Docker com vários contêineres. Um arquivo YAML é usado para configurar os serviços necessários para o aplicativo. Isso significa que todos os serviços de contêiner podem ser ativados em um único comando.

5.2 Instalação do ambiente Simulado

Utilizamos neste projeto o Sistema Operacional Linux Ubuntu 20.04.01 LTS (imagem Desktop). É possível baixá-lo por meio do link: <https://releases.ubuntu.com/20.04/>

Ele pode instalado em uma máquina virtual sob virtualizador Oracle VM VirtualBox 7.0.6, abstraindo o host base Windows 11 Pro.

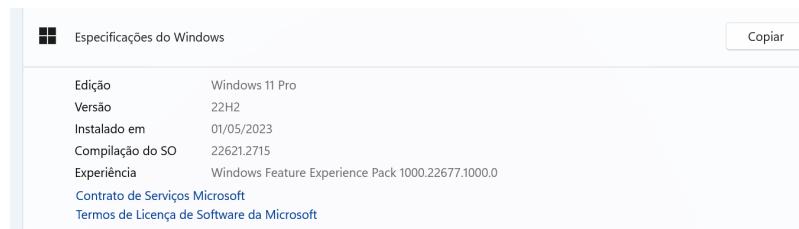
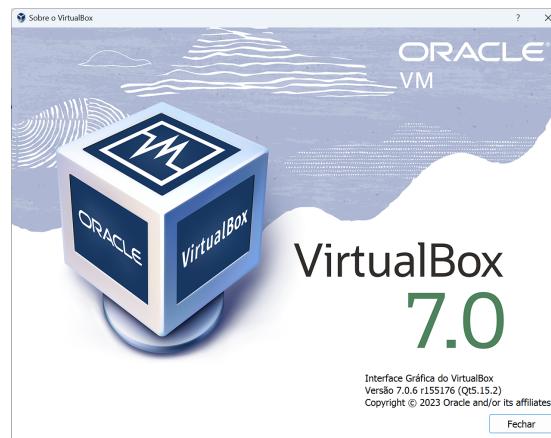


Figura 4 – Virtual box. Fonte: <https://releases.ubuntu.com/20.04/>

5.3. Instalação e configuração dos Parâmetros VM

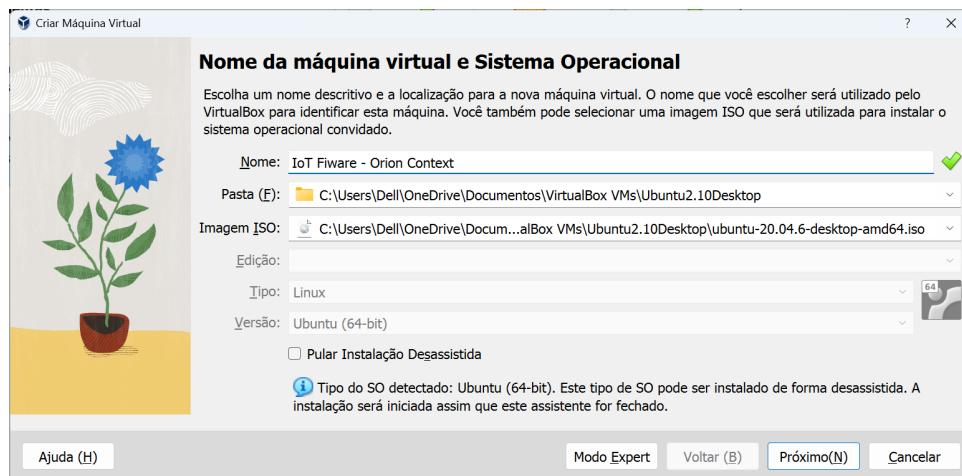


Figura 5 – Máquina virtual Fonte: autores

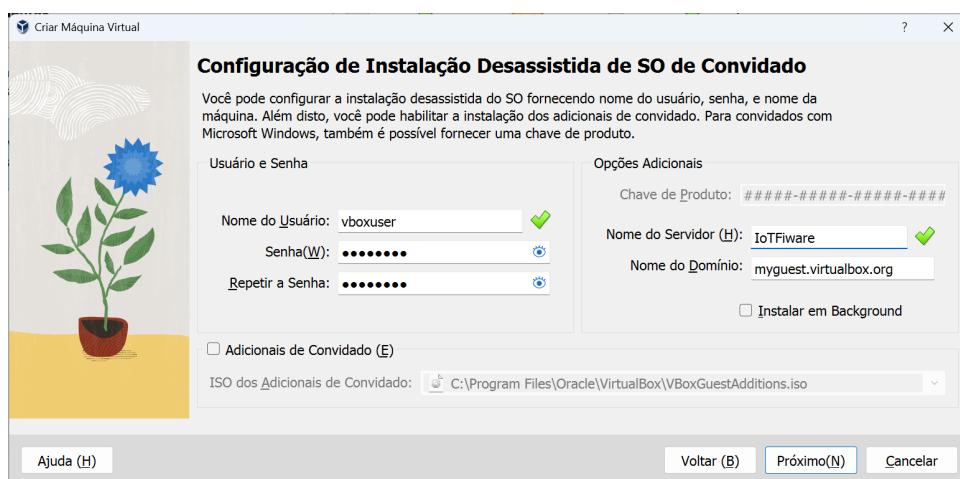


Figura 6 – Configuração e instalação Fonte: autores

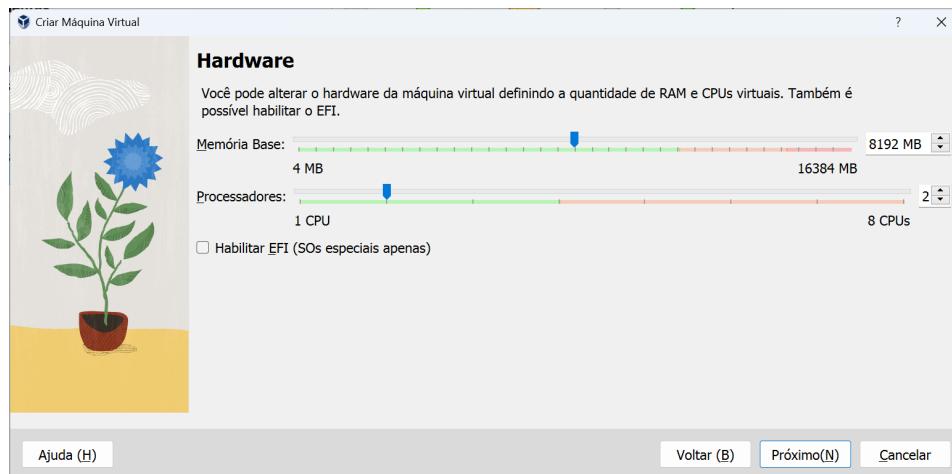


Figura 7 – Configuração do 1º HD

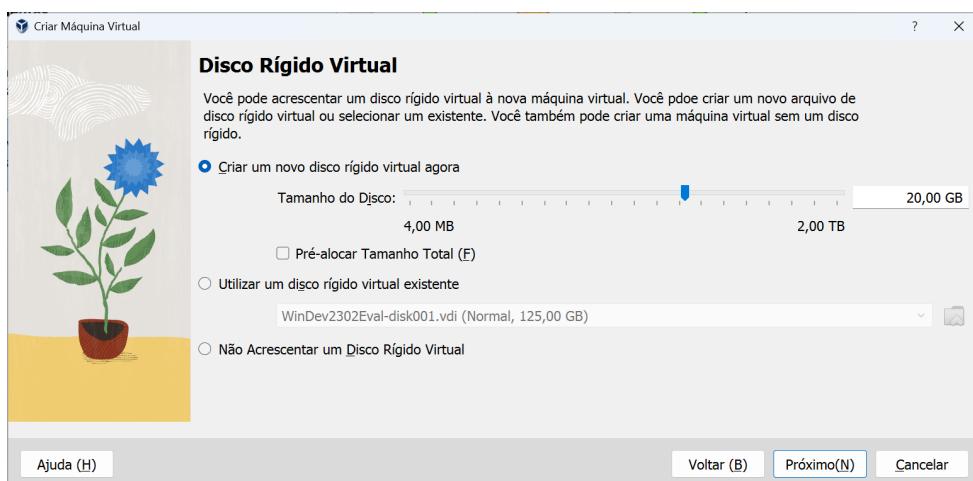


Figura 8 – Configuração do 2º HD. Fonte: autores

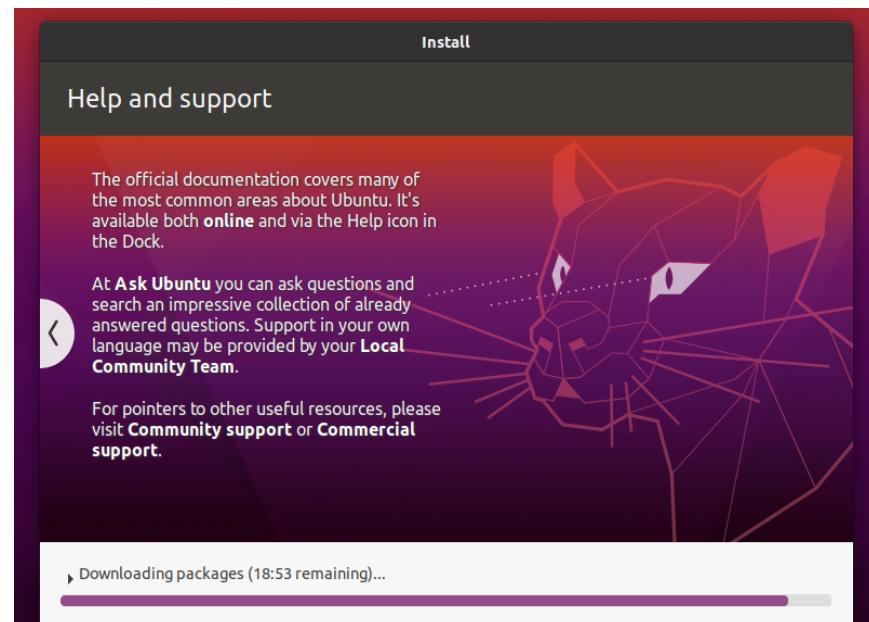


Figura 9 – Instalação do Linux Ubuntu. Fonte: autores

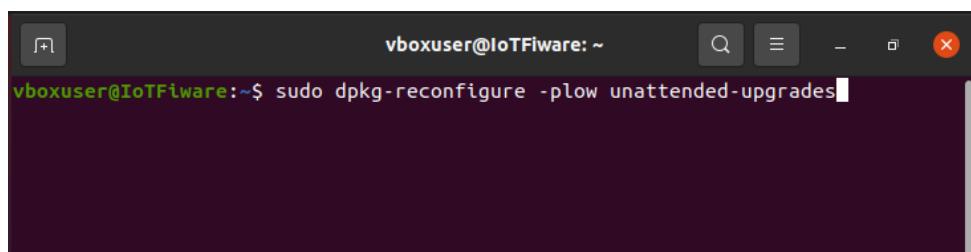


Figura 10 – Instalação do Linux Ubuntu. Fonte: autores

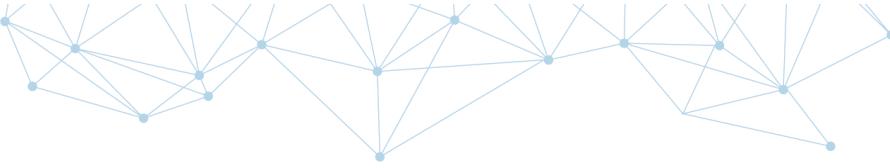
Para garantir compatibilidade dos pacotes com o Sistema operacional, deve-se (por hora) parar as atualizações automáticas do mesmo:

```
Unset
$ su root
$ dpkg-reconfigure -plow unattended-upgrades
```

Neste ponto, optou-se por não baixar e instalar atualizações. Além disso, foi realizada uma reinicialização da máquina virtual.

Deve-se certificar-se de que todos os pacotes em estado sujo estejam instalados:

```
Unset
$ sudo dpkg --configure -a
```



Mantenha o sistema atualizado.

Unset

```
$ sudo apt update && sudo apt -f install && sudo apt full-upgrade
```

Ligue novamente o atualizador automático, agora que o bloqueio foi eliminado.

Unset

```
$ sudo dpkg-reconfigure -plow unattended-upgrades
```

Selecione o pacote de atualizações autônomas novamente.

Alguns pacotes devem ser instalados agora. Foi utilizado o “aptitude” no lugar do padrão “apt-get” à fim de seguir fielmente a documentação disponibilizada em comunidade para o setup do ambiente base. Para ativar o aptitude:

Unset

```
$ sudo apt-get install -y aptitude
```

Para instalação das Ferramentas, execute o seguinte comando:

Unset

```
$ sudo aptitude install -y build-essential scons curl cmake
```

Para instalação das Bibliotecas, execute o seguinte comando:

Unset

```
$ sudo aptitude install -y libssl-dev gnutls-dev \
libcurl4-gnutls-dev libsasl2-dev \
libgcrypt-dev uuid-dev libboost1.67-dev libboost-regex1.67-dev \
libboost-thread1.67-dev \
libboost filesystem1.67-dev libz-dev libmongoclient-dev
```

Para o setup do repositório APT Docker, execute os seguintes comandos:

Unset

```
# Add Docker's official GPG key:
```

```
$ sudo apt-get update  
$ sudo apt-get install ca-certificates curl gnupg  
$ sudo install -m 0755 -d /etc/apt/keyrings  
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg  
$ sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg  
$ sudo chmod a+r /etc/apt/keyrings/docker.gpg  
# Add the repository to Apt sources:  
$ echo \  
"deb [arch=$(dpkg --print-architecture)  
signed-by=/etc/apt/keyrings/docker.gpg]  
https://download.docker.com/linux/ubuntu \  
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \  
$ sudo tee /etc/apt/sources.list.d/docker.list > /dev/null  
$ sudo apt-get update
```

Para instalação dos pacotes do Docker, execute o seguinte comando:

Unset

```
$ apt-get install docker-ce docker-ce-cli containerd.io /  
docker-buildx-plugin docker-compose-plugin
```

Para verificar se a instalação do Docker Engine foi bem-sucedida, execute a imagem “hello-world”.

Unset

```
$ sudo docker run hello-world
```

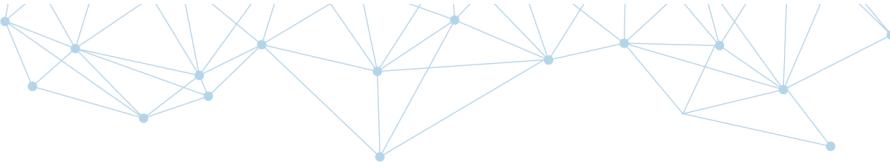
5.4. Instalando Orion via Tutorial:

Todos os serviços podem ser inicializados a partir da linha de comando executando o script bash fornecido no repositório. Clone o repositório e crie as imagens necessárias executando os comandos conforme mostrado a seguir:

Unset

```
$ git clone https://github.com/FIWARE/tutorials.IoT-Sensors.git
```

```
$ cd tutorials.IoT-Sensors
$ git checkout NGSI-v2
$ ./services create; ./services start;
```

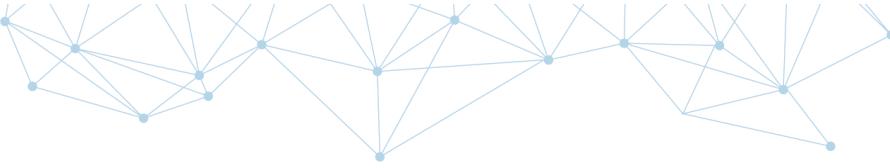


A terminal window showing the output of a Docker pull command. The command is pulling images for Orion Context Broker, MongoDB, and a tutorial container. The output shows the progress of each layer being pulled, including the size and time taken.

```
vboxuser@IoTFiware: ~/tutorials.IoT-Sensors
;
Pulling Docker images
quay.io/curl/curl:latest
WARN[0000] The "IOTA_SOUTH_PORT" variable is not set. Defaulting to a blank string.
WARN[0000] The "host" variable is not set. Defaulting to a blank string.
[+] Pulling 5/21
  :: mongo-db 9 layers [██████] 0B/0B      Pulling          108.3s
    :: 7a2c55901189 Waiting           95.5s
    :: b57cf58c4ffe Waiting           95.5s
    :: 5bb145f2aa46 Waiting           95.5s
    :: 93f83d52d64e Waiting           95.5s
    :: 5e012812179b Waiting           95.5s
    :: a6343ab9f99f Waiting           95.5s
    :: db691c26a2f1 Waiting           95.5s
    :: 5309bfc44820 Waiting           95.5s
    :: 6fb4bc4192f3 Waiting           95.5s
  :: orion 4 layers [██████] 40.85MB/91.11MB Pulling          108.3s
    :: 26c5c85e47da Downloading 19.27MB/31.42MB          105.1s
    ✓ 4f4fb700ef54 Download complete          1.1s
    :: 8356ea06043a Downloading 21.58MB/59.69MB          105.1s
    ✓ 12728836f1e3 Download complete          2.7s
  :: tutorial 5 layers [██████] 5.896MB/32.02MB Pulling          108.3s
    ✓ fc251a6e7981 Pull complete          8.9s
    ✓ fda4ba87f6fb Pull complete          46.7s
    ✓ a1f1879bb7de Pull complete          51.9s
    :: 51dfccb876247 Downloading 5.896MB/32.02MB          105.1s
    :: 0cae9754194 Waiting           105.1s
```

Figura 11 – instalando Orion Broker. Fonte: autores

O processo cria a rede virtual do ambiente Fiware, o volume do Banco de Dados MongoDB, e os containers do Orion Context Broker e dos Dummy IoT devices.



```
vboxuser@IoTFlware: ~/tutorials.IoT-Sensors
```

Starting one container **Tutorial**
- Tutorial acts as a series of dummy IoT Sensors over HTTP

WARN[0000] The "IOTA_SOUTH_PORT" variable is not set. Defaulting to a blank string.
WARN[0000] The "host" variable is not set. Defaulting to a blank string.

[+] Running 5/5
✓ Network `fiware_default` `Created` 0.1s
✓ Volume "fiware_mongo-db" `Created` 0.0s
✓ Container `db-mongo` `Started` 0.1s
✓ Container `fiware-tutorial` `Started` 0.1s
✓ Container `fiware-orion` `Started` 0.1s

⌚ Waiting for `MongoDB` to be available

⌚ Waiting for `Orion` to be available

⌚ Loading context data `done`

NAMES	STATUS	PORTS
<code>fiware-orion</code>	Up 6 seconds (healthy)	0.0.0.0:1026->1026/tcp, :::
<code>1026->1026/tcp</code>		
<code>fiware-tutorial</code>	Up 6 seconds (health: starting)	0.0.0.0:3000-3001->3000-3001/tcp, :::3000-3001->3000-3001/tcp

Now open <http://localhost:3000/device/monitor>

```
vboxuser@IoTFlware:~/tutorials.IoT-Sensors$
```

Figura 12 – Instalação do MongoDB, Orion e Dummy. Fonte: autores



IoT Devices (Ultralight Over HTTP)



Figura 13 – Fiware. Fonte: Teste: <http://localhost:3000/device/monitor>

Neste ponto é necessário editar o arquivo **docker-compose.yml** com os parâmetros:

Unset

```
image: quay.io/fiware/tutorials.context-provider
  hostname: iot-sensors
  container_name: fiware-tutorial
  networks:
    - default
  expose:
    - '3000'
    - '3001'
  ports:
    - '3000:3000'
    - '3001:3001'
  environment:
    - 'DEBUG=tutorial:/*'
    - 'PORT=3000'
    - 'IOTA_HTTP_HOST=iot-agent'
    - 'IOTA_HTTP_PORT=7896'
    - 'DUMMY_DEVICES_PORT=3001' #Porta usada pelos
      dispositivos IOT fictícios para receber comandos
    - 'DUMMY_DEVICES_API_KEY=4jggokgpepnvsb2uv4s40d59ov'
```

É necessário parar os serviços, com o seguinte comando:

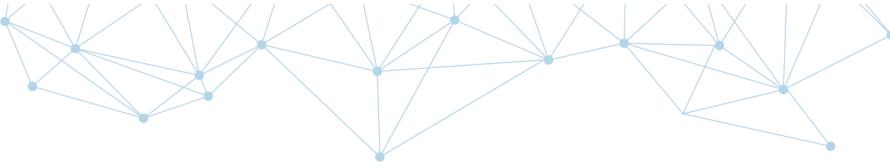
Unset

```
./services stop
```

Em seguida, são reiniciados os novos parâmetros:

Unset

```
./services create; ./services start;
```



```
vboxuser@IoTFIware: ~/tutorials.IoT-Sensors
✓ orion Pulled                                2.2s
✓ tutorial Pulled                            2.2s
Starting one container Tutorial
- Tutorial acts as a series of dummy IoT Sensors over HTTP

WARN[0000] The "host" variable is not set. Defaulting to a blank string.
[+] Running 5/5
  ✓ Network fiware_default      Created          0.1s
  ✓ Volume "fiware_mongo-db"    Created          0.0s
  ✓ Container fiware-tutorial  Started          0.1s
  ✓ Container db-mongo         Started          0.1s
  ✓ Container fiware-orion     Started          0.1s

  ⏳ Waiting for MongoDB to be available

  ⏳ Waiting for Orion to be available

  ⏳ Loading context data  done

  NAMES          STATUS           PORTS
  fiware-orion   Up 6 seconds (healthy)  0.0.0.0:1026->1026/tcp, :::
  1026->1026/tcp
  fiware-tutorial Up 6 seconds (health: starting)  0.0.0.0:3000-3001->3000-300
  1/tcp, :::3000-3001->3000-3001/tcp

Now open http://localhost:3000/device/monitor
vboxuser@IoTFIware:~/tutorials.IoT-Sensors$
```

Figura 14 - Serviços criados e inicializados. Fonte: autores

6. Teste da Aplicação

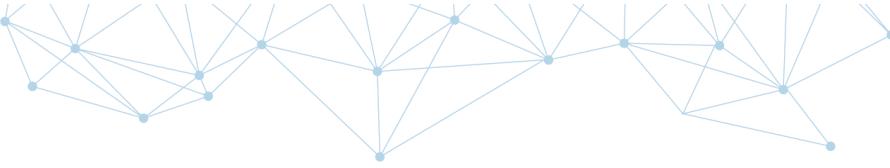
6.1. Abertura da Porta

Testando o comando de abertura de porta simulando um IoT agente utilizando **curl**:

```
Unset
$ curl -iX POST \
--url 'http://localhost:3001/iot/door001' \
--data urn:ngsi-ld:Door:001@open
```

Resposta esperada:

```
Unset
urn:ngsi-ld:Door:001@open| open OK
```



```
vboxuser@IoTFiware:~/tutorials.IoT-Sensors$ curl -iX POST \
>   --url 'http://localhost:3001/iot/door001' \
>   --data urn:ngsi-ld:Door:001@unlock
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 38
ETag: W/"26-pjGRJPy7HPAIQG8UBRHsAGe6tlQ"
Date: Mon, 27 Nov 2023 03:16:58 GMT
Connection: keep-alive
Keep-Alive: timeout=5

urn:ngsi-ld:Door:001@unlock| unlock OKvboxuser@IoTFiware:~/tutorials.IoT-Sensor
s$
```

Figura 15 - Mudança de status = aberta no Door001. Fonte: autores

IoT Devices (Ultralight Over HTTP)



Figura 16 - Mudança de status = Open no Door001. Fonte: autores

6.2. Fechamento da Porta

Para referência, o comando de fechamento de porta é:

```
Unset
$ curl -iX POST \
--url 'http://localhost:3001/iot/door001' \
--data urn:ngsi-ld:Door:001@close
```

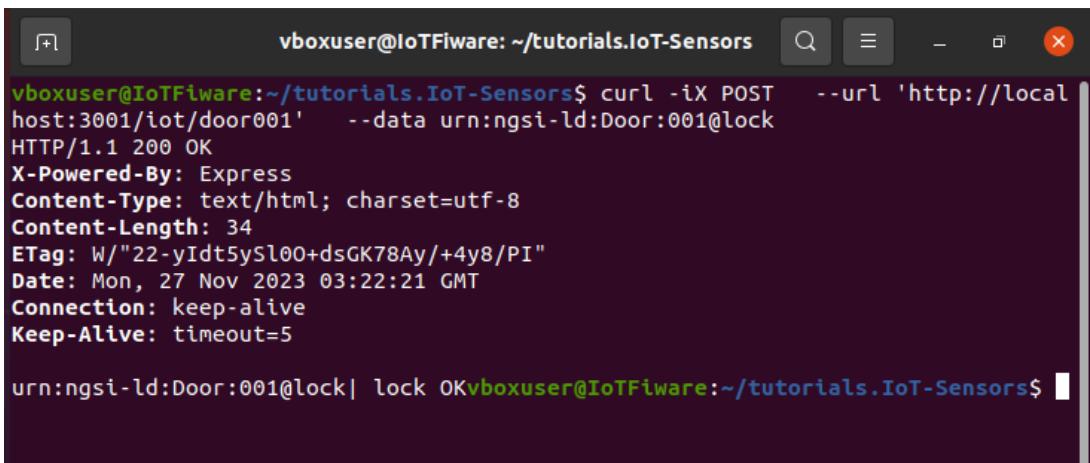
Resposta esperada:

Unset

```
urn:nsgsi-ld:Door:001@close| close OK
```

6.3. Trancamento da Porta

Agora trancando novamente:



```
vboxuser@IoTFlware:~/tutorials.IoT-Sensors$ curl -iX POST --url 'http://localhost:3001/iot/door001' --data urn:nsgsi-ld:Door:001@lock
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 34
ETag: W/"22-yIdt5ySl00+dsGK78Ay/+4y8/PI"
Date: Mon, 27 Nov 2023 03:22:21 GMT
Connection: keep-alive
Keep-Alive: timeout=5

urn:nsgsi-ld:Door:001@lock| lock OKvboxuser@IoTFlware:~/tutorials.IoT-Sensors$
```

Figura 17 - Mudança de status = locked no Door001. Fonte: autores

Foram executados os seguintes comandos:

Unset

```
$ curl -iX POST \
--url 'http://localhost:3001/iot/door001' \
--data urn:nsgsi-ld:Door:001@close
```

Resposta esperada:

Unset

```
urn:nsgsi-ld:Door:001@lock| lock OK
```



IoT Devices (Ultralight Over HTTP)

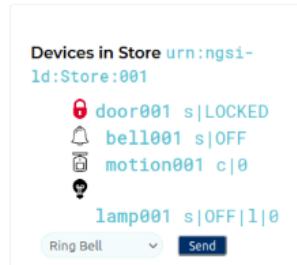


Figura 18 - Mudança de status = locked(travado) no Door001. Fonte: autores

7. Documentações de Setup

7.1. Setup: [Docker-compose.yml]

```

Unset

# Trabalho Pratico FIWARE - Internet Of Things [IoT]
# UFF - Universidade Federal Fluminense
# 2023-2
# Bruno Macena dos Santos
# Elenice Costa
# Rodrigo Oliveira
#
version: "3.8"
services:
  # Orion context broker
  orion:
    labels:
      org.fiware: 'tutorial'
    image: quay.io/fiware/orion:${ORION_VERSION}
    hostname: orion
    container_name: fiware-orion
    depends_on:
      - mongo-db
    networks:
      - default
    expose:
      - "${ORION_PORT}"
    ports:
      - "${ORION_PORT}:${ORION_PORT}" # localhost:1026
    command: -dbhost mongo-db -logLevel DEBUG
    healthcheck:
      test: curl --fail -s http://orion:${ORION_PORT}/version|exit 1
      interval: 5s

```

```
# Tutorial dummies de IoT Sensors via HTTP
tutorial:
  labels:
    org.fiware: 'tutorial'
  image: quay.io/fiware/tutorials.context-provider
  hostname: iot-sensors
  container_name: fiware-tutorial
  networks:
    - default
  expose:
    - '3000'
    - '3001'
  ports:
    - '3000:3000'
    - '3001:3001'
  environment:
    - "MONGO_URL=mongodb://mongo-db:27017"
    - "DEBUG=tutorial:)"
    - "PORT=3000"
    - "WEB_APP_PORT=${TUTORIAL_APP_PORT}"
    - "IOTA_HTTP_HOST=iot-agent"
    - "IOTA_HTTP_PORT=7896"
    - "DUMMY_DEVICES_PORT=3001"
    - "DUMMY_DEVICES_TRANSPORT=HTTP"
    - "DUMMY_DEVICES_API_KEY=4jggokgpepnvsb2uv4s40d59ov"
    - "CONTEXT_BROKER=http://orion:${ORION_PORT}/v2"
    - "OPENWEATHERMAP_KEY_ID=<ADD_YOUR_KEY_ID>"
    - "TWITTER_CONSUMER_KEY=<ADD_YOUR_CONSUMER_KEY>"
    - "TWITTER_CONSUMER_SECRET=<ADD_YOUR_CONSUMER_SECRET>"

# Database
mongo-db:
  labels:
    org.fiware: 'tutorial'
  image: mongo:${MONGO_DB_VERSION}
  hostname: mongo-db
  container_name: db-mongo
  expose:
    - "${MONGO_DB_PORT}"
  ports:
    - "${MONGO_DB_PORT}:${MONGO_DB_PORT}" # localhost:27017
  networks:
    - default
  volumes:
    - mongo-db:/data
  healthcheck:
    test: |
      host=`hostname --ip-address || echo '127.0.0.1'`;
```

```

    mongo --quiet $host/test --eval 'quit(db.runCommand({ ping: 1
        }).ok ? 0 : 2)' && echo 0 || echo 1
    interval: 5s

networks:
  default:
    labels:
      org.fiware: 'tutorial'
    ipam:
      config:
        - subnet: 172.18.1.0/24

volumes:
  mongo-db: ~

```

7.2. Setup: [IoT Agent Ultralight 2.0]

Primeiramente devemos parar o setup inicial :

```

Unset
$ ./services start

```

Deve-se realizar Clone o repositório com o seguinte comando:

```

Unset
$ git clone https://github.com/FIWARE/tutorials.IoT-Agent.git

$ cd tutorials.IoT-Agent

```

Neste ponto é necessário editar o arquivo **docker-compose.yml** IoT-Agent

```

Unset
# Trabalho Prático FIWARE - INTERNET OF THINGS [IoT]
# UFF - Universidade Federal Fluminense
# 2023-2
# Bruno Macena dos Santos
# Elenice Costa
# Rodrigo Oliveira
#
version: "3.8"
services:

```

```
# Orion is the context broker
orion:
  labels:
    org.fiware: 'tutorial'
  image: quay.io/fiware/orion:${ORION_VERSION}
  hostname: orion
  container_name: fiware-orion
  depends_on:
    - mongo-db
  networks:
    - default
  expose:
    - "${ORION_PORT}"
  ports:
    - "${ORION_PORT}:${ORION_PORT}" # localhost:1026
  command: -dbhost mongo-db -logLevel DEBUG
  healthcheck:
    test: curl --fail -s http://orion:${ORION_PORT}/version||exit 1
    interval: 5s
# IoT-Agent is configured for the UltraLight Protocol
iot-agent:
  labels:
    org.fiware: 'tutorial'
  image: quay.io/fiware/iotagent-ul:${ULTRALIGHT_VERSION}
  hostname: iot-agent
  container_name: fiware-iot-agent
  depends_on:
    - mongo-db
  networks:
    - default
  expose:
    - "4041"
    - "7896"
  ports:
    - "4041:4041" # localhost:4041
    - "7896:7896" # localhost:7896
  environment:
    - IOTA_CB_HOST=orion
    - IOTA_CB_PORT=1026
    - IOTA_NORTH_PORT=4041
    - IOTA_REGISTRY_TYPE=mongodb
    - IOTA_LOG_LEVEL=DEBUG
    - IOTA_TIMESTAMP=true
    - IOTA_CB_NGSI_VERSION=v2
    - IOTA_AUTOCAST=true
    - IOTA_MONGO_HOST=mongo-db
    - IOTA_MONGO_PORT=27017
    - IOTA_MONGO_DB=iotagentul
    - IOTA_HTTP_PORT=7896
```

```
- IOTA_PROVIDER_URL=http://iot-agent:4041
- IOTA_DEFAULT_RESOURCE=/iot/d
healthcheck:
  interval: 5s

# Tutorial acts as a series of dummy IoT Sensors over HTTP
tutorial:
  labels:
    org.fiware: 'tutorial'
  image: quay.io/fiware/tutorials.context-provider
  hostname: iot-sensors
  container_name: fiware-tutorial
  depends_on:
    - orion
    - iot-agent
  networks:
    - default
  expose:
    - "3000"
    - "3001"
  ports:
    - "3000:3000" # localhost:3000
    - "3001:3001" # localhost:3001
  environment:
    - "MONGO_URL=mongodb://mongo-db:27017"
    - "DEBUG=tutorial:/*"
    - "PORT=3000"
    - "WEB_APP_PORT=${TUTORIAL_APP_PORT}"
    - "IOTA_HTTP_HOST=iot-agent"
    - "IOTA_HTTP_PORT=7896"
    - "DUMMY_DEVICES_PORT=3001"
    - "DUMMY_DEVICES_TRANSPORT=HTTP"
    - "DUMMY_DEVICES_API_KEY=4jggokgpepnvsb2uv4s40d59ov"
    - "CONTEXT_BROKER=http://orion:${ORION_PORT}/v2"
    - "OPENWEATHERMAP_KEY_ID=<ADD_YOUR_KEY_ID>"
    - "TWITTER_CONSUMER_KEY=<ADD_YOUR_CONSUMER_KEY>"
    - "TWITTER_CONSUMER_SECRET=<ADD_YOUR_CONSUMER_SECRET>"

# Database
mongo-db:
  labels:
    org.fiware: 'tutorial'
  image: mongo:${MONGO_DB_VERSION}
  hostname: mongo-db
  container_name: db-mongo
  expose:
    - "${MONGO_DB_PORT}"
  ports:
    - "${MONGO_DB_PORT}:${MONGO_DB_PORT}" # localhost:27017
```

```
networks:
  - default
volumes:
  - mongo-db:/data
healthcheck:
  test: |
    host=`hostname --ip-address || echo '127.0.0.1'`;
    mongo --quiet $host/test --eval 'quit(db.runCommand({ ping: 1
      }).ok ? 0 : 2)' && echo 0 || echo 1
  interval: 5s
networks:
  default:
    labels:
      org.fiware: 'tutorial'
    ipam:
      config:
        - subnet: 172.18.1.0/24
volumes:
  mongo-db: ~
```

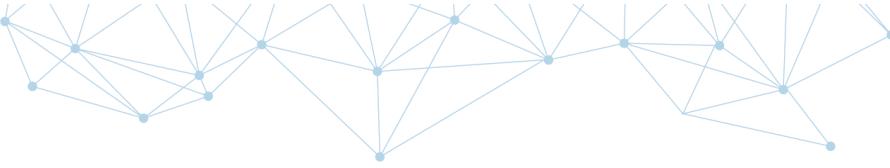
7.3. Criando e inicializando os containers e serviços

```
Unset
$ ./services create
$ ./services start
```

Verificando a integridade do serviço do agente IoT

Pode-se verificar se o Agente IoT está em execução fazendo uma solicitação HTTP para a porta exposta:

```
Unset
$ curl -X GET \ 'http://localhost:4041/iot/about'
```



```
Firefox Web Browser
vboxuser@IoTFiware:~/tutorials.IoT-Sensors/tutorials.IoT-...
vboxuser@IoTFiware:~/tutorials.IoT-Sensors/tutorials.IoT-Agent$ curl -X GET \
http://localhost:4041/iot/about
{"libVersion": "3.4.4", "port": "4041", "baseRoot": "/", "version": "2.4.2"}vboxuser@IoTFiware:~/tutorials.IoT-Sensors/tutorials.IoT-Agent$
```

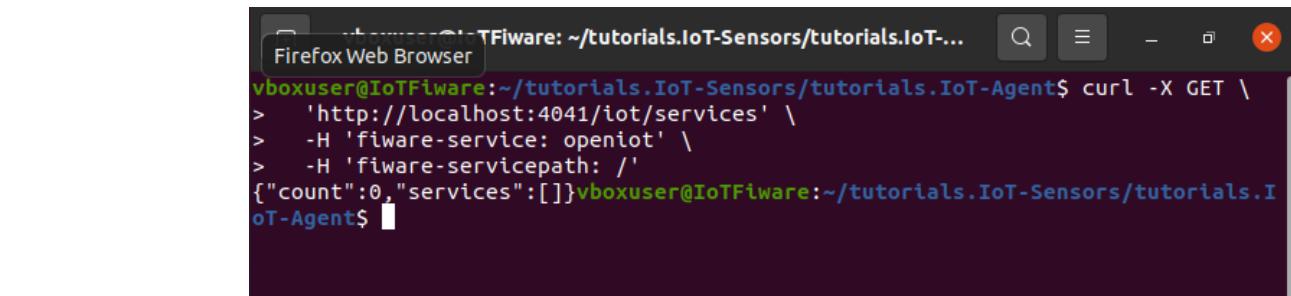
Figura 19 - Agente IoT está em execução. Fonte: autores

Aqui exibimos a saída:

```
Unset
{
  "libVersion": "2.24.0",
  "port": "4041",
  "baseRoot": "/",
  "version": "1.24.0"
}
# Nota: Bem sucedido
```

Para obter a lista completa de dispositivos provisionados pode ser obtida fazendo uma solicitação GET ao endpoint /iot/devices:

```
Unset
$ curl -X GET \
  'http://localhost:4041/iot/devices' \
-H 'fiware-service: openiot' \
-H 'fiware-servicepath: /'
```



```
Firefox Web Browser
vboxuser@IoTFiware:~/tutorials.IoT-Sensors/tutorials.IoT-...
vboxuser@IoTFiware:~/tutorials.IoT-Sensors/tutorials.IoT-Agent$ curl -X GET \
>   'http://localhost:4041/iot/services' \
>   -H 'fiware-service: openiot' \
>   -H 'fiware-servicepath: /'
>{"count":0,"services":[]}{vboxuser@IoTFiware:~/tutorials.IoT-Sensors/tutorials.IoT-Agent$ }
```

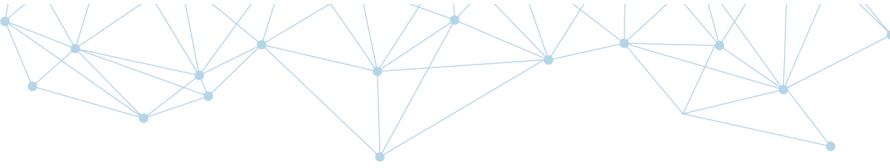
Figura 20 - Dispositivos provisionados. Fonte: autores

Nota: Veja que o total de dispositivos provisionados é 0 :: “Count” = 0!

7.4 Provisionando uma porta inteligente

Provisionar um dispositivo que oferece comandos e medições é apenas uma questão de fazer uma solicitação HTTP POST com atributos e atributos de comando no corpo da solicitação.

```
Unset
$ curl -iX POST \
  'http://localhost:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
  "devices": [
    {
      "device_id": "door001",
      "entity_name": "urn:ngsi-ld:Door:001",
      "entity_type": "Door",
      "protocol": "PDI-IoTA-UltraLight",
      "transport": "HTTP",
      "endpoint": "http://iot-sensors:3001/iot/door001",
      "commands": [
        {"name": "unlock", "type": "command"},
        {"name": "open", "type": "command"},
        {"name": "close", "type": "command"},
        {"name": "lock", "type": "command"}
      ],
      "attributes": [
        {"object_id": "s", "name": "state", "type": "Text"}
      ],
      "static_attributes": [
        {"name": "refStore", "type": "Relationship", "value": "urn:ngsi-ld:Store:001"}
      ]
    }
  ]
}'
```

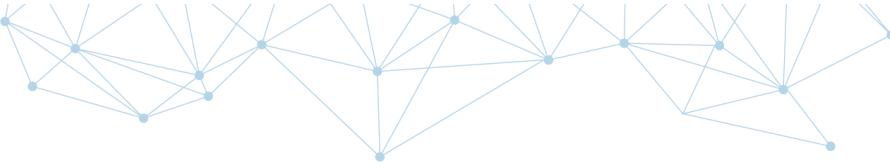


```
vboxuser@IoTFiware: ~/tutorials.IoT-Sensors/tutorials.IoT-...
>     "endpoint": "http://iot-sensors:3001/iot/door001",
>     "commands": [
>         {"name": "unlock", "type": "command"}, 
>         {"name": "open", "type": "command"}, 
>         {"name": "close", "type": "command"}, 
>         {"name": "lock", "type": "command"} 
>     ],
>     "attributes": [
>         {"object_id": "s", "name": "state", "type": "Text"} 
>     ],
>     "static_attributes": [
>         {"name": "refStore", "type": "Relationship", "value": "urn:ngsi-ld:Sto
re:001"} 
>     ]
>   }
> ]
> }
>
HTTP/1.1 201 Created
X-Powered-By: Express
Fiware-Correlator: 2fb028ed-c2ac-4349-91ea-d81840024487
Content-Type: application/json; charset=utf-8
Content-Length: 2
ETag: W/"2-vyGp6PvFo4RvsFtPoIWeCReyIC8"
Date: Mon, 27 Nov 2023 15:04:01 GMT
Connection: keep-alive
Keep-Alive: timeout=5
[]vboxuser@IoTFiware:~/tutorials.IoT-Sensors/tutorials.IoT-Agent$
```

Figura 21 - Solicitação HTTP POST. Fonte: autores

O código HTTP 201, confirma que o comando foi aceito. Para obter a lista completa atualizada de dispositivos provisionados pode ser obtida fazendo uma solicitação GET ao endpoint **/iot/devices**:

```
Unset
$ curl -X GET \
'http://localhost:4041/iot/devices' \
-H 'fiware-service: openiot' \
-H 'fiware-servicepath: /'
```



```
vboxuser@IoTFiware:~/tutorials.IoT-Sensors/tutorials.IoT-Agent$ curl -X GET \
> 'http://localhost:4041/iot/devices' \
> -H 'fiware-service: openiot' \
> -H 'fiware-servicepath: /'
{"count":1,"devices":[{"device_id":"door001","service":"openiot","service_path":"/","entity_name":"urn:ngsi-ld:Door:001","entity_type":"Door","endpoint":"http://iot-sensors:3001/iot/door001","polling":false,"transport":"HTTP","attributes":[{"object_id":"s","name":"state","type":"Text"}],"lazy":[],"commands":[{"object_id":"unlock","name":"unlock","type":"command"}, {"object_id":"open","name":"open","type":"command"}, {"object_id":"close","name":"close","type":"command"}, {"object_id":"lock","name":"lock","type":"command"}],"static_attributes":[{"name":"refStore","type":"Relationship","value":"urn:ngsi-ld:Store:001"}]},{"protocol":"PDI-IoTA-UltraLight","explicitAttrs":false}]}vboxuser@IoTFiware:~/tutorials.IoT-Sensors/tutorials.IoT-Agent$
```

Figura 22 - Lista de dispositivos provisionados. Fonte: autores

Observe o count:1, confirmando um dispositivo registrado e o Device_id: door001. Perceba que dispositivo apresenta alguns objetos e estados como Open, Closed, Lock e Unlock.

8. Testando IoT Agent via Ultralight 2.0

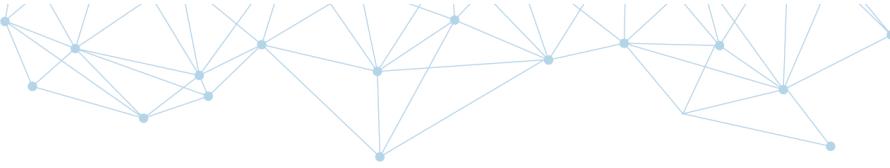
8.1. Abrindo a porta inteligente

Para invocar o comando unlock, o atributo open deve ser atualizado no contexto.

```
Unset
$ curl -iX POST \
--url 'http://localhost:3001/iot/door001' \
--data urn:ngsi-ld:Door:001@unlock
```

Alternativamente para abrir a porta inteligente através do IoT Agent com uma atualização de parâmetro, pode-se também fazê-lo invocando o atributo comando 'Open' no contexto.

```
Unset
$ curl -iX PATCH \
'http://localhost:1026/v2/entities/urn:ngsi-ld:Door:001/attrs' \
-H 'Content-Type: application/json' \
-H 'fiware-service: openiot' \
-H 'fiware-servicepath: /' \
-d '{
  "open": {
    "type" : "command",
    "value" : ""
  }
}'
```



```
vboxuser@IoTFiware:~/tutorials.IoT-Sensors/tutorials.IoT-Agent$ curl -iX POST \
>   --url 'http://localhost:3001/iot/door001' \
>   --data urn:ngsi-ld:Door:001@unlock
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 38
ETag: W/"26-pjGRJPy7HPAI0G8UBRHsAGe6tlQ"
Date: Mon, 27 Nov 2023 17:27:16 GMT
Connection: keep-alive
Keep-Alive: timeout=5

urn:ngsi-ld:Door:001@unlock| unlock OKvboxuser@IoTFiware:~/tutorials.IoT-Sensor
s/tutorials.IoT-Agent$
```

Figura 23 - destravando a porta inteligente. Fonte: autores

O Código http 200, confirmando o sucesso no comando executado anteriormente.



The screenshot shows a web interface titled "IoT Devices (Ultralight Over HTTP)". On the left, there is a sidebar with the title "Devices in Store" and the identifier "urn:ngsi-ld:Store:001". Below this, a list of devices is displayed with their current states:

- door001 s|OPEN
- bell001 s|OFF
- motion001 c|1
- lamp001 s|OFF|1|0

At the bottom of the sidebar, there are two buttons: "Ring Bell" and "Send".

Figura 24 - Porta aberta. Fonte: autores

Página da web do monitor de dispositivos UltraLight, confirmando a porta door001 em estado aberto, portanto destrancada (unlocked). Importante entender que uma porta no estado destrancado (unlocked) pode ainda estar em estado aberta ou fechada (Open ou closed).

8.2. Fechando a porta inteligente

Para invocar o comando close, o atributo open deve ser atualizado no contexto.

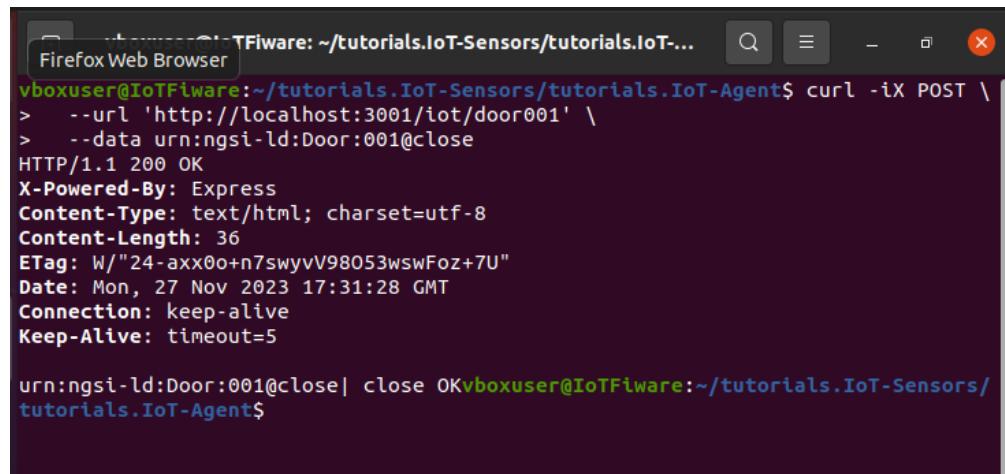
Unset

```
$ curl -iX POST \
--url 'http://localhost:3001/iot/door001' \
--data urn:ngsi-ld:Door:001@close
```

Alternativamente para fechar a porta inteligente através do IoT Agent com uma atualização de parâmetro, pode-se também fazê-lo invocando o atributo comando 'Close' no contexto.

Unset

```
$ curl -iX PATCH \
'http://localhost:1026/v2/entities/urn:ngsi-ld:Door:001/attrs' \
-H 'Content-Type: application/json' \
-H 'fiware-service: openiot' \
-H 'fiware-servicepath: /' \
-d '{
  "close": {
    "type" : "command",
    "value" : ""
  }
}'
```



The screenshot shows a terminal window titled 'Firefox Web Browser' running on an IoTFiware system. The user has run the command:

```
vboxuser@IoTFiware:~/tutorials.IoT-Sensors/tutorials.IoT-Agent$ curl -iX POST \
>   --url 'http://localhost:3001/iot/door001' \
>   --data urn:ngsi-ld:Door:001@close
```

The response from the server is:

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 36
ETag: W/"24-axx0o+n7swyvV98053wswFoz+7U"
Date: Mon, 27 Nov 2023 17:31:28 GMT
Connection: keep-alive
Keep-Alive: timeout=5

urn:ngsi-ld:Door:001@close| close OKvboxuser@IoTFiware:~/tutorials.IoT-Sensors/
tutorials.IoT-Agent$
```

Figura 25 - Porta fechada.Fonte: autores

O código http 200 confirma o sucesso no comando executado anteriormente.



IoT Devices (Ultralight Over HTTP)

Devices in Store urn:ngsi-ld:Store:001

- door001 s|CLOSED
- bell001 s|OFF
- motion001 c|0
- lamp001 s|OFF|1|0

Ring Bell

Figura 26 - Porta fechada. Fonte: autores

Página da web do monitor de dispositivos UltraLight, confirmando a porta door001 em estado fechada (closed), enquanto ainda destrancada.

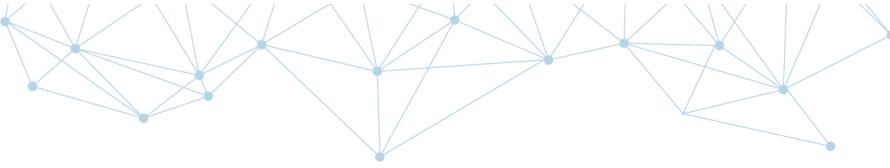
8.3. Trancando a porta inteligente

Para invocar o comando lock, o atributo open deve ser atualizado no contexto.

```
Unset
$ curl -iX POST \
--url 'http://localhost:3001/iot/door001' \
--data urn:ngsi-ld:Door:001@lock
```

Alternativamente para trancar a porta inteligente através do IoT Agent com uma atualização de parâmetro, pode-se também fazê-lo invocando o atributo comando 'lock' no contexto.

```
Unset
$ curl -iX PATCH \
'http://localhost:1026/v2/entities/urn:ngsi-ld:Door:001/attrs' \
-H 'Content-Type: application/json' \
-H 'fiware-service: openiot' \
-H 'fiware-servicepath: /' \
-d '{
  "lock": {
    "type" : "command",
    "value" : ""
  }
}'
```



```
vboxuser@IoTFiware:~/tutorials.IoT-Sensors/tutorials.IoT-Agent$ curl -iX POST \
>   --url 'http://localhost:3001/iot/door001' \
>   --data urn:ngsi-ld:Door:001@lock
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 34
ETag: W/"22-yIdt5yS100+dsGK78Ay/+4y8/PI"
Date: Mon, 27 Nov 2023 17:36:02 GMT
Connection: keep-alive
Keep-Alive: timeout=5

urn:ngsi-ld:Door:001@lock| lock OKvboxuser@IoTFiware:~/tutorials.IoT-Sensors/tutorials.IoT-Agent$ 
vboxuser@IoTFiware:~/tutorials.IoT-Sensors/tutorials.IoT-Agent$ 
```

Figura 27 - Código http 200 confirmando sucesso no comando. Fonte: autores



IoT Devices (Ultralight Over HTTP)

Devices in Store urn:ngsi-ld:Store:001

- door001 s|LOCKED
- bell001 s|OFF
- motion001 c|0
- lamp001 s|OFF|1|0

Ring Bell Send

Figura 28 - Porta fechada. Fonte: autores

Página da web do monitor de dispositivos UltraLight, confirmando a porta door001 em estado trancada (locked).

Observação Geral

Para fins do projeto, o atuador seriam os objetos/comandos lock e unlocked. E quanto ao sensoriamento de abertura de porta, podem se utilizar os objetos/comandos Open Closed.

9. Processamento em Python

Para simular a atuação do smart lock frente a autorizações de abertura para manutenção, foi necessário a criação de dados simulados para representar os acessos autorizados ou tentativas a serem não autorizadas. Um script em python foi elaborado para fornecer os dados a partir dele o processamento com os componentes do Fiware seja para manter a porta trancada (no caso de acessos não autorizados) ou confirmação de abertura em casos de acesso autorizados. Primeiramente, foi realizada a instalação do Python no Ubuntu:

Unset

```
$ sudo apt instal python
```

O script final em python é apresentado a seguir (ler_tag_rfid.py):

```
Python
#!/usr/bin/env python3
import subprocess

def is_authorized(id, file_path):
    with open(file_path, 'r') as file:
        for line in file:
            if id.strip() == line.strip():
                return True
    return False

def unlock_door():
    command = """
    curl -iX POST \
        --url 'http://localhost:3001/iot/door001' \
        --data urn:ngsi-ld:Door:001@unlock
    """
    subprocess.run(command, shell=True)

def main():
    id_to_check = input("Por favor, insira o ID do colaborador: ")
    file_path = "autorizados.txt"

    if is_authorized(id_to_check, file_path):
        unlock_door()
        print("Porta destrancada.")
    else:
        print("Acesso negado.")

if __name__ == "__main__":
    main()
```

```
main()
```

O arquivo de autorizados ("autorizados.txt"), ficou assim.

```
Unset
123456
123457
```

Ambos ficam no mesmo diretório, e foi dada permissão de execução no arquivo Python:

```
Unset
$ chmod +x ler_tag_rfid.py
```

Execução do script com fornecimento de um ID autorizado na base:

```
vboxuser@IoTFiware:~/tutorials.IoT-Sensors/tutorials.IoT-Agent$ ./ler_tag_rfid.py
Por favor, insira o ID do colaborador: 123456
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 38
ETag: W/"26-pjGRJPy7HPAI0G8UBRHsAGe6tlQ"
Date: Mon, 27 Nov 2023 19:06:48 GMT
Connection: keep-alive
Keep-Alive: timeout=5

urn:ngsi-ld:Door:001@unlock| unlock OKPorta destrancada.
vboxuser@IoTFiware:~/tutorials.IoT-Sensors/tutorials.IoT-Agent$
```

Figura 29 - Acesso autorizado. Fonte: autores



IoT Devices (Ultralight Over HTTP)

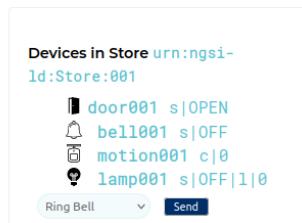


Figura 30 - Porta aberta. Fonte: autores



Página da web do monitor de dispositivos UltraLight, confirmando a porta door001 em estado destrancada/aberta (unlocked: Open).

Execução do script com fornecimento de um ID não autorizado na base:

```
vboxuser@IoTFiware:~/tutorials.IoT-Sensors/tutorials.IoT-Agent$ ./ler_tag_rfid.py
Por favor, insira o ID do colaborador: 123
Acesso negado.
vboxuser@IoTFiware:~/tutorials.IoT-Sensors/tutorials.IoT-Agent$
```

Figura 31 - Acesso negado. Fonte: autores



IoT Devices (Ultralight Over HTTP)



Figura 32 - Porta trancada. Fonte: autores

Página da web do monitor de dispositivos UltraLight, confirmando a porta door001 em estado trancada (unlocked). Devido ao acesso negado.

Referências

1. FIWARE. Installation Guide (Ubuntu 20.04.1). Disponível em:
<https://github.com/FIWARE/context.Orion-LD/blob/develop/doc/manuals-l1/installation-guide-ubuntu-20.04.1.md>. Acesso em: 29 nov. 2023.
2. FIWARE. Tutorials: IoT Sensors. Disponível em: <https://github.com/fiware/tutorials.iot-sensors>. Acesso em: 29 nov. 2023.
3. Canonical Ltd. Ubuntu 20.04 LTS (Focal Fossa). Disponível em:
<https://releases.ubuntu.com/20.04/>. Acesso em: 29 nov. 2023.
4. FIWARE. IoT Agent UL. Disponível em: <https://fiware-iotagent-ul.readthedocs.io/>. Acesso em: 29 nov. 2023.
5. FIWARE. Tutorials: IoT Sensors (GitHub Repository). Disponível em:
<https://github.com/FIWARE/tutorials.IoT-Sensors.git>. Acesso em: 29 nov. 2023.
6. Docker, Inc. Install Docker Desktop on Ubuntu. Disponível em:
<https://docs.docker.com/desktop/install/ubuntu/>. Acesso em: 29 nov. 2023.
7. Docker, Inc. Install Docker Engine on Ubuntu. Disponível em:
<https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository>. Acesso em: 29 nov. 2023.
8. FIWARE. IoT Agent. Disponível em:
<https://fiware-tutorials.readthedocs.io/en/latest/iot-agent.html>. Acesso em: 29 nov. 2023.
9. HVEX. Segurança Cibernética no Setor Elétrico: Medidores Inteligentes. Disponível em:
<https://conteudo.hvex.com.br/tecnologia/seguranca-cibernetica-no-setor-eletrico-medidores-inteligentes/>. Acesso em: 29 nov. 2023.