Instituto Superior de Engenharia de Lisboa

Computação Distribuída

Sistema Distribuído para Coleção e Armazenamento de Amostras de Velocidade de Automóveis

Mestrado em Engenharia Informática de Multimédia

Pedro Gonçalves, 45890

Rodrigo Dias, 45881

Rúben Santos, 49063

Semestre de Inverno, 2021/2022

Índice

Índice de Ilustrações

1. Introdução

O objetivo deste projeto consiste na implementação de um sistema distribuído de obtenção e armazenamento de valores de velocidade coletados por sensores dispostos em diversos locais.

Tirar-se-á partido das seguintes tecnologias:

- ***RabbitMQ***
- ***Spread Daemon***
- ***gRPC***

O código fonte, os executáveis (packages em formato "***.jar***" na pasta **/packages**) e os comandos necessários à execução dos mesmos encontram-se nesta página do **Github**.

2. Análise de Requisitos

Será necessário primeiro definir o problema em concreto, os mecanismos que estarão envolvidos e uma possível solução. A análise de requisitos é o primeiro passo no desenvolvimento de qualquer tipo de software, e deve apontar o programador na direção correta.

2.1. Definição do Problema

O sistema deverá apresentar as seguintes funcionalidades fundamentais:

1. Coleção de valores de velocidade (em **km/h**) juntamente com **ID** do sensor correspondente e o local e data em que foi coletado.
2. Os valores coletados deverão ser armazenados num cluster de servidores que suporte distribuição de carga.
3. Os servidores deverão ser capazes de responder a alguns pedidos por parte de aplicações do utilizador, como a velocidade mais alta alguma vez registada ou a média das velocidades numa determinada data.

Na **figura 1** apresenta-se o diagrama geral do sistema, que ilustra os quatro níveis, cada um com determinados requisitos.

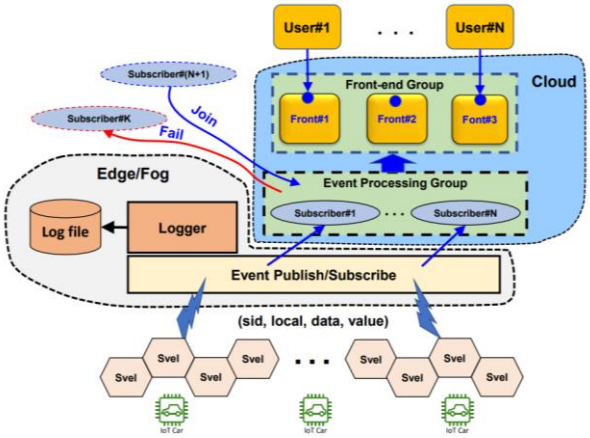

Figura 1 - Diagrama geral do sistema

- **1º Nível (Sensores):** Uma aplicação que simule a coleção de dados por parte do sensor.
- **2º Nível (Edge/Fog):** Um servidor ***RabbitMQ*** que suporte o modelo ***Publish/Subscribe***, e que receba os dados dos múltiplos sensores. Neste nível deverá ser feito ***logging*** dos dados recebidos dos sensores.
- **3º Nível (Cloud):** Dois grupos Spread. O ***Event Processing Group*** contará com consumidores cuja função é coletar os dados publicados pelos sensores no servidor ***RabbitMQ***, e ***Front-End Group*** contará com um cluster de servidores que irão receber os dados coletados pelos consumidores, armazenando-os em memória.
- **4º Nível (Acesso aos dados):** Aplicações para que os utilizadores, por ***gRPC***, possam interagir com os servidores do ***Front-End Group*** para obter determinadas informações acerca dos dados coletados.

2.2. Requisitos da Solução

Perante os requisitos do capítulo anterior, começar-se-á por definir, com mais rigor, os diferentes níveis do sistema. Na **figura 2**, verifica-se uma ilustração do sistema mais detalhada.

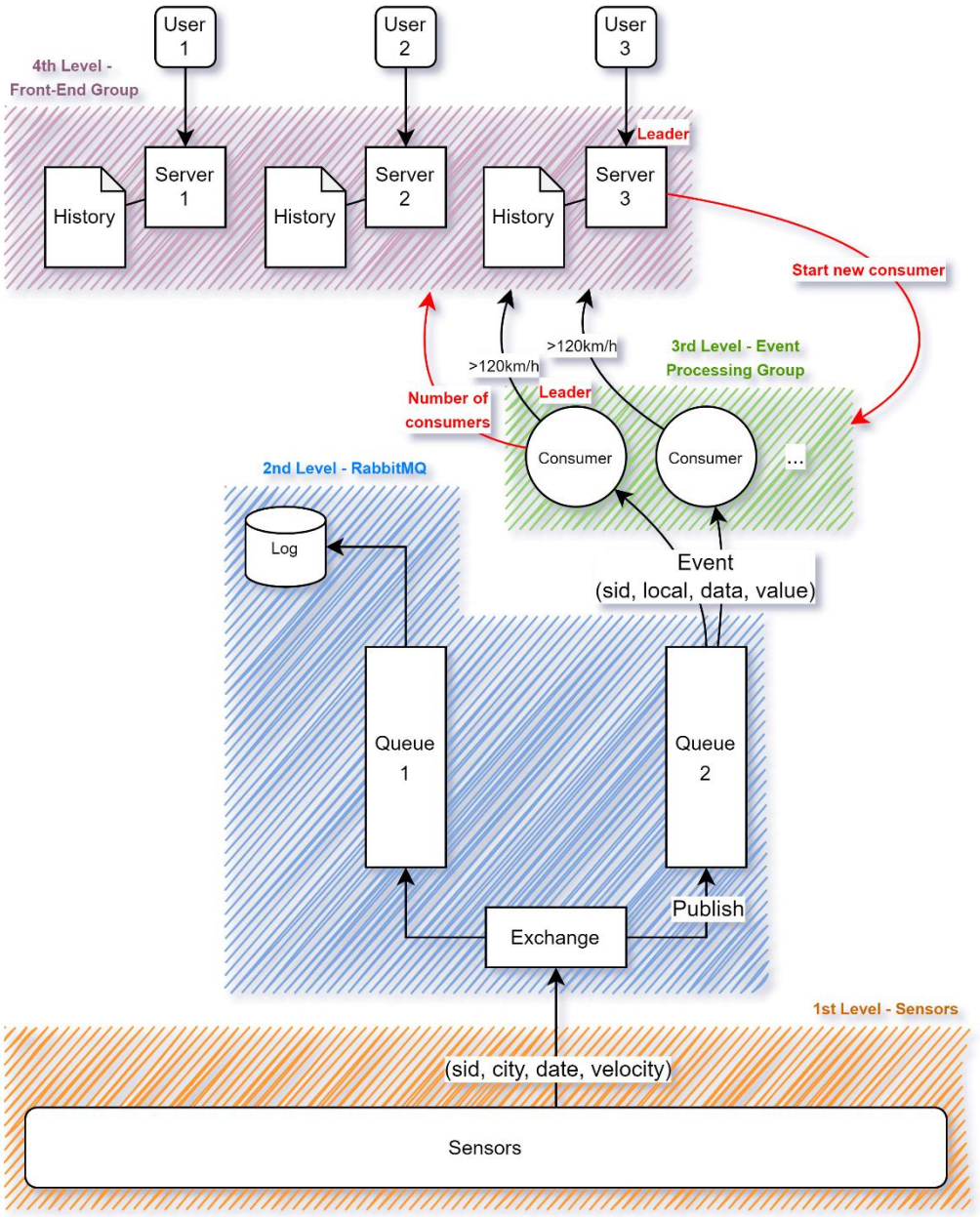

Figura 2 - Diagram detalhado do sistema

2.2.1. Primeiro Nível

No primeiro nível do sistema, no qual se substituirão os sensores por aplicações que simulam a coleção de dados, dever-se-á considerar alguns requisitos adicionais:

- A aplicação deverá aceitar como parâmetros:
  - O ***Endpoint*** (**<IP>:<PORT>**) do servidor ***broker RabbitMQ***;
  - O ritmo a que serão gerados dados;
  - O ID do sensor;
  - A cidade onde o sensor se situa;
  - Um parâmetro que permita influenciar a data de coleção dos dados (para efeitos de teste).
- Deverá gerar valores de velocidade aleatórios, compreendidos entre **0 km/h** e **270 km/h**.
- Aquando da geração de um novo valor, deverá publicá-lo para o ***Exchange*** do servidor ***RabbitMQ*** no segundo nível.

2.2.2. Segundo Nível

No segundo nível, deverá existir uma aplicação capaz de configurar o servidor ***RabbitMQ*** devidamente. Essa aplicação deverá aceitar como parâmetros o Endpoint do servidor ***broker RabbitMQ***. Deverá criar um **Exchange** com duas **Queues** associadas (***bind***), uma visível para os consumidores do terceiro nível do sistema, e outra para ***logging***.

Devido à necessidade de fazer logging, deverá ser criada uma segunda aplicação neste nível, capaz de coletar os dados da **Queue** de ***logging*** e de os guardar num ficheiro.

2.2.3. Terceiro Nível

O terceiro nível será o mais complexo, visto que se responsabiliza por processar, armazenar e disponibilizar os dados, e irá interagir com os outros níveis.

Será necessária uma aplicação que faça o papel de consumidor, que terá os seguintes objetivos:

- Ingressar no **Event Processing Spread Group**;
- Coletar dados da **Queue** do servidor ***RabbitMQ*** do segundo nível;
- Dos dados que coleta, deverá partilhar com o **Front-End Group** aqueles que apresentam uma velocidade maior que **120 km/h**;
- Se o consumidor em questão for o líder do grupo, partilhar também com o **Front-End Group** atualizações acerca de alterações no número de membros, isto é, consumidores, do **Event Processing Group**.

O consumidor líder, como referido, possui tarefas adicionais. Mas como é que um consumidor sabe se é o líder ou não? Isso será abordado no capítulo de **Implementação**.

Será também necessária uma aplicação que faça o papel de servidor, e que deverá concretizar o seguinte:

- Ingressar no **Front-End Spread Group**;
- Receber e armazenar dados que vêm dos consumidores;
- Receber atualizações do número de membros do **Event Processing Group**;
- Através de um serviço ***gRPC***, disponibilizar determinadas ***queries*** para que a aplicação do utilizador possa interagir com os dados armazenados no servidor.
- Se o servidor em questão for o líder do grupo, deverá, quando um novo servidor se juntar ao Front-End Group, partilhar com ele o seu histórico de dados armazenado em memória. Essa partilha será feita através de um evento no Front-End Group, que deverá ser interpretado apenas pelo servidor que acabou de se juntar.

O algoritmo de eleição do líder, e de partilha privada do histórico (no caso de ser o líder), será abordado no capítulo de **Implementação**.

As ***queries*** a disponibilizar serão:

- Maior valor de velocidade registado;
- Menor valor (acima de 120 km/h) de velocidade registado;
- Dados coletados numa determinada cidade;
- Dados coletados numa determinada data;
- Média das velocidades coletadas numa determinada cidade;
- Média das velocidades coletadas numa determinada data;
- Verificar o número de consumidores ativos no **Event Processing Group**.

2.2.4. Quarto Nível

O quarto nível representa o acesso aos dados por uma aplicação de utilizador. Essa aplicação, implementando um contrato de serviço do servidor a que se irá conectar, deverá ser capaz de pedir dados, consoante as ***queries*** disponíveis nesse mesmo contrato.

Os resultados das queries deverão ser imprimidos no formato **"SID: <sid>, CITY: <city>, DATE: <date>, VELOCITY: <velocity>".**

3. Implementação

3.1. Configurador RabbitMQ

A aplicação **RabbitConfigurator** deverá configurar um ***Exchange*** (**Fanout**) no servidor **RabbitMQ**, juntamente com duas filas associadas. O ***Exchange*** terá nome **VELOCITY_SAMPLES** e ***routing key*** **VELOCITY_SAMPLES**. O nome das filas será **VELOCITY_QUEUE** para a fila a que os consumidores estarão subscritos, e **VELOCITY_QUEUE_LOG** para a fila para efeitos de ***logging***.

3.2. Sensor

A aplicação **Sensor** deverá apresentar duas funcionalidades fundamentais:

- Gerar números aleatórios de forma periódica.
- Publicar eventos no servidor **RabbitMQ**.

Para executar código periodicamente, tirar-se-á partido da classe **ScheduledExecutorService** para criar uma thread pool que se responsabilizará por agendar e executar instruções após um determinado ***delay***.

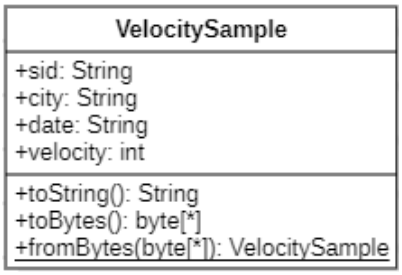

Figura 3 - Estrutura que representa uma velocidade

Na **figura 3** apresenta-se a estrutura de dados que será convertida para um array de ***bytes***, e que será publicada no servidor **RabbitMQ** (Exchange de nome **VELOCITY_SAMPLES**), bem como será transportada nos eventos do servidor **Spread**.

3.3. Consumidor

A aplicação **Consumer** deverá adotar uma abordagem de subscrição à fila **VELOCITY_QUEUE** sem ***Acknowledge***. Deverá ser a própria aplicação a decidir se o **ACK** é enviado ou não: Se o envio dos dados coletados para o **Front-End Group** falhar, deve ser enviado **NACK** para os dados sejam repostos na fila, de forma a que não sejam esquecidos e possam voltar a ser coletados. Caso os dados sejam enviados para o **Front-End Group** com sucesso, **ACK** deverá ser enviado. A **figura 4** ilustra os dois casos possíveis

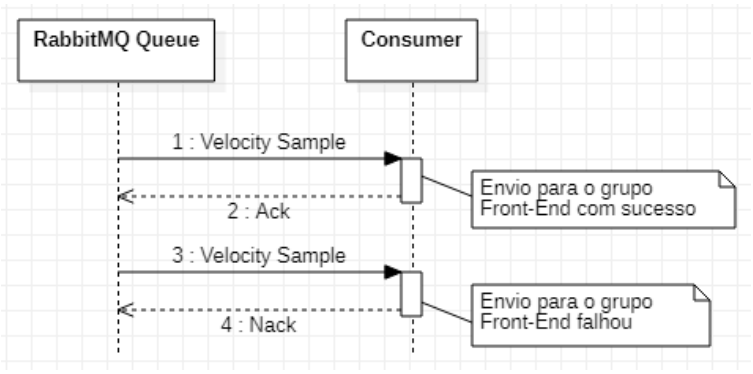

Figura 4 - Sistema de Acknowledge

Outro aspeto importante da implementação do consumidor, incide na necessidade de eleger um consumidor líder que se responsabilizará por enviar atualizações acerca do número de consumidores ativos para o grupo de **Front-End**. O algoritmo de eleição tem início no nome dado a cada um dos consumidores quando ingressam no grupo **Spread** de **Processamento de Eventos**. Deverá corresponder aos segundos desde o **EPOCH**, de forma a que todos os consumidores tenham nomes únicos, e o mais antigo apresento o nome lexicamente menor. Isto permite que ao obter o nome dos membros do **Event Processing Group**, ao ordenar, o primeiro da lista será o mais antigo, e portanto, o líder. Quando um consumidor abandona o grupo (originando um evento do qual tirar-se-á partido), o consumidor com o nome lexicamente menor (ou seja, o mais antigo) será eleito o novo líder.

A **figura 5** representa um esquema temporal, que exemplifica um cenário em que três consumidores entram em funcionamento, e quando o **Consumidor 1** se disconecta, ocorre necessidade de reeleição de um líder.

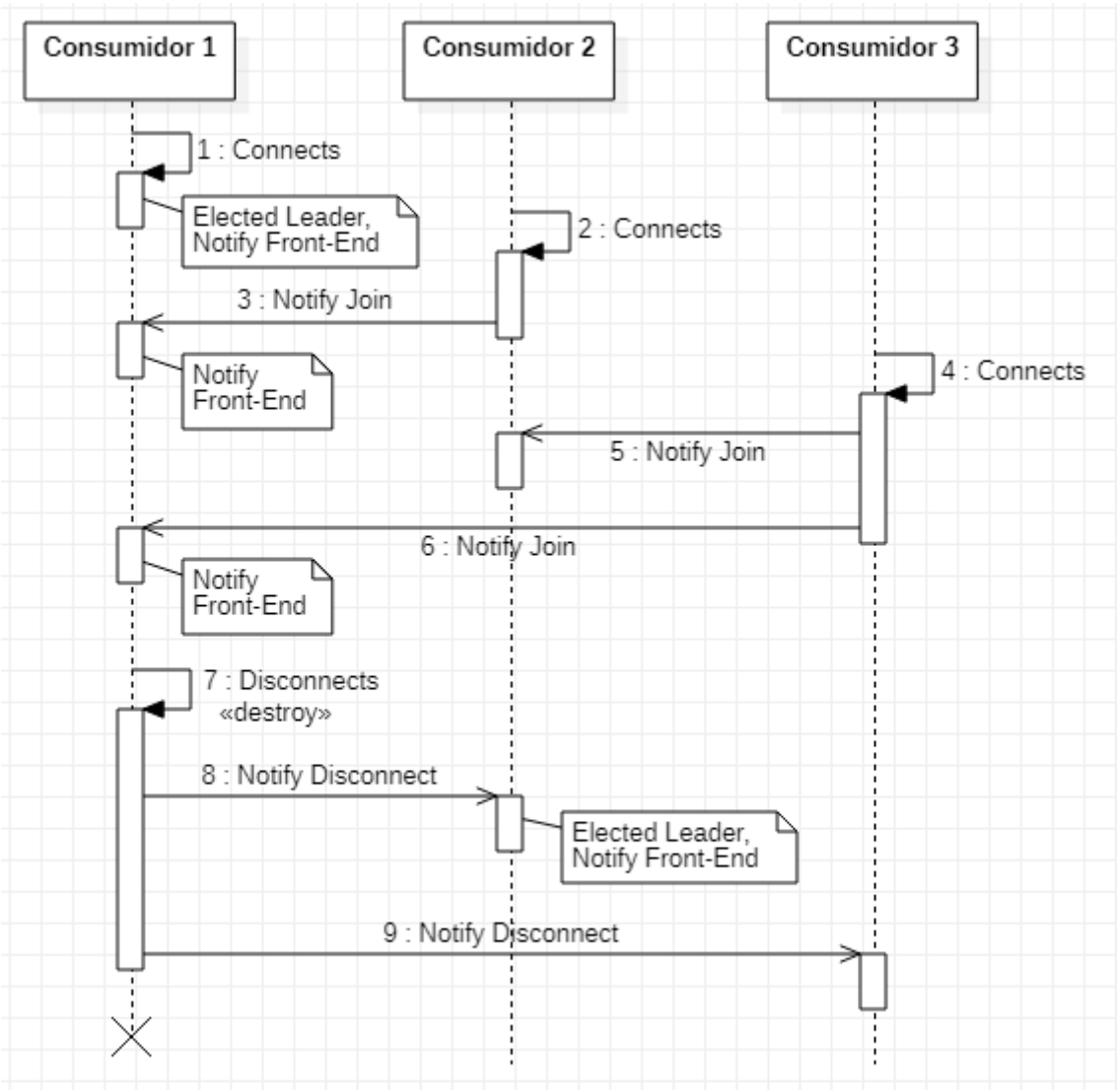

Figura 5 - Cenário com três consumidores

3.4. Server

A aplicação **Server** deverá implementar um contrato de serviço que permite disponibilizar à aplicação **User** algumas queries acerca da informação armazenada sobre as velocidades coletadas pelos sensores. O contrato implementado é o seguinte:

```
1 service VelocityQueries {
2     rpc queryHighestVelocity(google.protobuf.Empty)
  returns (Sample);
3     rpc queryLowestVelocity(google.protobuf.Empty)
  returns (Sample);
4     rpc queryVelocitiesInCity(City) returns (Answer);
5     rpc queryVelocitiesInDate(Date) returns (Answer);
6     rpc queryAverageVelocityInCity(City) returns (
  Value);
7     rpc queryAverageVelocityInDate(Date) returns (
  Value);
8     rpc queryNumberOfConsumers(google.protobuf.Empty
  ) returns (Value);
9 }
10
11 message City {
12     string city = 1;
13 }
14
15 message Date {
16     string date = 1;
17 }
18
19 message Value {
20     int32 value = 7;
21 }
22
23 message Answer {
24     repeated Sample samples = 2;
25 }
26
27 message Sample {
28     string sid = 3;
29     string city = 4;
30     string date = 5;
31     int32 velocity = 6;
32 }
33
34
```

Também apresentará um algoritmo de eleição do líder semelhante à aplicação **Consumer**. Neste caso, o mecanismo discutido previamente que elege sempre o servidor mais antigo, é particularmente interessante, visto que o servidor mais antigo terá sempre os dados mais atualizados (histórico e número de consumidores ativos).

Além disso, existirá outro algoritmo importante a desenvolver relativamente à sincronização do histórico de velocidades entre os servidores do grupo ***Spread*** **Front-End**.

Imagine-se um cenário em que estão dois servidores a trabalhar, e que já possuem, em memória, diversos dados recebidos dos consumidores. Se um terceiro servidor se juntar ao grupo **Front-End**, o líder deverá atualizá-lo com o seu histórico, de modo a manter todos os servidores ativos sincronizados.

Os dados serão enviados pelo líder através de uma única mensagem, com um prefixo "**ANSWER**", seguido da lista de velocidades, seguido de uma ***keyword*** "**CONSUMERS**", seguido do número de consumidores

ativos. O problema surge quando, a meio desse processo (em que o líder já percebeu que tem de atualizar o novo servidor e prepara os dados para o envio) o servidor líder falha e desconecta-se. Neste caso, o novo servidor acaba por não receber a mensagem de atualização do histórico, e o novo líder (que foi eleito em consequência da desconexão do líder anterior) não sabe que existe um novo servidor.

De modo precaver esta situação, todos os servidores devem ser iniciados com uma variável booleana ***pending*** colocada a **true**. Enquanto esta variável for verdadeira, sempre que ocorrer qualquer evento no grupo, o servidor em questão enviará para os restantes servidores uma mensagem "**REQUEST**", indicando que está à espera que o líder lhe envie o histórico. Por sua vez, o líder, recebendo esta mensagem, procede ao envio dos dados com o prefixo "**ANSWER**". Quando esta mensagem é recebida e serializada pelo novo servidor, a variável ***pending*** é colocada a **false**.

Para evitar duplicação de dados de velocidades, um servidor que possua a variável ***pending*** a **true**, ignorará novos valores de velocidades fornecidos por consumidores.

Na **figura 6** ilustra-se um cenário em que existem dois servidores ativos, que já receberam dados de um consumidor. Depois, junta-se um terceiro servidor que admite uma variável ***pending*** a **true**. Neste cenário o **servidor 1**, que é o líder, recebe a mensagem de ***request***, mas falha e desconecta-se antes que possa enviar a mensagem de resposta, com o histórico. Contudo, o terceiro servidor deteta isso, e reenvia a mensgaem de ***request*** ao novo líder, obtendo o histórico com sucesso.

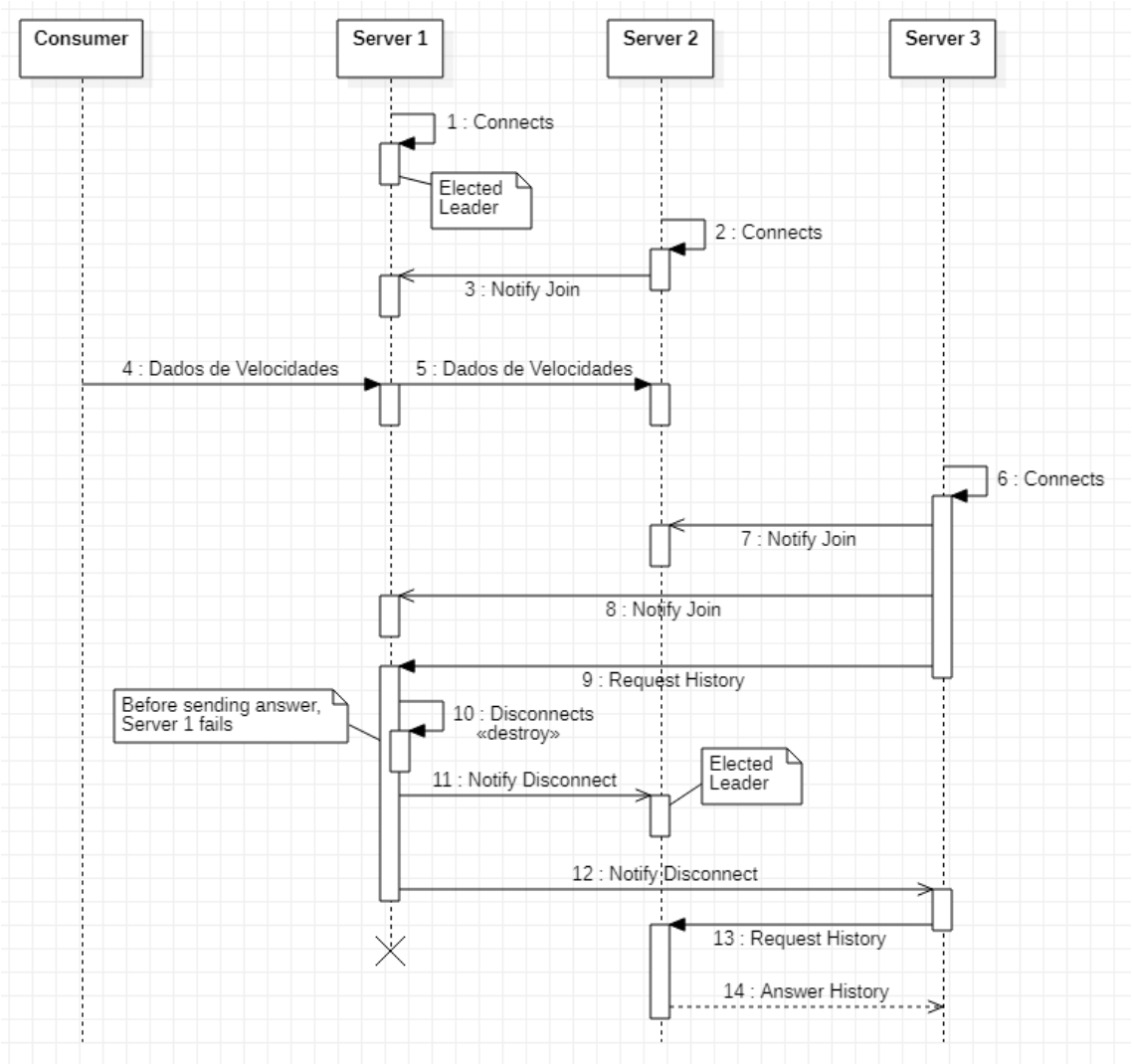

Figura 6 - Cenário com três servidores

3.5. User

A aplicação **User** terá de abrir um Stub de forma a utilizar os serviços disponibilizados pelo **Server**, mais precisamente o serviço **VelocityQueries**. Durante a execução, deverá manter o stub aberto, para que o utilizador consiga realizar as ***queries*** que desejar.

3.6. Logger

A aplicação Logger deverá obter os eventos da fila QUEUE_VELOCITY_LOG do servidor RabbitMQ, e escrevê-los para um ficheiro. Para escrever num ficheiro recorrer-se-á à class FileWriter.

4. Teste

De forma a tornar os programas o mais versáteis possível, ao executá-los na janela de comandos, existem alguns parâmetros opcionais. A forma como os comandos de execução devem ser utilizados encontra-se nesta página do **Github**, onde também se encontra o código fonte e os programas (**packages**) executáveis na pasta **/packages**.

Para testar o sistema, foram utilizadas quatro máquinas virtuais. A primeira servirá o propósito de hospedar o servidor **RabbitMQ**, onde serão configurados o exchange e as filas, com a aplicação **RabbitConfigurator**. As outras três estarã conectadas entre si para hospedar um servidor distribuído Daemon Spread, onde serão criados os grupos **Event Processing** e **Front-End**. Ou seja, nas três últimas três máquinas virtuais, hospedar-se-ão também **Servers** e **Consumers**. O **RabbitConfigurator**, o **Sensor**, o **User** e o **Logger** serão executados.

O primeiro passo é ligar a máquina em que será hospedado o RabbitMQ, e configurá-lo utilizando a aplicação **RabbitConfigurator**.

No máquina virtual com ***IP* 35.197.247.130**:

```
[CD2122-G11@vm-centos8-java11-4 ~]$ docker run -d --hostname rabbithost --name rabbitmg -p 5672:5672
 -p 15672:15672 rabbitmq:management
docker: Error response from daemon: Conflict. The container name "/rabbitmg" is already in use by co
ntainer "02b591cc7ffa93b6d8be45a9938e08257d790efb533fdfb92855889562ab9477". You have to remove (or r
ename) that container to be able to reuse that name.
See 'docker run --help'.
[CD2122-G11@vm-centos8-java11-4 ~]$ docker stop rabbitmg
rabbitmg
[CD2122-G11@vm-centos8-java11-4 ~]$ docker rm rabbitmg
rabbitmg
[CD2122-G11@vm-centos8-java11-4 ~]$ docker run -d --hostname rabbithost --name rabbitmg -p 5672:5672
 -p 15672:15672 rabbitmq:management
021c80d94626add12051a68e83bb445025cac1fc3cd74d7c47608577ba0d49a8
[CD2122-G11@vm-centos8-java11-4 ~]$
```

Figura 7 - Iniciar o docker RabbitMQ

Na máquina local, executar RabbitConfigurator, especificando o **IP** da máquina virtual e o porto em que o **RabbitMQ** está hospedado.

```
Microsoft Windows [Version 10.0.19043.1466]
(c) Microsoft Corporation. All rights reserved.

C:\Users\pedro\Desktop\Projetos\Distributed-Computing\SensorDistributedSystem\packages>java -jar RabbitConfigurator.jar
--broker-endpoint=35.197.247.130:5672
Queues VELOCITY_QUEUE and VELOCITY_QUEUE_LOG bound to Exchange VELOCITY_SAMPLES

C:\Users\pedro\Desktop\Projetos\Distributed-Computing\SensorDistributedSystem\packages>
```

Figura 8 - Execução do RabbitConfigurator

Na interface web (http://35.197.247.130:15672) verifica-se que o Exchange foi criado:

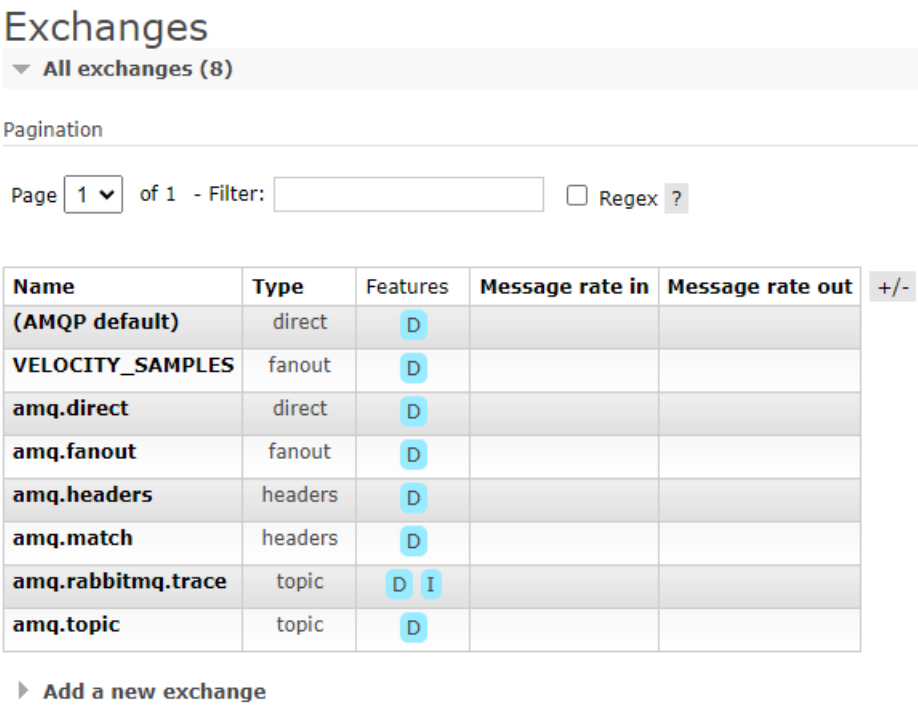

Figura 9 – Confirmação da criação do Exchange

Bem como as duas filas:

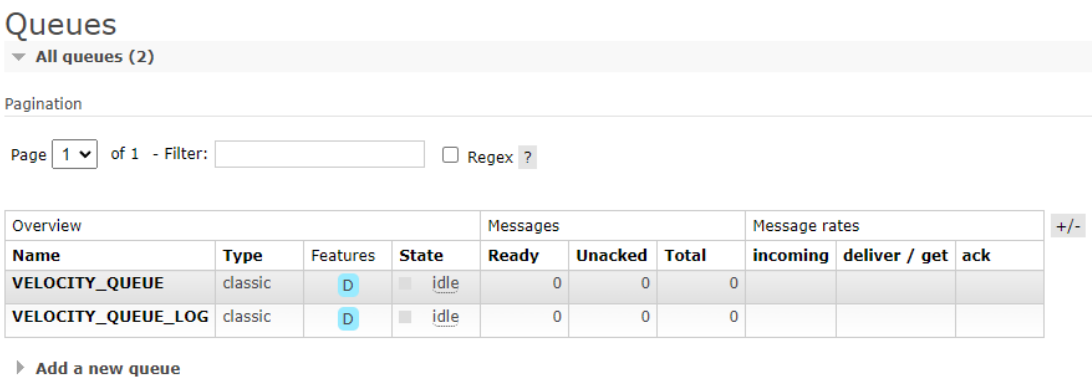

Figura 10 - Confirmação da criação das filas

Na figura 11, verifica-se que as duas filas estão associadas ao Exchange, e que o próprio é do tipo Fanout como pretendido.

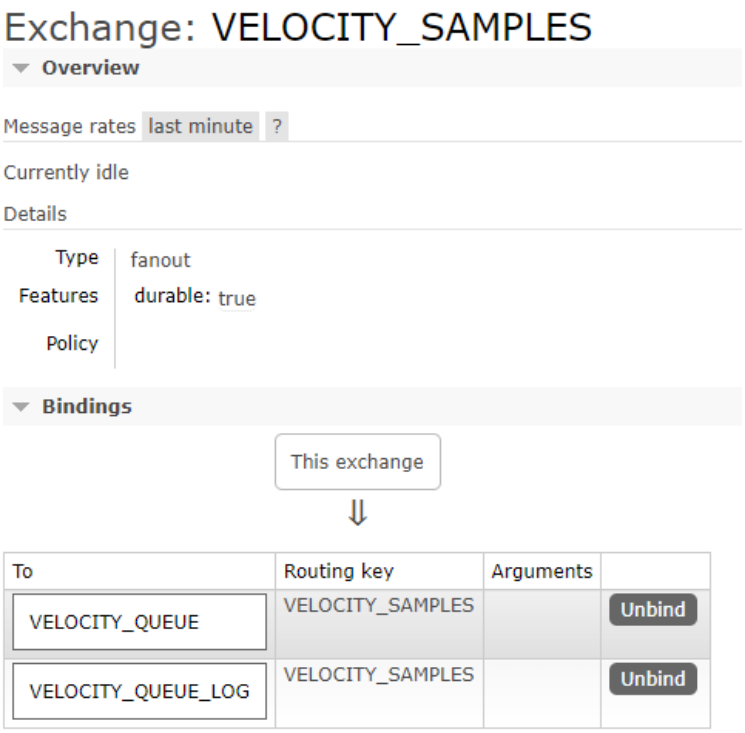

Figura 11 - Confirmação da associação das filas e do tipo de Exchange

O próximo passo é ligar as restantes máquinas virtuais, iniciar o **Daemon Spread** e ligar os **Servers** e **Consumers**. Note-se que dever-se-á ligar primeiro os servidores, para que estes detetem os eventos enviados pelos consumidores aquando da sua entrada (para atualizar o **Front-End Group** com o número de consumidores ativos). Se os Consumidores forem ligados primeiro, esses eventos não são recebidos por ninguém, e quando os servidores se ligarem, terão de esperar que algum consumidor saia ou entre para receberem atualizações acerca do número de consumidores ativos.

O ficheiro **newspread.conf**, partilhado entre as três máquians virtuais, apresenta o seguinte conteúdo:

```
Spread_Segment 10.154.0.2:4803 {
spreadNode1 10.154.0.2
}
Spread_Segment 10.154.0.3:4803 {
spreadNode2 10.154.0.3
}
Spread_Segment 10.154.0.6:4803 {
spreadNode3 10.154.0.6
}
# Linux user e group para o spread
DaemonUser = spread
DaemonGroup = spread
# Facilitates quick daemon restarts
SocketPortReuse = AUTO
```

Figura 12 - Conteúdo do ficheiro newspread.conf

Ligando o **Daemon Spread** nas três máquinas virtuais:

```
--------------------
Configuration at spreadNode3 is:
Num Segments 3
        1       [10.154.0.2]:4803
                spreadNode1             10.154.0.2      ID: 2108568596
        1       [10.154.0.3]:4803
                spreadNode2             10.154.0.3      ID: 643008652
        1       [10.154.0.6]:4803
                spreadNode3             10.154.0.6      ID: 3458913816
====================

++++++++++++++++++++++
Num of groups: 0
```

Figura 13 - Confirmação das treês máquinas virtuais no Sporead Daemon

Na **figura 13** encotnram-se os ***IPS*** itnernos das máquinas. Os ***IPS*** externos serão:

- Máquina 1: **34.89.26.148**
- Máquina 2: **35.242.139.84**
- Máquina 3: **35.242.139.84**

Na Máquina 1 hospedar-se-á um servidor no porto **5000**, especificando o ponto de acesso ao **Daemon Spread** da Máquina 2 (podia ser de qualquer uma das 3 máquinas):

```
Last login: Sat Jan 15 12:19:29 2022 from 81.84.67.234
[CD2122-G11@vm-centos8-java11 ~]$ java -jar TP2/Server.jar --daemon-endpoint=35.242.139.84:4803 --po
rt=5000
JOIN of #1642249238#spreadNode2
I'm the new leader!
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Jan 15, 2022 12:20:39 PM server.Server main
INFO: SERVER: Server started, listening on 5000...
```

Figura 14 - Execução do servidor da Máquina 1

Desde já, verifica-se que o Servidor 1 é promovido a líder.

Na Máquina 2 hospedar-se-á um servidor no porto **5000**, especificando o ponto de acesso ao **Daemon Spread** da Máquina 3:

```
Last login: Sat Jan 15 12:13:37 2022 from 81.84.67.234
[CD2122-G11@vm-centos8-java11-1 ~]$ java -jar TP2/Server.jar --daemon-endpoint=34.89.108.254:4803 --
port=5000
JOIN of #1642249546#spreadNode3
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Jan 15, 2022 12:25:46 PM server.Server main
INFO: SERVER: Server started, listening on 5000...
```

Figura 15 - Execução do servidor da Máquina 2

O Servidor 2 não é promovido a líder, pois existe no grupo um mais antigo (Servidor 1).

Por equanto não se executará um terceiro servidor. Na Máquina 1, executa-se um consumidor com o ponto de acesso ao **Daemon Spread** da Máquina 2, e com o ***IP*** e porto do broker **RabbitMQ**:

```
Last login: Sat Jan 15 12:31:00 2022 from 81.84.67.234
[CD2122-G11@vm-centos8-java11 ~]$ java -jar TP2/Consumer.jar --daemon-endpoint=35.242.139.84:4803 --
broker-endpoint=35.197.247.130:5672
Consumer Tag: [amq.ctag--1VlaP1W-Hsmey7NptC6VQ]
Type 'exit' to terminate this Consumer.
JOIN of #1642249977#spreadNode2
Sending current Event-Processing Group membership to Front-End Group...
```

Figura 16 - Execução de um servidor na Máquina 1

Note-se que o consumidor imediatamente envia uma atualização do número de consumidores ativos para o grupo **Front-End**, visto que é o líder.

Na Máquina 2, executa-se um consumidor com o ponto de acesso ao **Daemon Spread** da Máquina 3, e com o ***IP*** e porto do broker **RabbitMQ**.

```
Last login: Sat Jan 15 12:42:51 2022 from 81.84.67.234
[CD2122-G11@vm-centos8-java11-1 ~]$ java -jar TP2/Consumer.jar --broker-endpoint=10.154.0.8:5672 --d
aemon-endpoint=34.89.108.254:4803
Consumer Tag: [amq.ctag-uFmVddHmQMfEBS31lr4V2A]
Type 'exit' to terminate this Consumer.
JOIN of #1642250607#spreadNode3
```

Figura 17 - Execução de um servidor na Máquina 2

Assim que este segundo consumidor se junta, o primeiro, percebendo que se mantém o líder, volta a atualizar o **Front-End** com o número atual de consumidores ativos.

a Máquina 3, executa-se um consumidor com o ponto de acesso ao **Daemon Spread** da Máquina 1, e com o ***IP*** e porto do broker **RabbitMQ**.

```
Last login: Sat Jan 15 12:41:25 2022 from 81.84.67.234
[CD2122-G11@vm-centos8-java11-2 ~]$ java -jar TP2/Consumer.jar --broker-endpoint=35.197.247.130:5672
 --daemon-endpoint=34.89.26.148:4803
Consumer Tag: [amq.ctag-WNYegbaHekMR3e05w959KQ]
Type 'exit' to terminate this Consumer.
JOIN of #1642250772#spreadNode1
```

Figura 18 - Execução de um servidor na Máquina 3

Mais uma vez, o primeiro consumidor atualiza o grupo **Front-End** com o número atual de consumidores ativos, que agora são **3**.

O próximo passo é iniciar a aplicação **Logger** na máquina local:

```
C:\Users\pedro\Desktop\Projetos\Distributed-Computing\SensorDistributedSystem\packages>java -jar Logger.jar broker-endpo
int=35.197.247.130:5672
Created file: velocity_samples.txt
Consumer Tag: [amq.ctag-vYTFBgjzsbR_PDJfP0w9yw]
```

Figura 19 - Execução da aplicação Logger

Verifica-se que criou o ficheiro velocity_samples.txt, onde armazenará todos os valores de velocidade publicado na fila a que se subscreveu (**VELOCITY_QUEUE_LOG**).

Resta apenas iniciar alguns aplicações **Sensor** e utilizar a aplicação **User** para questionar os servidores acerca dos dados coletados. Executando alguns sensores:

```
C:\Users\pedro\Desktop\Projetos\Distributed-Computing\SensorDistributedSystem\packages>java -jar Sensor.jar --sid=1 --ci
ty=Lisbon --minusDay=3 --publish-rate=2000
Type 'exit' to terminate this Sensor.
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 259}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 6}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 54}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 186}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 15}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 84}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 70}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 61}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 24}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 183}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 136}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 4}
```

Figura 23 - Sensor 1, em Lisboa, há 3 dias

```
C:\Users\pedro\Desktop\Projetos\Distributed-Computing\SensorDistributedSystem\packages>java -jar Sensor.jar --sid=2 --ci
ty=Madrid --publish-rate=2000
Type 'exit' to terminate this Sensor.
VelocitySample {SID: 2, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 221}
VelocitySample {SID: 2, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 242}
VelocitySample {SID: 2, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 164}
VelocitySample {SID: 2, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 99}
VelocitySample {SID: 2, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 254}
VelocitySample {SID: 2, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 194}
VelocitySample {SID: 2, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 59}
VelocitySample {SID: 2, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 189}
VelocitySample {SID: 2, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 5}
VelocitySample {SID: 2, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 77}
VelocitySample {SID: 2, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 61}
VelocitySample {SID: 2, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 93}
```

Figura 22 - Sensor 2, em Madrid, hoje

```
C:\Users\pedro\Desktop\Projetos\Distributed-Computing\SensorDistributedSystem\packages>java -jar Sensor.jar --sid=1 --ci
ty=Lisbon --publish-rate=2000
Type 'exit' to terminate this Sensor.
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 207}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 61}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 162}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 71}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 43}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 241}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 135}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 164}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 240}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 213}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 71}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 166}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 217}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 156}
```

Figura 21 - Sensor 1, em Lisboa, hoje

```
C:\Users\pedro\Desktop\Projetos\Distributed-Computing\SensorDistributedSystem\packages>java -jar Sensor.jar --sid=3 --ci
ty=Madrid --minusDay=6 --publish-rate=2000
Type 'exit' to terminate this Sensor.
VelocitySample {SID: 3, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 139}
VelocitySample {SID: 3, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 175}
VelocitySample {SID: 3, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 249}
VelocitySample {SID: 3, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 88}
VelocitySample {SID: 3, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 258}
VelocitySample {SID: 3, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 258}
VelocitySample {SID: 3, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 49}
VelocitySample {SID: 3, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 165}
VelocitySample {SID: 3, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 247}
VelocitySample {SID: 3, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 100}
VelocitySample {SID: 3, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 159}
VelocitySample {SID: 3, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 131}
VelocitySample {SID: 3, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 49}
VelocitySample {SID: 3, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 244}
```

Figura 20 - Sensor 3, em Madrid, há 6 dias

Ainda antes de executar a aplicação **User**, excutar-se-á uma nova aplicação **Server** na Máquina 3, com acesso ao **Daemon Spread** pelo endpoint da Máquina 1.

```
Last login: Sat Jan 15 12:45:18 2022 from 81.84.67.234
[CD2122-G11@vm-centos8-java11-2 ~]$ java -jar TP2/Server.jar --daemon-endpoint=34.89.26.148:4803
JOIN of #1642251612#spreadNode1
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Jan 15, 2022 1:00:12 PM server.Server main
INFO: SERVER: Server started, listening on 5000...
```

Figura 24 - Execução de um servidor na Máquina 3

O servidor líder (o servidor da Máquina 1, visto que ainda é o mais antigo), é reponsável pro enviar o histórico corrente (composto por uma lista de velocidades e pelo número de consumidores ativos) para este novo servidor da Máquina 3.

Para testar se esse envio foi bem sucedido, executar-se-á a aplicação **User** com o ***endpoint*** deste novo servidor:

```
C:\Users\pedro\Desktop\Projetos\Distributed-Computing\SensorDistributedSystem\packages>java -jar User.jar --endpoint=34.
89.108.254:5000

MENU
 0 - Highest velocity registered
 1 - Lowest velocity registered
 2 - Velocities in a city
 3 - Velocities in a date
 4 - Average velocity in a city
 5 - Average velocity in a date
 6 - Number of active consumers
99 - Exit

Choose an Option?
6
There are 3 active consumers.

MENU
 0 - Highest velocity registered
 1 - Lowest velocity registered
 2 - Velocities in a city
 3 - Velocities in a date
 4 - Average velocity in a city
 5 - Average velocity in a date
 6 - Number of active consumers
99 - Exit

Choose an Option?
```

Figura 25 - Teste do número de consumidores ativos

Verifica-se que o servidor retorna o número de consumidores ativos atualizado (**3**). Se a mensagem de atualização dos histórico tivesse falhado, a resposta seria **0.** Se se disconectarem dois dois consumidores ativos, e utilizar-se o mesmo comando:

```
MENU
 0 - Highest velocity registered
 1 - Lowest velocity registered
 2 - Velocities in a city
 3 - Velocities in a date
 4 - Average velocity in a city
 5 - Average velocity in a date
 6 - Number of active consumers
99 - Exit

Choose an Option?
6
There are 1 active consumers.
```

Figura 26 - Disconexão de dois consumidores

Testam-se os restantes comandos:

```
MENU
 0 - Highest velocity registered
 1 - Lowest velocity registered
 2 - Velocities in a city
 3 - Velocities in a date
 4 - Average velocity in a city
 5 - Average velocity in a date
 6 - Number of active consumers
99 - Exit

Choose an Option?
0
SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 269

MENU
 0 - Highest velocity registered
 1 - Lowest velocity registered
 2 - Velocities in a city
 3 - Velocities in a date
 4 - Average velocity in a city
 5 - Average velocity in a date
 6 - Number of active consumers
99 - Exit

Choose an Option?
1
SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 121
```

Figura 27 - Maior e menor velocidade coletada

```
MENU
 0 - Highest velocity registered
 1 - Lowest velocity registered
 2 - Velocities in a city
 3 - Velocities in a date
 4 - Average velocity in a city
 5 - Average velocity in a date
 6 - Number of active consumers
99 - Exit

Choose an Option?
4
What city?
Lisbon
191
```

Figura 29 - Média numa cidade

```
MENU
 0 - Highest velocity registered
 1 - Lowest velocity registered
 2 - Velocities in a city
 3 - Velocities in a date
 4 - Average velocity in a city
 5 - Average velocity in a date
 6 - Number of active consumers
99 - Exit

Choose an Option?
5
What date? (DD-MM-YYYY)
12-01-2022
194
```

Figura 28 - Média numa data

Por fim, confirma-se que a aplicação **Logger** funcionou corretamente, e que o ficheiro **velocity_samples.txt** contém os dados publicados pelos sensores:

velocity_samples.txt - Notepad

File Edit Format View Help

```
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 259}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 6}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 54}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 186}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 15}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 84}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 70}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 61}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 24}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 183}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 136}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 4}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 4}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 127}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 120}
VelocitySample {SID: 2, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 221}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 268}
VelocitySample {SID: 2, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 242}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 146}
VelocitySample {SID: 2, CITY: Madrid, DATE: 15-01-2022, VELOCITY: 164}
VelocitySample {SID: 1, CITY: Lisbon, DATE: 15-01-2022, VELOCITY: 83}
```

Ln 1, Col 1 | 100% | Unix (LF) | UTF-8

Figura 30 - Ficheiro velocity_samples.txt

5. Conclusão

O projeto consistiu num sistema com várias camadas. A camada da aquisição de dados, em que se simularam sensores a obter valores de velocidade a um determinado período de tempo. Na camada **Edge/Fog**, estes dados são publicados para um mecanismo de **Publish/Subscribe** muito útil disponibilizado por um servidor **RabbitMQ**. Por sua vez, na camada da **Cloud**, as entidades que se subscreveram a esse mecanismo, recebem os eventos com novos dados e partilham-nos (filtrando-os) com o grupo **Spread** de servidores **Front-End** que estão abertos a aplicações **User** por **gRPC**. Por fim, as aplicações **User** permitem inquerir os dados armazenados nestes servidores. Todas estas tecnologias contribuem para a sincronização, fluxo e bom funcionamento do sistema.