



**Universidade do Minho**

UNIVERSIDADE DO MINHO  
**Mestrado em Engenharia de Telecomunicações e  
Informática**

---

# **Relatório Final**

UC de Emulação e Simulação de Redes de Telecomunicações

---

*Docentes:*

Prof. Fábio Raul Costa Gonçalves  
Prof. José Augusto Afonso  
Prof. Bruno Daniel Mestre Viana Ribeiro

## **Autores:**

- Carlos Daniel Machado Salazar Dias - PG53718
- Carlos Daniel Silva Félix - PG54719
- Luís Filipe Oliveira Azevedo - PG54008

Ano Letivo 2023/2024

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>4</b>
1.1	Fase A: Desenvolvimento da Infra-estrutura de Rede Utilizando GNS3 . . .	4
1.2	Fase B: Implementação da Rede de Sensores Utilizando CupCarbon . . . .	4
1.3	Fase C: Desenvolvimento da Plataforma de IoT . . . . .	4
<b>2</b>	<b>Arquitetura</b>	<b>5</b>
2.1	Rede de Routers . . . . .	5
2.2	Simulação de Sensores com CupCarbon . . . . .	5
2.3	Brokers de IoT e NGINX . . . . .	5
2.4	Base de Dados Centralizada . . . . .	5
2.5	Serviços de DNS . . . . .	5
<b>3</b>	<b>Tema do Projeto</b>	<b>7</b>
<b>4</b>	<b>Sensores e atuadores utilizados</b>	<b>8</b>
<b>5</b>	<b>CupCarbon: Ferramenta de Simulação para IoT</b>	<b>9</b>
5.1	Modelação de Rede de Sensores com CupCarbon . . . . .	9
5.1.1	Composição e Configuração da Rede . . . . .	9
5.1.2	Integração de Eventos Analógicos . . . . .	9
5.2	Código Sensores individuais . . . . .	10
5.3	Código Atuadores . . . . .	11
5.4	Analog Event . . . . .	12
5.5	Recetor . . . . .	13
5.6	Publisher . . . . .	14
5.6.1	Configurações do Broker MQTT: . . . . .	14
5.6.2	Função para conectar ao MQTT Broker: . . . . .	14
5.6.3	Função para publicar mensagens no broker MQTT: . . . . .	14
5.6.4	Função para ler ficheiro e enviar para o broker MQTT: . . . . .	15
<b>6</b>	<b>GNS3</b>	<b>16</b>
6.1	Topologia . . . . .	16
6.2	Configuração dos routers . . . . .	17
6.3	Protocolo de Encaminhamento . . . . .	18
6.4	Servidor DNS . . . . .	19
6.4.1	Configuração do Named.conf.local . . . . .	20
6.4.2	Configuração do Ficheiro emulacao.pt.db . . . . .	21
6.5	Balanceador de Carga . . . . .	22
6.6	Gestão de Bases de Dados . . . . .	24
6.7	Configuração dos Brokers MQTT . . . . .	25
6.8	Interface Gráfica do Utilizador . . . . .	28
<b>7</b>	<b>Testes</b>	<b>29</b>
7.1	Rede sensores . . . . .	29
7.2	Avaliação do Balanceamento de Carga no Ambiente GNS3 . . . . .	30
7.3	Server HTTP e busca dos valores na base de dados. . . . .	31
<b>8</b>	<b>Tecnologias/Recursos:</b>	<b>32</b>
8.1	Hardware: . . . . .	32

8.2 Software: . . . . .	32
<b>9 Competências Desenvolvidas:</b>	<b>33</b>
<b>10 Conclusão</b>	<b>34</b>

## Lista de Figuras

1	Arquitetura do Sistema . . . . .	6
2	Representação esquemática da Rede de Sensores modelada no CupCarbon. . . . .	9
3	Conteúdo do ficheiro evt de temperatura. . . . .	12
4	Topologia usada no GNS3. . . . .	16
5	Diagrama das Bases de Dados MongoDB para Sensores e Autenticação. . . . .	24
6	Brokers na Cloud 1 . . . . .	25
7	Brokers na Cloud 2 . . . . .	25
8	Gráficos representativos dos parâmetros ambientais medidos pelos sensores. . . . .	28
9	Detalhe das últimas amostras de dados e alertas emitidos pelo sistema. . . . .	28
10	Funcionamento da recolha amostras e envio para os atuadores e no central. . . . .	29
11	Algumas amostras registadas pelo recetor. . . . .	29
12	Subscriber em espera . . . . .	30
13	Recepção de dados no Subscriber 0 . . . . .	30
14	Recepção de dados no Subscriber 1 . . . . .	30
15	Server HTTP iniciou sem qualquer problema ao conectar com a base de dados. . . . .	31
16	Servidor recuperou amostras. . . . .	31
17	Gráficos com as amostras recuperadas. . . . .	31

## Lista de Tabelas

1	Descrição de Sensores Urbanos . . . . .	8
2	Limites de Seguros dos Dados Coletados . . . . .	8
3	Ferramentas de hardware. . . . .	32
4	Tabela com Logos e Descrições. . . . .	32
5	Competências Adquiridas e a Contribuição das Disciplinas . . . . .	33

# 1 Introdução

A Internet das Coisas (IoT), representa uma inovação tecnológica significativa, criou uma ponte entre o mundo físico e o virtual, permitindo a interconexão automatizada de uma vasta gama de objetos, pessoas e locais com o ambiente digital. Neste projeto, sensores e atuadores desempenharam um papel crucial, recolhendo dados ambientais como temperatura, humidade, som e qualidade do ar, e efectuando ajustes automáticos com base nesses dados. Este fluxo contínuo de informação gerou um volume significativo de dados, exigindo uma arquitetura de suporte avançada para uma análise eficiente.

O projeto centrou-se no desenvolvimento de um protótipo para um sistema de IoT que integra uma plataforma na nuvem com uma rede de dispositivos sensores e de atuadores. Este sistema facilitou a gestão e armazenamento de dados dos sensores na nuvem, além de proporcionar uma administração eficaz dos utilizadores. Adicionalmente, implementou-se uma rede de sensores robusta que permitiu a recolha e transmissão de dados para a nuvem.

Durante as etapas do projeto, o grupo desenvolveu competências em áreas como simulação, emulação, aplicações de rede e IoT, bem como na criação e gestão de infra-estruturas na nuvem, incluindo a configuração de servidores DNS. O projeto dividiu-se em três fases principais, cada uma com objectivos específicos bem definidos e agora concluídos.

## 1.1 Fase A: Desenvolvimento da Infra-estrutura de Rede Utilizando GNS3

Na fase inicial, o foco foi na concepção e implementação da infra-estrutura de rede através do software GNS3. A rede projetada facilitou a comunicação eficiente entre os nós, proporcionando acesso à Internet e à plataforma IoT. Esta fase permitiu ao grupo explorar a configuração de redes virtuais, compreendendo os princípios fundamentais de comunicação e conectividade.

## 1.2 Fase B: Implementação da Rede de Sensores Utilizando CupCarbon

Na segunda fase, o grupo utilizou a ferramenta CupCarbon para implementar redes de sensores focadas no monitoramento e controlo de ambientes específicos. Estes sensores, estrategicamente posicionados, enviaram dados importantes para a plataforma IoT. Esta fase também abordou a integração entre a rede de sensores desenvolvida e a infra-estrutura de rede previamente estabelecida, destacando a importância da interoperabilidade no contexto da IoT.

## 1.3 Fase C: Desenvolvimento da Plataforma de IoT

Na fase final, o grupo dedicou-se ao desenvolvimento e aperfeiçoamento da plataforma de IoT. Esta plataforma foi desenhada para oferecer funcionalidades vitais, como o recebimento e armazenamento eficaz de dados dos sensores, gestão avançada de utilizadores, visualização intuitiva de dados e sistemas robustos de autenticação. Implementaram-se múltiplos servidores replicados, um serviço de balanceamento de carga e protocolos de aplicação como HTTP e MQTT, assegurando a escalabilidade e segurança da plataforma.

## 2 Arquitetura

A arquitetura final do sistema IoT destaca-se pela interconexão eficaz entre a rede de routers e a infraestrutura cloud. Esta estrutura otimizada enfatiza a gestão eficiente do fluxo de dados, permitindo a recolha estratégica de informações e o seu armazenamento e processamento numa robusta plataforma cloud. Esta arquitetura integrada garante segurança e eficiência operacional, oferecendo conectividade fluída e adaptabilidade para uma vasta gama de aplicações.

### 2.1 Rede de Routers

A base da arquitetura é formada por uma rede interligada de routers, que gerem o tráfego de dados internos e asseguram a ligação ao servidor central. Este servidor atua como um gateway, fornecendo a conectividade necessária com a Internet e com outros componentes da rede.

### 2.2 Simulação de Sensores com CupCarbon

Com o recurso ao CupCarbon, uma ferramenta de simulação para redes de sensores, a arquitetura permite uma modelação detalhada do comportamento dos sensores. Esta fase é vital para refinar os processos de recolha e transmissão de dados antes da implementação física dos sensores.

### 2.3 Brokers de IoT e NGINX

Os brokers de IoT, situados na cloud, são cruciais para a gestão das mensagens entre os dispositivos e a plataforma central. O NGINX é utilizado como um balanceador de carga, otimizando a distribuição das requisições aos brokers, o que melhora a disponibilidade e escalabilidade do sistema.

### 2.4 Base de Dados Centralizada

A arquitetura inclui ainda uma base de dados centralizada, que armazena e gere o grande volume de dados gerados. A conexão entre os brokers e a base de dados é essencial para assegurar que os dados recolhidos sejam processados e armazenados de forma segura e acessível.

### 2.5 Serviços de DNS

Integrado na arquitetura, um servidor DNS facilita a resolução de nomes de domínio, permitindo que dispositivos e serviços na Internet encontrem e comuniquem eficientemente com os brokers e a base de dados.

A figura abaixo ilustra de forma clara a arquitetura detalhada do sistema, realçando a interligação entre os componentes e o fluxo de dados desde a sua recolha até ao armazenamento.

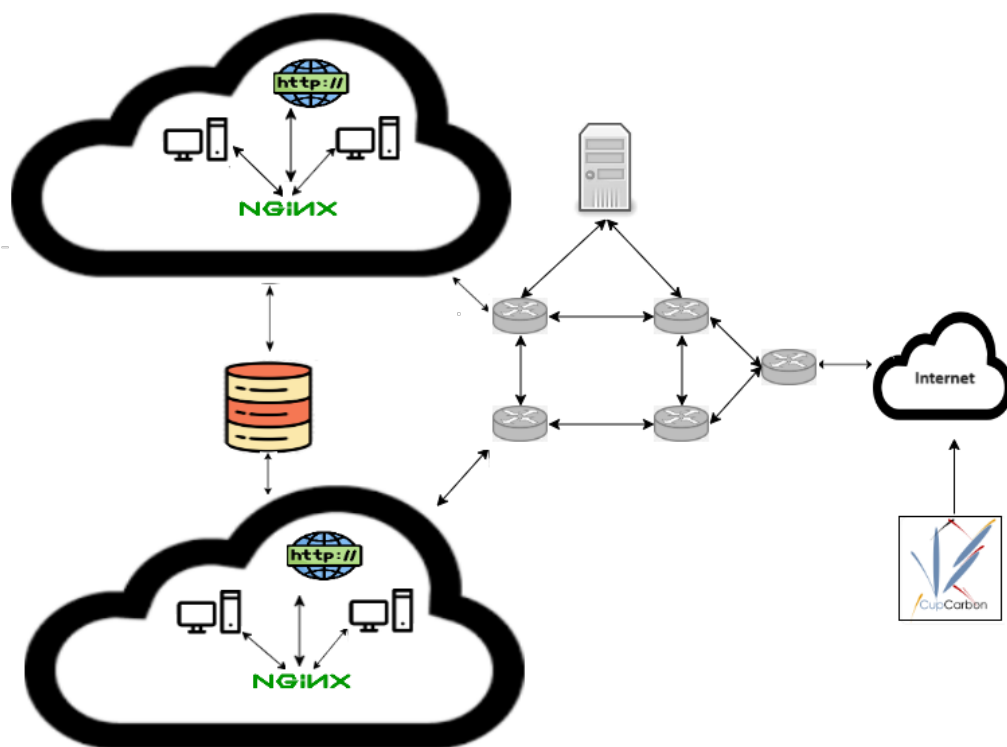


Figura 1: Arquitetura do Sistema

### 3 Tema do Projeto

O foco do projeto do grupo, intitulado "Saúde Inteligente em Ambientes Urbanos", concentra-se na monitorização de diversos parâmetros relacionados à saúde em ambientes urbanos. Dentre esses parâmetros, destacam-se a qualidade do ar, poluição sonora, temperatura e humidade.

A importância dessa monitorização reside na capacidade de fornecer dados cruciais para avaliar e compreender os impactos desses fatores na saúde humana em cenários urbanos.

Esses dados não apenas possibilitam uma compreensão mais profunda do ambiente urbano, mas também fornecem informações valiosas para a implementação de medidas preventivas e corretivas. Dessa forma, o projeto busca não apenas monitorar, mas também contribuir para a promoção da saúde e bem-estar em comunidades urbanas.



## 4 Sensores e atuadores utilizados

Para coletar uma variedade de dados necessários para o projeto, é essencial usar diversos sensores. De acordo com o tema do projeto, empregamos os seguintes sensores:

Sensor	Descrição
Sensores de Qualidade do Ar	Medem a concentração de poluentes atmosféricos, como o dióxido de carbono, contribuindo para a avaliação da qualidade do ar.
Sensores de Som	Detetam níveis de ruído no ambiente urbano, permitindo o monitoramento da poluição sonora e seus impactos na saúde auditiva.
Sensores de Temperatura	Monitoram a temperatura ambiente, fornecendo dados essenciais para análises climáticas e detecção de variações térmicas.
Sensores de Humidade	Avaliam a humidade do ar, possibilitando análises meteorológicas detalhadas e contribuindo para compreensão de condições ambientais.

Tabela 1: Descrição de Sensores Urbanos

No contexto do projeto, os atuadores desempenham um papel fundamental, proporcionando uma resposta imediata aos dados coletados. Esses dispositivos entram em ação quando detectam valores considerados excessivamente altos ou baixos, permitindo uma intervenção rápida e eficaz em situações críticas.

Tipo de Dado Medido	Limite Seguro
Qualidade do Ar	600-1100 ppm
Som	53-62dB
Humidade	0-100%
Temperatura	5-40°C

Tabela 2: Limites de Seguros dos Dados Coletados

## 5 CupCarbon: Ferramenta de Simulação para IoT

O CupCarbon destaca-se como uma plataforma de simulação avançada, concebida para modelar e ensaiar redes de Internet das Coisas e redes de sensores sem fios. A sua interface permite uma visualização intuitiva das topologias de rede em ambientes urbanos e naturais.

### 5.1 Modelação de Rede de Sensores com CupCarbon

A aplicabilidade do CupCarbon é demonstrada através da modelação de uma rede de sensores na zona de Guimarães, mais precisamente nas imediações da Universidade do Minho. O objectivo desta simulação é a análise da comunicação e processamento de dados numa infraestrutura IoT característica.

#### 5.1.1 Composição e Configuração da Rede

Cada rede concebida é composta por 9 Nós de Sensores, distribuídos estrategicamente para maximizar a cobertura e eficiência da comunicação. Cada Nó Sensor foi classificado consoante a sua função na rede: - **4 Sensores Individuais:** Responsáveis pela recolha de dados ambientais. - **4 Atuadores:** Nós que executam ações concretas no ambiente, reagindo a comandos, como por exemplo, ajustar as condições de iluminação ou operar dispositivos mecânicos. - **1 Recetor Central:** Aglomera dados dos sensores e guarda-os em um arquivo.

#### 5.1.2 Integração de Eventos Analógicos

A cada Nó Sensor foi associado um Evento Analógico, que simula uma variável ambiental monitorizada pelo sensor — tal como temperatura, humidade ou níveis de luminosidade. Estes eventos são essenciais para o estudo das dinâmicas de resposta da rede, como a ativação de atuadores em consequência de alterações detectadas pelos sensores.



Figura 2: Representação esquemática da Rede de Sensores modelada no CupCarbon.

## 5.2 Código Sensores individuais

O código apresentado é destinado ao processamento de leituras de sensores, com um exemplo específico voltado para um sensor de temperatura. No entanto, as estruturas e lógicas do código são semelhantes para outros sensores.

```
set num 0
loop
  areadsensor v
  rdata v a b valor
  if (num < 2)
    set valor 0
    data p "Temperatura: " valor
    print p
    send p 10
    send p 27
    inc num
  else
    data p "Temperatura: " valor
    print p
    send p 10
    send p 30
    time tempo
    cprint 11 10 tempo p
  end
  delay 1000
```

O código acima começa por inicializar a variável "num" com o valor zero e depois inicia um loop infinito. Em seguida lê uma linha do analog event e guarda o resultado na variável "v" e separa o resultado anterior em duas variáveis "a" e "b", sendo que a b corresponde ao valor da temperatura.

NO if (num<2), recorreremos a esta expressão pois os primeiros 2 valores lidos usando areadsensor v, são sempre null e de modo a evitar isso usamos isto.

Por outro lado, quando if anterior passa a false, preparamos a mensagem a enviar "p", contendo informação do tipo de dado coletado e o respectivo valor e enviamos essa informação para o receptor e o respectivo atuador.

Após a execução do bloco condicional, há um atraso de 1000 milissegundos (1 segundo) antes de reiniciar o loop.

### 5.3 Código Atuadores

A implementação dos códigos para os atuadores é bastante direta. Eles recebem os dados enviados pelos sensores individuais e verificam se os valores estão dentro dos limites seguros predefinidos. Em caso de violação desses limites, o atuador registra um alerta, marcando-o como "1" (mark 1), e escreve uma mensagem no arquivo correspondente. Essa mensagem inclui informações cruciais, como a área específica e o tipo de dados que excedeu o limite seguro.

No caso de cumprir os limites seguros, apenas faz mark 0.

```
loop
receive y
rdata y string_Temp Temp

if ((Temp>40) || (Temp < 5))
    mark 1
    data envio "Area 1-> Alerta Temperatura:" Temp
    printfile envio
    time tempo
    cprint 11 30 tempo envio
else
    mark 0
end
delay 1000
```

## 5.4 Analog Event

O evento analógico está vinculado a um arquivo evt, o qual contém valores representativos dos sensores. Esses valores foram gerados por meio de uma função Python que assegura uma variação mais gradual, em contraste com a natureza abrupta associada ao uso da função random. Essa escolha visa criar um cenário mais realista, refletindo de maneira mais fiel as condições reais de medição de dados de sensores.

Cabe destacar que o arquivo evt é composto por um total de 1000 amostras. Essas amostras foram cuidadosamente geradas para simular a variação dos dados provenientes dos sensores. A figura abaixo apresenta o conteúdo do ficheiro evt na área 1 do sensor de temperatura.

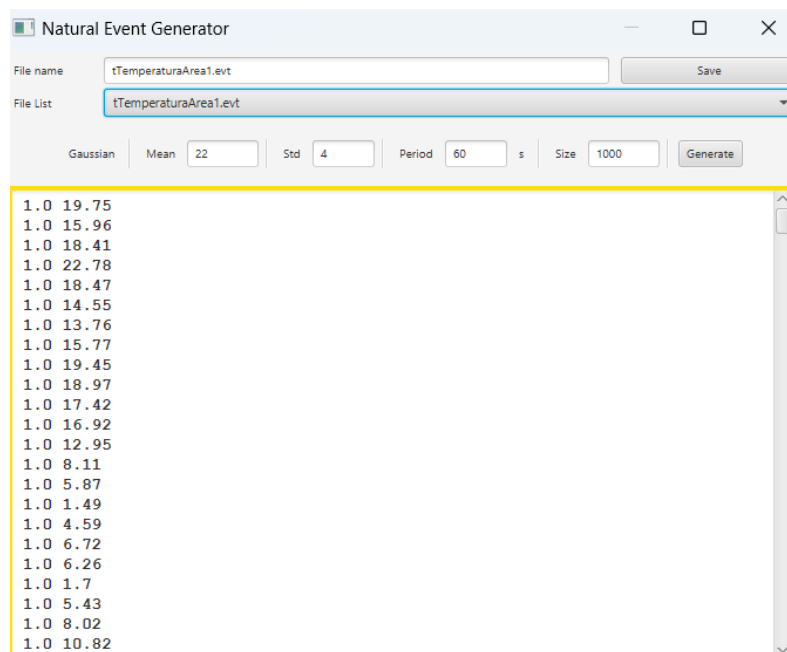


Figura 3: Conteúdo do ficheiro evt de temperatura.

## 5.5 Recetor

No recetor, sua função principal é receber mensagens provenientes de diversos sensores, processá-las e, finalmente, armazenar os resultados em um arquivo "printfile envio", conforme ilustrado no código abaixo:

```
set a 0
loop
if (a>2)
  receive y
  rdata y string_som som
  receive x
  rdata x string_co2 co2
  receive z
  rdata z string_humidade humidade
  receive w
  rdata w string_temperatura temperatura
  data envio "Area 1:" string_temperatura temperatura string_som som string_co2
    ↪ co2 string_humidade humidade
  print envio
  printfile envio
else
  inc a
end
delay 1000
```

Assim como nos sensores, o recetor também incorpora a condição  $\text{if } (a > 2)$ . Essa condição é aplicada para assegurar que o processamento não ocorra nas duas primeiras mensagens recebidas, uma vez que esses dados inicialmente capturados tendem a ser nulos e incorretos.

## 5.6 Publisher

O Publisher tem a responsabilidade de ler os dados armazenados nos arquivos correspondentes (dados e alertas da área 1, dados e alertas da área 2) e, em seguida, transmiti-los através do protocolo MQTT para o load balancer.

### 5.6.1 Configurações do Broker MQTT:

```
broker = "emulacao.pt"
port = 1883
topic = "meu/topico"
```

### 5.6.2 Função para conectar ao MQTT Broker:

```
def connect_mqtt() -> mqtt_client.Client:
    def on_connect(client, userdata, flags, rc):
        if rc == 0:
            print("Connected to MQTT Broker via Load Balancer!")
        else:
            print("Failed to connect, return code %d\n", rc)

    client = mqtt_client.Client()
    client.on_connect = on_connect
    client.connect(broker_lb, port_lb)
    return client
```

### 5.6.3 Função para publicar mensagens no broker MQTT:

```
def publish(message):
    client = connect_mqtt() # Conectar ao broker via Load Balancer
    client.loop_start()
    result = client.publish(topic, message)
    status = result[0]
    if status == 0:
        print(f"Sent `{message}` to topic `{topic}` via Load Balancer")
    else:
        print(f"Failed to send message to topic {topic} via Load Balancer")
    client.loop_stop()
    client.disconnect() # Desconectar do broker após publicar
```

#### 5.6.4 Função para ler ficheiro e enviar para o broker MQTT:

```
def readFileAndSend(file_path):
    last_line = None
    while True:
        if os.path.exists(file_path):
            with open(file_path, 'r') as file:
                lines = file.readlines()

            for line in lines:
                line = line.replace("#", " ").strip()
                if line != last_line: # Publicar apenas se a linha for diferente
                                     ↪ da última publicada
                    print(line)
                    publish(line)
                    time.sleep(1)
                last_line = line # Atualizar a última linha publicada

            for line in tailer.follow(open(file_path)):
                line = line.replace("#", " ").strip()
                if line != last_line:
                    print(line)
                    publish(line)
                    time.sleep(1)
                last_line = line
        else:
            time.sleep(1)
```

É fundamental destacar que essa função requer um caminho de arquivo, que consiste essencialmente em um array de strings. Nesse array, são especificados os PATH para vários arquivos de leitura. Além disso, é crucial ressaltar que cada arquivo de leitura é processado em uma thread separada.



## 6 GNS3

O GNS3, ou Graphical Network Simulator 3, é uma plataforma de simulação de redes que permite aos utilizadores modelar, emular e testar configurações de redes de computadores. Sua vantagem distintiva reside na capacidade de simular redes em ambientes virtuais antes da implementação em hardware real, proporcionando aos profissionais de redes uma ferramenta valiosa para testar configurações e soluções.

## 6.1 Topologia

A topologia utilizada é a que está apresentada na seguinte imagem:

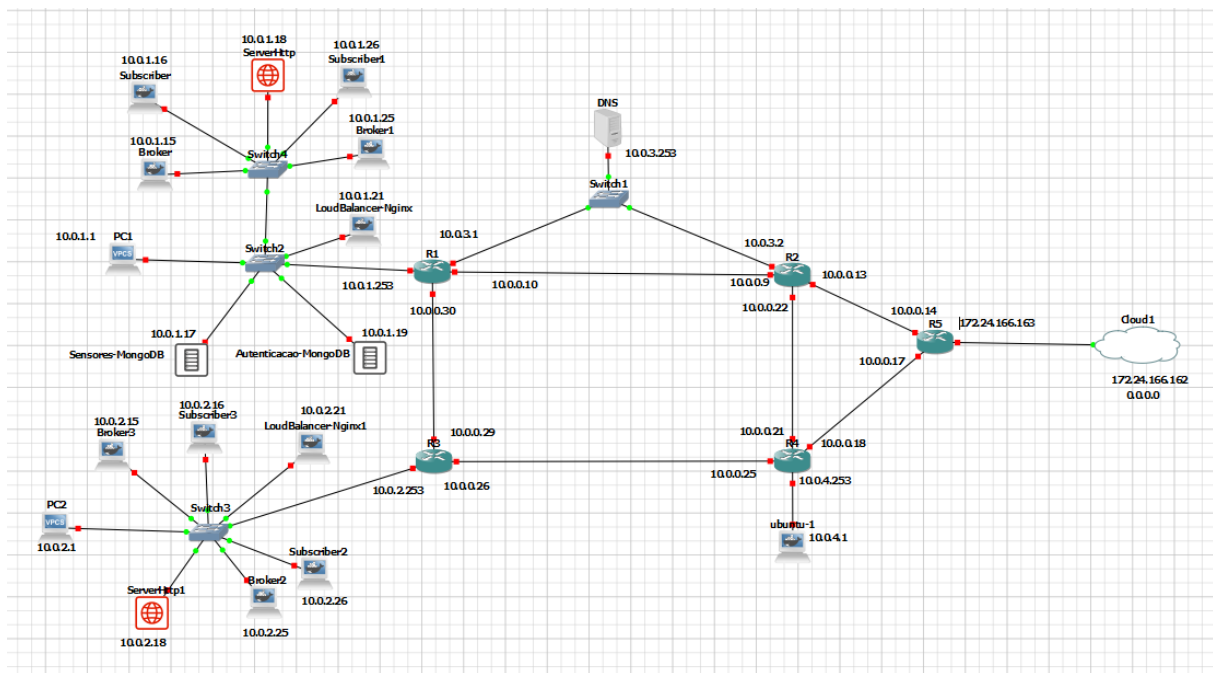


Figura 4: Topologia usada no GNS3.

## 6.2 Configuração dos routers

A atribuição de endereços foi realizada manualmente nos routers. Abaixo estão os comandos correspondentes utilizados no GNS3 para essa configuração, correspondente ao R1, sendo que para os restantes routers é feito de forma semelhante:

```
interface FastEthernet0/0
 ip address 10.0.1.253 255.255.255.0
 duplex auto
 speed auto
!
interface FastEthernet0/1
 ip address 10.0.3.1 255.255.255.0
 duplex auto
 speed auto
!
interface FastEthernet1/0
 ip address 10.0.0.30 255.255.255.252
 duplex auto
 speed auto
!
interface FastEthernet2/0
 ip address 10.0.0.10 255.255.255.252
 duplex auto
 speed auto
!
```

## 6.3 Protocolo de Encaminhamento

O protocolo utilizado para o encaminhamento foi o OSPF, Open Shortest Path First, é um protocolo dinâmico de encaminhamento projetado para facilitar a troca eficiente de informações entre routers em uma rede. Em essência, o OSPF determina o caminho mais eficiente para alcançar destinos na rede, para isso recorre a métricas como largura de banda para otimizar o encaminhamento de dados.

Para o bom funcionamento deste protocolo foi utilizado a seguinte configuração:

```
router ospf 1
network 10.0.0.0 0.0.0.3 area 0
network 10.0.0.8 0.0.0.3 area 0
network 10.0.0.28 0.0.0.3 area 0
network 10.0.1.0 0.0.0.255 area 0
!
```

### Comando utilizados:

- router ospf 1: ativa o protocolo OSPF.
- network <rede> <máscara> area <número da área>: Este comando é utilizado para informar ao OSPF quais redes estão associadas a ele e em qual área estão localizadas.

## 6.4 Servidor DNS

O Servidor DNS (Domain Name System) desempenha um papel crucial na navegação na internet, traduzindo nomes de domínio legíveis por humanos em endereços IP compreendidos por máquinas. Ao associar um nome de domínio a um endereço IP, o DNS facilita a localização de recursos online, como websites e serviços.

Configuração usada no `named.conf.options`:

```
options {
    directory "/var/cache/bind";

    recursion yes;
    allow-recursion { any; };

    listen-on port 53 { any; };
    allow-query { any; };

    forwarders {
        8.8.8.8;
        8.8.4.4;
    };

    auth-nxdomain no;      # conform to RFC1035
};
```

No arquivo `named.conf.options`, definimos o diretório para armazenar dados de cache, ativamos a recursão para obter informações externas, permitimos que qualquer dispositivo na rede faça consultas recursivas, configuramos o servidor para escutar em todas as interfaces na porta 53 e permitimos consultas de qualquer cliente. Além disso, encaminhamos consultas não locais para os servidores DNS do Google e desativamos autenticação para respostas NXDOMAIN.

### 6.4.1 Configuração do Named.conf.local

A configuração do servidor DNS BIND utiliza o arquivo ‘named.conf.local’ para definir zonas DNS. A seguir, apresenta-se a configuração essencial para o estabelecimento de zonas autoritárias necessárias para a resolução de nomes dentro de uma rede simulada.

```
zone "cloud1.local" {  
    type master;  
    file "/etc/bind/cloud1.local.db";  
};  
  
zone "cloud2.local" {  
    type master;  
    file "/etc/bind/cloud2.local.db";  
};  
  
zone "cloud1" {  
    type master;  
    file "/etc/bind/cloud1.db";  
};  
  
zone "cloud2" {  
    type master;  
    file "/etc/bind/cloud2.db";  
};  
  
zone "emulacao.pt" {  
    type master;  
    file "/etc/bind/emulacao.pt.db";  
};
```

Cada ‘zone’ declara um domínio para o qual este servidor DNS atua como autoridade máxima (‘type master’). Os arquivos especificados em ‘file’ contêm os registros DNS para cada zona, determinando como os nomes são resolvidos para endereços IP. A zona ‘emulacao.pt’ é configurada para ser gerida localmente, permitindo a resolução de nomes para serviços dentro da infraestrutura de emulação.

### 6.4.2 Configuração do Ficheiro emulacao.pt.db

A configuração do domínio ‘emulacao.pt’ é especificada no ficheiro de zona ‘emulacao.pt.db’, que contém os registos essenciais para a resolução de nomes DNS. A seguir, encontra-se um excerto desta configuração.

```
$TTL      604800
@         IN      SOA      ns.emulacao.pt. admin.emulacao.pt. (
                                2           ; Serial
                                604800      ; Refresh
                                86400       ; Retry
                                2419200    ; Expire
                                604800 )    ; Negative Cache TTL
;
@         IN      NS       ns.emulacao.pt.
ns        IN      A        10.0.3.253
@         IN      A        10.0.1.21
@         IN      A        10.0.2.21
```

O registo SOA inicializa a zona com parâmetros essenciais, como o número de série e tempos de expiração de cache. O NS declara o servidor de nomes autoritativo, e os registos A associam o nome do servidor de nomes e o domínio à sua representação numérica IP correspondente. Esta configuração assegura que as requisições para ‘emulacao.pt’ sejam dirigidas para os endereços IP apropriados.

## 6.5 Balanceador de Carga

O Balanceador de Carga é um componente crucial em uma arquitetura de rede moderna. Sua função é distribuir o tráfego de rede de maneira uniforme entre vários servidores, promovendo um desempenho otimizado, alta disponibilidade e escalabilidade. Funcionando como um ponto de acesso central para os utilizadores, o balanceador de carga redireciona as solicitações para os servidores de backend, prevenindo a sobrecarga em um único ponto do sistema.

```
# Configuração NGINX para Balanceamento de Carga
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log notice;
pid /var/run/nginx.pid;

events {
    worker_connections 1024; # Número máximo de conexões simultâneas por processo
}

# Configurações HTTP
http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
        '$status $body_bytes_sent "$http_referer" '
        '"$http_user_agent" "$http_x_forwarded_for"';

    upstream http_servers {
        server 10.0.1.18; # Servidor HTTP 1
    }

    server {
        listen 80; # Porta padrão para HTTP
        location / {
            proxy_pass http://http_servers; # Proxy para o grupo http_servers
        }
    }
}

stream {
    upstream mqtt_brokers {
        server 10.0.1.15:1883; # Broker MQTT 1
        server 10.0.1.25:1883; # Broker MQTT 2
    }

    server {
        listen 1883; # Porta padrão MQTT
        proxy_pass mqtt_brokers;
    }
}
```

Este código configura o NGINX para atuar como um balanceador de carga para dois tipos de tráfego: HTTP e MQTT. Para o tráfego HTTP, o servidor está configurado para escutar na porta 80 e distribuir as solicitações para o servidor HTTP no endereço IP 10.0.1.18. O método de balanceamento de carga *round-robin* é aplicado para assegurar uma distribuição equitativa do tráfego pelos brokers MQTT especificados.

- Servidor HTTP: 10.0.1.18:80
- Brokers MQTT:
  - 10.0.1.15:1883
  - 10.0.1.25:1883

O algoritmo *Round Robin* garante que cada servidor receba uma parte equitativa do tráfego antes de se iniciar um novo ciclo, promovendo assim uma utilização eficaz dos recursos disponíveis.



## 6.6 Gestão de Bases de Dados

As bases de dados são fundamentais para a gestão eficaz de dados em qualquer sistema de informação, servindo como repositórios centrais onde os dados são armazenados, manipulados e organizados. No âmbito do nosso projeto, optou-se pela implementação de duas bases de dados MongoDB distintas, cada uma otimizada para um tipo específico de dados e carga de trabalho.

A primeira base de dados, dedicada aos sensores, é responsável pelo armazenamento de todas as amostras de dados coletadas em tempo real. Esta base de dados está configurada para lidar com grandes volumes de inserções de dados e consultas orientadas a séries temporais.

A segunda base de dados foca-se na autenticação e gestão de utilizadores, armazenando credenciais e perfis de utilizador. Este sistema é otimizado para operações de segurança e acesso rápido a dados de autenticação.

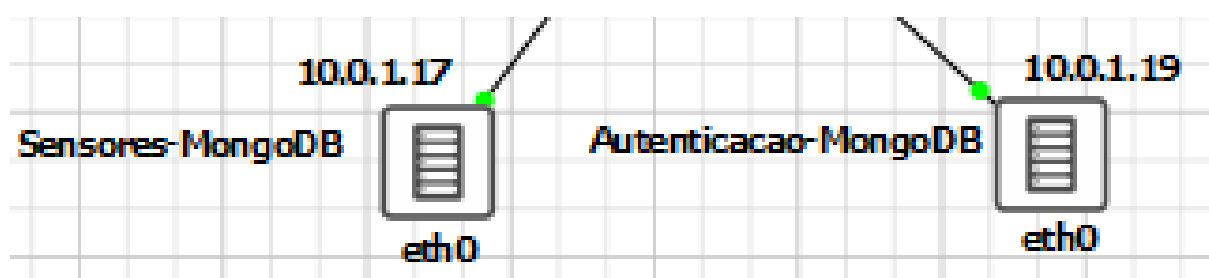


Figura 5: Diagrama das Bases de Dados MongoDB para Sensores e Autenticação.

Com esta arquitetura, assegura-se que a integridade e a performance não são comprometidas, ao mesmo tempo que se mantém uma clara separação de responsabilidades dentro do ecossistema do projeto.

## 6.7 Configuração dos Brokers MQTT

Os Brokers MQTT são componentes cruciais na infraestrutura IoT, responsáveis pela gestão de mensagens entre os dispositivos sensores e o sistema de processamento central. Utilizamos uma abordagem replicada que assegura continuidade do serviço e balanceamento de carga.

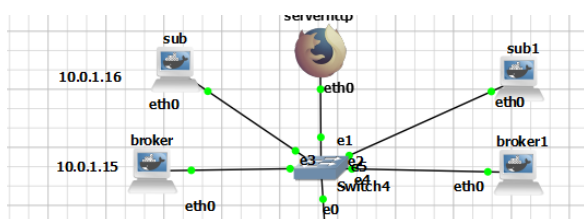


Figura 6: Brokers na Cloud 1

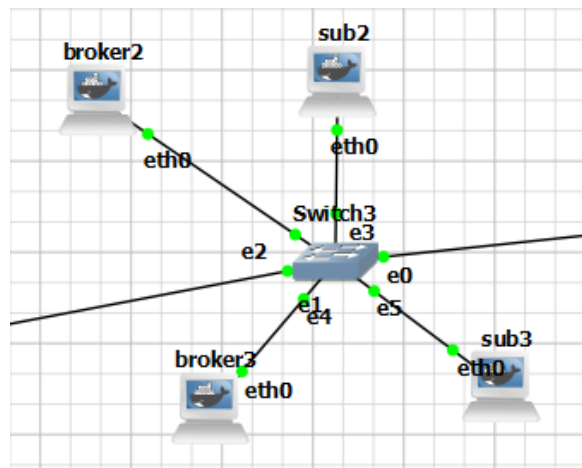


Figura 7: Brokers na Cloud 2

A configuração de cada broker é mantida intencionalmente simples para facilitar a manutenção e escalabilidade. Os brokers estão configurados para aceitar conexões na porta padrão MQTT (1883) e permitir a autenticação anônima, adequada para cenários controlados onde a simplificação do acesso é prioritária. Abaixo encontra-se o excerto de configuração para os brokers MQTT utilizados:

```
listener 1883
allow_anonymous true
```

Através desta configuração, os brokers estão prontos a receber comunicações dos dispositivos IoT distribuídos pela rede, garantindo uma transmissão eficiente dos dados sensoriais para posterior processamento e análise.

Quanto aos subscribers, é preciso também modificar o endereço e criar um arquivo Python chamado "subscriber.py" com o seguinte código:

```
import paho.mqtt.client as mqtt
from pymongo import MongoClient
import datetime

# Conexão com o MongoDB
mongo_client = MongoClient('10.0.1.17', 27017)
db = mongo_client['EmulacaoBaseDados']
colecacao_normal = db['entradas_normais']
colecacao_alertas = db['alertas']

def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Conectado ao broker MQTT")
        client.subscribe("meu/topico")
    else:
        print("Falha na conexão. Código de retorno:",rc)

def on_message(client, userdata, msg):
    mensagem = msg.payload.decode()
    print(f"Nova mensagem recebida no tópico {msg.topic}: {mensagem}")

    # Capturar o timestamp atual
    timestamp_atual = datetime.datetime.now()

    if "-> Alerta " in mensagem:
        colecacao_alertas.insert_one({"topico": msg.topic, "mensagem": mensagem,
                                      "timestamp": timestamp_atual})
    else:
        colecacao_normal.insert_one({"topico": msg.topic, "mensagem": mensagem,
                                     "timestamp": timestamp_atual})

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
broker_address = "10.0.1.15"
port = 1883
client.connect(broker_address, port, 60)
client.loop_forever()
```

Este código é responsável por estabelecer uma conexão MQTT com um broker específico e inserir os dados recebidos na base de dados. Cada subscriber terá um broker associado, portanto, é crucial garantir que o endereço do broker no código seja o correto. Para realizar essa personalização, utilizamos a seguinte linha, que deve ser ajustada para diferentes subscribers:

```
broker_address = "10.0.1.15"
```

Após realizar a configuração adequada, basta executar o seguinte comando no console: `python subscriber.py`. É essencial garantir que o broker associado esteja ativo durante a execução do script.

## 6.8 Interface Gráfica do Utilizador

A interface gráfica do utilizador foi desenvolvida para proporcionar uma visualização intuitiva e detalhada dos dados recolhidos pelos sensores. Esta interface permite aos utilizadores monitorizar as condições ambientais em tempo real e analisar tendências históricas através de gráficos dinâmicos.



Figura 8: Gráficos representativos dos parâmetros ambientais medidos pelos sensores.

A funcionalidade de visualização de dados é complementada com a capacidade de exibir as mais recentes amostras de dados recolhidos e os alertas gerados, permitindo uma rápida resposta a condições críticas ou a anomalias detetadas.

Área	Data/Hora	Temperatura	Som	CO2	Humidade
2	28/12/2023, 12:13:39	6.88	60.99	804.99	31.94
1	28/12/2023, 12:13:39	19.57	53.1	800.46	25.42
2	28/12/2023, 12:13:38	7.17	59.64	803.7	32.82
1	28/12/2023, 12:13:38	20.47	55.39	798.31	29.97
2	28/12/2023, 12:13:37	10.17	55.22	803.6	30.98
1	28/12/2023, 12:13:37	20.35	57.84	793.97	30.77
2	28/12/2023, 12:13:36	29.75	55.33	793.91	46.03
1	28/12/2023, 12:13:36	4.0	61.19	804.48	35.31

Área	Tipo de Alerta	Valor
2	Alerta Som	51.0
2	Alerta Som	51.0
2	Alerta Som	52.18
1	Alerta Som	64.0
2	Alerta Som	51.0
2	Alerta Temperatura	4.0
1	Alerta Som	51.0
1	Alerta Temperatura	4.83
2	Alerta Som	52.26
1	Alerta Som	64.0
2	Alerta Som	62.32

Figura 9: Detalhe das últimas amostras de dados e alertas emitidos pelo sistema.

## 7 Testes

### 7.1 Rede sensores

A avaliação da rede de sensores foi focada em verificar a eficiência da captura de dados ambientais e a fiabilidade da transmissão subsequente para os atuadores e o nó central. A integridade e a precisão da coleta de amostras são vitais para assegurar a autenticidade dos dados monitorizados.

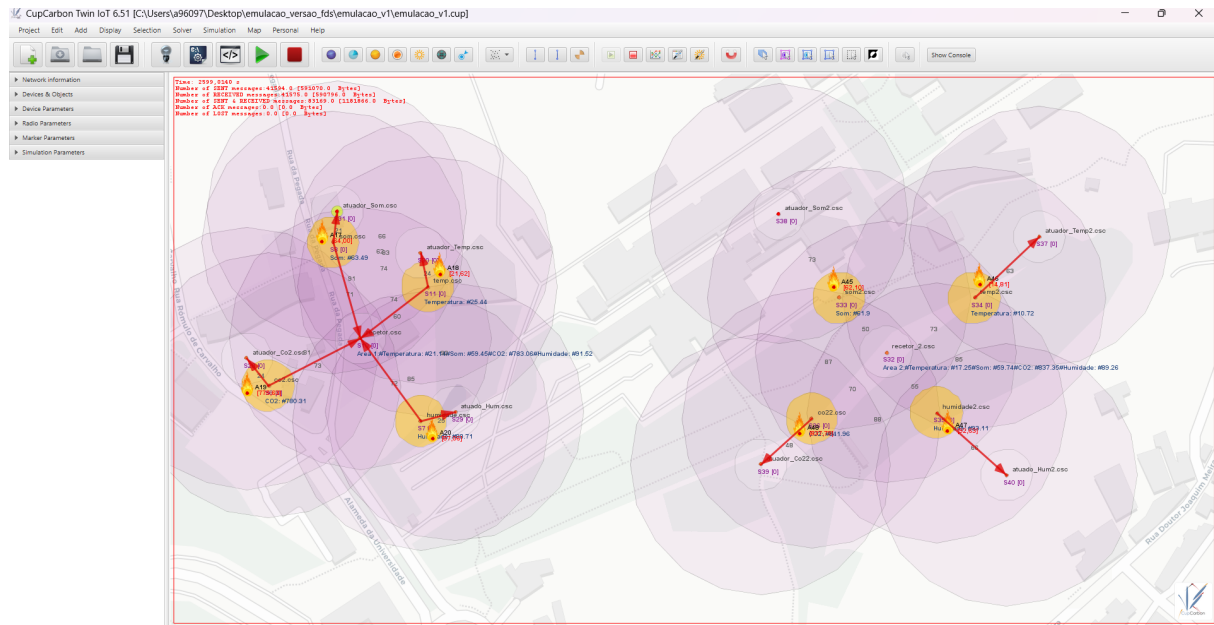


Figura 10: Funcionamento da recolha amostras e envio para os atuadores e no central.

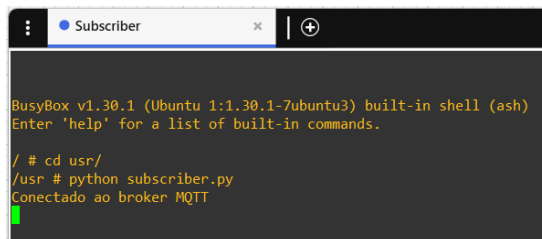
O dispositivo recetor desempenha um papel fundamental no sistema de IoT ao receber e registar os dados enviados pelos sensores distribuídos pela rede. É imperativo que o recetor processe todas as informações recebidas sem perda de dados.

```
Area 1:#Temperatura: #22.51#Som: #54.42#CO2: #849.51#Humidade: #51.72
Area 1:#Temperatura: #23.05#Som: #52.56#CO2: #845.74#Humidade: #55.78
Area 1:#Temperatura: #18.7#Som: #56.84#CO2: #843.39#Humidade: #52.74
Area 1:#Temperatura: #22.28#Som: #56.68#CO2: #838.44#Humidade: #56.78
Area 1:#Temperatura: #25.96#Som: #56.61#CO2: #840.32#Humidade: #52.27
Area 1:#Temperatura: #28.31#Som: #61.33#CO2: #839.41#Humidade: #47.68
Area 1:#Temperatura: #27.61#Som: #60.87#CO2: #835.98#Humidade: #46.04
```

Figura 11: Algumas amostras registadas pelo recetor.

## 7.2 Avaliação do Balanceamento de Carga no Ambiente GNS3

A avaliação do mecanismo de balanceamento de carga foi essencial para garantir a distribuição homogênea de mensagens MQTT no ambiente simulado. Utilizando o GNS3, configuramos um cenário que replicasse um sistema distribuído real, onde múltiplos brokers MQTT e subscribers interagem.

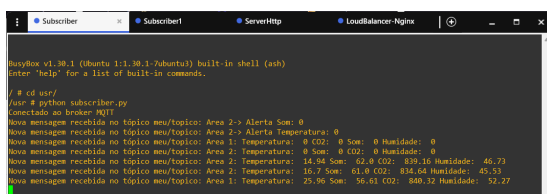


```
Subscriber
BusyBox v1.30.1 (Ubuntu 1:1.30.1-7ubuntu3) built-in shell (ash)
Enter 'help' for a list of built-in commands.

/ # cd usr/
/usr # python subscriber.py
Conectado ao broker MQTT
```

Figura 12: Instância de subscriber à espera de mensagens, pronta para iniciar o processamento assim que os dados forem recebidos.

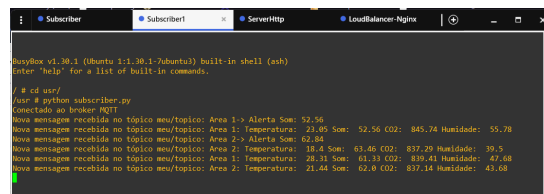
Com a simulação de tráfego de rede originado por dispositivos IoT, observamos a eficiência do load balancer na gestão de carga entre os diferentes nodos. Esta simulação foi vital para verificar a resiliência e a escalabilidade do sistema.



```
Subscriber
BusyBox v1.30.1 (Ubuntu 1:1.30.1-7ubuntu3) built-in shell (ash)
Enter 'help' for a list of built-in commands.

/ # cd usr/
/usr # python subscriber.py
Conectado ao broker MQTT
Nova mensagem recebida no tópico meu/topico: Area 2-> Alerta Som: 0
Nova mensagem recebida no tópico meu/topico: Area 2-> Alerta Temperatura: 0
Nova mensagem recebida no tópico meu/topico: Area 1: Temperatura: 0 CO2: 0 Som: 0 Humidade: 0
Nova mensagem recebida no tópico meu/topico: Area 2: Temperatura: 0 Som: 0 CO2: 0 Humidade: 0
Nova mensagem recebida no tópico meu/topico: Area 2: Temperatura: 14.54 Som: 62.8 CO2: 839.16 Humidade: 46.73
Nova mensagem recebida no tópico meu/topico: Area 2: Temperatura: 16.7 Som: 61.0 CO2: 834.64 Humidade: 45.53
Nova mensagem recebida no tópico meu/topico: Area 1: Temperatura: 25.96 Som: 56.61 CO2: 840.32 Humidade: 52.27
```

Figura 13: Subscriber 0 processando mensagens recebidas, indicando a efetiva distribuição de carga pelo balanceador.



```
Subscriber
BusyBox v1.30.1 (Ubuntu 1:1.30.1-7ubuntu3) built-in shell (ash)
Enter 'help' for a list of built-in commands.

/ # cd usr/
/usr # python subscriber.py
Conectado ao broker MQTT
Nova mensagem recebida no tópico meu/topico: Area 1-> Alerta Som: 52.50
Nova mensagem recebida no tópico meu/topico: Area 1: Temperatura: 23.85 Som: 52.56 CO2: 845.74 Humidade: 55.78
Nova mensagem recebida no tópico meu/topico: Area 2-> Alerta Som: 62.84
Nova mensagem recebida no tópico meu/topico: Area 2: Temperatura: 18.4 Som: 63.46 CO2: 837.29 Humidade: 39.5
Nova mensagem recebida no tópico meu/topico: Area 1: Temperatura: 28.31 Som: 61.33 CO2: 839.41 Humidade: 47.68
Nova mensagem recebida no tópico meu/topico: Area 2: Temperatura: 21.44 Som: 62.0 CO2: 837.14 Humidade: 43.68
```

Figura 14: Subscriber 1 igualmente recebendo dados, evidenciando a distribuição equitativa de mensagens.

As Figuras 13 e 14 demonstram os subscribers em ação, processando dados de maneira eficiente. A performance observada confirma a capacidade do sistema em lidar com a carga de rede de forma eficaz, um aspecto crítico para a manutenção da qualidade do serviço em ambientes IoT distribuídos.

### 7.3 Server HTTP e busca dos valores na base de dados.

Inicialmente, focamos em confirmar a conectividade entre o servidor HTTP e o MongoDB. Através de testes automatizados, asseguramos que as configurações de rede estão corretas e que o servidor é capaz de acessar o banco de dados sem interrupções, um fator crítico para a operação contínua do sistema.

```
Subscriber  Subscriber1  ServerHttp
/ # cd usr/
/usr #
/usr # ls
app.py  games  lib    lib64  libx32  sbin    src
bin     include lib32  libxec local  share   templates
/usr # python app.py
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:80
* Running on http://10.0.1.18:80
Press CTRL+C to quit
```

Figura 15: Server HTTP iniciou sem qualquer problema ao conectar com a base de dados.

Após estabelecer a conectividade, procedemos com testes para a recuperação de dados. Estes testes são projetados para verificar se o servidor pode efetivamente consultar e recuperar as amostras de dados armazenadas, essenciais para a funcionalidade de monitoramento.

Área	Data/Hora	Temperatura	Som	CO2	Humidade
2	11/01/2024, 16:23:00	22.05	61.0	829.51	52.97
1	11/01/2024, 16:22:59	29.77	56.93	831.98	59.01
2	11/01/2024, 16:22:47	27.06	63.88	832.99	49.56
1	11/01/2024, 16:22:47	28.02	58.21	834.05	58.45
1	11/01/2024, 16:22:37	43.5	736.0	56.39	64.33
1	11/01/2024, 16:22:32	45.68	56.58	532.09	67.88
2	11/01/2024, 16:22:31	27.82	61.0	831.5	43.17
2	11/01/2024,	30.57	64.0	835.02	43.4

Área	Tipo de Alerta	Valor
2	Alerta Som	62.32
1	Alerta Temperatura	43.5
1	Alerta Temperatura	45.68
1	Alerta CO2	532.09
1	Alerta CO2	531.44
2	Alerta Som	64.0
2	Alerta Som	63.22
2	Alerta Temperatura	3.84
2	Alerta Som	62.84
1	Alerta Som	52.56
2	Alerta Temperatura	0

Figura 16: Servidor recuperou amostras.

A capacidade do servidor, que opera no domínio *"emulacao.pt"*, de gerar gráficos a partir das amostras de dados é crucial para a análise e interpretação dos utilizadores. A figura seguinte ilustra os gráficos gerados a partir dos dados coletados.

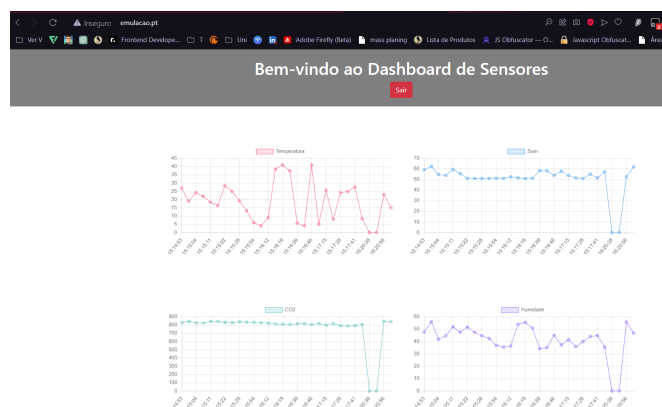


Figura 17: Gráficos com as amostras recuperadas.



## 8 Tecnologias/Recursos:

### 8.1 Hardware:

Ao longo deste projeto foram utilizadas diversas ferramentas tanto a nível de hardware como a nível de software, para assim se atingir todos os objetivos propostos.

Ferramenta	Quantidade	Utilização
Computador Portátil	3	Elaboração do relatório, desenvolvimento do código, pesquisa e testes.

Tabela 3: Ferramentas de hardware.

### 8.2 Software:

Relativamente ao software, as ferramentas utilizadas encontram-se na Tabela 4.






Logo	Descrição	Utilização
	Discord	Comunicação entre os elementos do grupo.
	Visual Studio Code	IDE utilizado para o desenvolvimento do código para o serverHTTP.
	Overleaf	Realização e sincronização do relatório.
	Cupcarbon	Desenvolvimento e teste de redes sensores.
	GNS3	Desenvolvimento e teste da plataforma IoT.

Tabela 4: Tabela com Logos e Descrições.

## 9 Competências Desenvolvidas:

O desenvolvimento bem-sucedido do nosso projeto "Saúde Inteligente em Ambientes Urbanos" exigiu uma ampla gama de competências técnicas e conhecimentos específicos. Abaixo, apresentamos as principais competências que foram essenciais para a implementação eficaz do sistema IoT e para alcançar os objetivos do projeto:

Competências Adquiridas	Contribuição das Disciplinas
Conhecimentos aprofundados em Redes e Comunicação	As disciplinas de Redes de Computadores I e II foram cruciais para compreender a arquitetura de redes, protocolos e segurança, fundamentais na configuração e manutenção da rede de sensores IoT.
Competências em Simulação e Emulação IoT	O Projeto Integrado de Telecomunicações proporcionou experiência prática em simulação e emulação, essencial para testar e validar a eficácia da nossa rede de sensores e comunicações IoT.
Habilidades em Servidores e Infraestrutura Cloud	As disciplinas de Projeto Integrado de Telecomunicações e Métodos de Programação I e II foram fundamentais para desenvolver nossa capacidade de implementar e gerir servidores, bem como para compreender os princípios da computação em cloud, que sustentam a nossa plataforma IoT.

Tabela 5: Competências Adquiridas e a Contribuição das Disciplinas

## 10 Conclusão

Com a finalização deste projeto, nós, enquanto grupo, consolidámos um leque substancial de competências dentro do âmbito da Internet das Coisas. Desde a configuração de redes complexas até à orquestração e administração de uma infraestrutura IoT integrada, cada etapa foi uma oportunidade de aprofundar conhecimentos técnicos e práticos essenciais para os desafios atuais do setor.

Aprendemos a importância de uma rede segura e bem estruturada e desenvolvemos soluções IoT que prometem transformar positivamente os ambientes urbanos, com especial enfoque na saúde pública. As competências adquiridas em simulação, análise de dados e gestão de sistemas em cloud são agora parte intrínseca do nosso arsenal profissional, capacitando-nos a implementar estas tecnologias com eficiência e responsabilidade.

Concluimos que a monitorização ambiental é um alicerce para cidades mais inteligentes e sustentáveis. Os dados que recolhemos fornecem uma base sólida para decisões políticas informadas e ações que visem melhorar a qualidade de vida nas áreas urbanas.

Em resumo, este projeto foi uma jornada enriquecedora e um marco significativo no nosso desenvolvimento. Estamos preparados e motivados para levar adiante as competências adquiridas e aplicá-las em futuras iniciativas profissionais e académicas.