 ISEL <small>INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA</small>	Departamento de Engenharia de Eletrónica de Telecomunicações e Computadores Mestrado em Engenharia Informática e de Computadores (MEIC) Mestrado em Engenharia Informática e Multimédia (MEIM)
14-12-2021	Computação Distribuída (Inverno 21/22)

Trabalho Prático de Avaliação Final

Objetivo: Implementação de um sistema distribuído englobando os diferentes paradigmas de interação e *middleware* estudados e já utilizados em Laboratório de aulas práticas

Notas prévias:

- As aulas de 03 e 04 de janeiro de 2022, serão totalmente alocadas para apoio à realização do trabalho. No entanto, é pressuposto, e faz parte dos ECTS da Unidade Curricular, que têm de dedicar horas de trabalho fora das aulas. Para eventual apoio e esclarecimento de dúvidas fora das aulas devem agendar com os professores o pedido de ajuda que poderá ser feito presencial ou remoto via Zoom. (nos *links* disponíveis no Moodle de cada turma);
- De acordo com as regras de avaliação definidas no slide 5 do conjunto *CD-01 Apresentação.pdf*, este trabalho tem um peso de 30% na avaliação final e é de entrega obrigatória, com avaliação maior ou igual a 10 valores;
- A entrega será realizada em Moodle com um ficheiro Zip, incluindo os projetos desenvolvidos (src e pom.xml sem incluir os artefactos), bem como outros ficheiros que considerem pertinentes para a avaliação do trabalho. É obrigatório a entrega de documento PDF como relatório técnico que descreve o sistema implementado, permitindo a um leitor compreender o objetivo, pressupostos, a arquitetura, a configuração para execução do sistema e as conclusões.
- Nas últimas aulas do semestre (17 e 18 de janeiro de 2022) cada grupo terá de apresentar e demonstrar, durante 10 minutos e para toda a turma, a funcionalidade do trabalho realizado;
- **A entrega limite no Moodle será 15 de janeiro de 2022 até às 23:59h.**

Considere um contexto (ver Figura 1) onde se pretende obter dados sobre eventos da velocidade de múltiplos veículos automóveis. Cada veículo está munido de um dispositivo IoT que ao passar em determinados locais comunica com sensores existentes nesse local, indicando a velocidade instantânea do veículo.

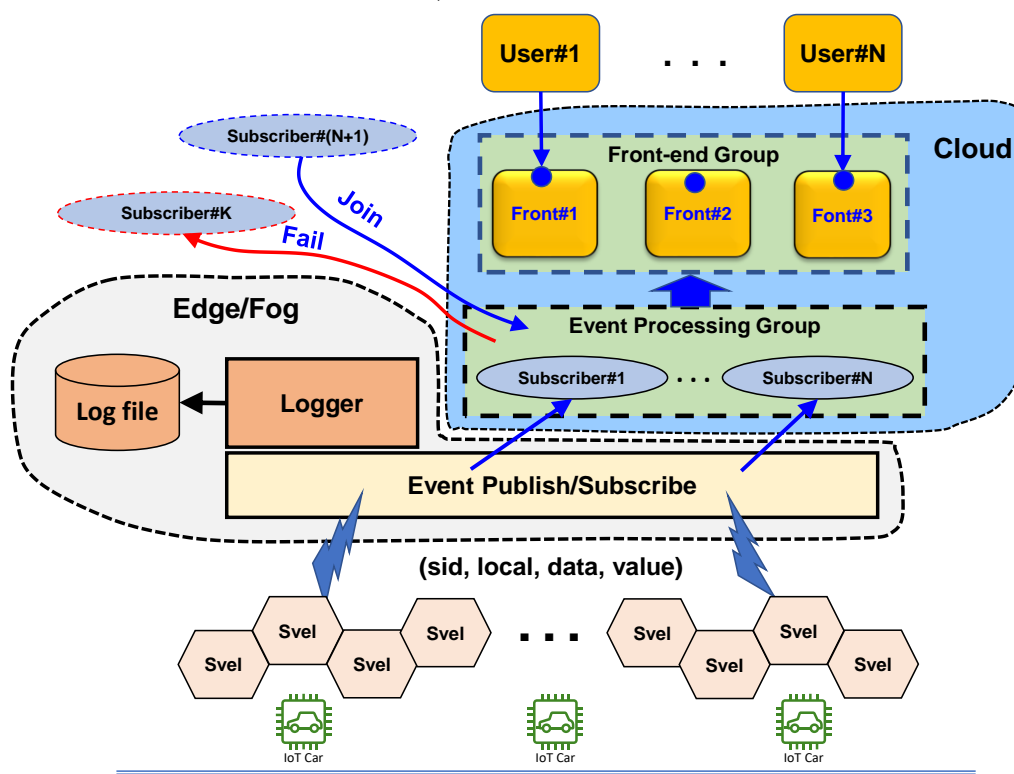



Figura 1 - Diagrama geral do sistema

 ISEL <small>INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA</small>	Departamento de Engenharia de Eletrónica de Telecomunicações e Computadores Mestrado em Engenharia Informática e de Computadores (MEIC) Mestrado em Engenharia Informática e Multimédia (MEIM)
14-12-2021	Computação Distribuída (Inverno 21/22)

Pretende-se desenvolver um sistema com componentes a três níveis com os requisitos a seguir descritos.

Requisitos funcionais

No primeiro nível deve ser desenvolvida uma pequena aplicação que simula o sensor que recebe a velocidade instantânea do veículo (geração aleatória de valores de velocidade instantânea entre 30 e 150 Km/hora). Assuma que cada sensor tem um identificador único (sid) e um local, incluído no seu *firmware*. Cada sensor, por cada velocidade detetada, gera um evento (sid, local, data, value) que envia para um segundo nível baseado no modelo *Publish/Subscribe*.

O segundo nível, normalmente designado *Edge/Fog*, tem as seguintes responsabilidades:


- Suportar o modelo *Publish/Subscribe* através de um servidor *RabbitMQ*;
- Receber todos os eventos vindos dos múltiplos sensores;
- Fazer *logging* num ficheiro de todos os eventos recebidos;
- Colocar num fila todos os eventos que serão processados por um grupo de consumidores existentes num terceiro nível.

No terceiro nível, normalmente designado *Cloud*, existem dois grupos de processos:

- *Event Processing Group*: É um grupo *Spread* de consumidores que, usando o padrão *work-queue*, recebem os múltiplos eventos da fila existente no 2º nível. Por cada evento recebido o consumidor verifica se o evento transporta um valor de velocidade superior a 120 Km/h. Se o valor for inferior o evento é descartado. Se o valor for superior a 120 Km/h o consumidor envia o evento como uma mensagem *Spread* para o grupo *Front-End*, descrito a seguir. É também responsabilidade deste grupo ter um *Leader* que informa o grupo *Front-End* sempre que um consumidor falha ou que um novo consumidor entra em funcionamento;
- *Front-End Group*: É um grupo *Spread* que garante tolerância a falhas e equilíbrio de carga para a aplicação dos utilizadores (*User*). Para tal o grupo é formado por 3 servidores que armazenam réplicas de todos os eventos vindos dos consumidores. O grupo deve ter um *Leader* que, no caso de reentrada de um servidor após falha, deve promover um consenso para repor o estado consistente dos eventos armazenados por todos os servidores do grupo. Cada servidor disponibiliza para a aplicação *User* um contrato gRPC com as seguintes funcionalidades:
 - ✓ Pesquisas simples sobre velocidades registadas nos sensores (valor mais alto de velocidade; média entre datas etc.)
 - ✓ Obtenção do número de consumidores existentes no *Event Processing Group* permitindo a determinados utilizadores atuarem lançando em execução novos consumidores;


Requisitos não funcionais

- Tanto na plataforma *RabbitMQ* como no *Spread* os dados das mensagens são normalmente um array de bytes (byte[]). No entanto, para maior flexibilidade deve ser possível, nas várias aplicações a desenvolver, usar classes Java para definir as mensagens a transferir entre os diversos intervenientes. Por exemplo, a passagem de estado entre servidores do grupo *front-end*, após uma falha terá de ser feita com mensagens *Spread* que transportam a serialização de objetos no formato JSON. Para isso, devem usar a biblioteca *Gson* (disponível no repositório central Maven: <https://mvnrepository.com/artifact/com.google.code.gson/gson/2.8.9>) que de forma simples permite de forma flexível converter objetos em byte[] e de byte[] para objetos. Em anexo apresenta-se um exemplo de uso da referida biblioteca (*Gson*).
- Utilize uma VM do seu projeto GCP, onde deve instalar o *Docker* com o objetivo de funcionar como o segundo nível de *Edge/Fog* e onde se executará como container o servidor *RabbitMQ*
- Utilize 3 VMs onde, deve instalar o *middleware* *Spread*, e onde em cada uma se executa um servidor *Front-End*. Os consumidores do grupo *Event Processing Group* devem estar também distribuídos nas 3 VMs;
- As instâncias da aplicação *User*, e da aplicação de simulação dos sensores para publicação de eventos de velocidade, devem executar-se nas máquinas locais dos elementos do grupo (computadores pessoais).

	<p>Departamento de Engenharia de Eletrónica de Telecomunicações e Computadores Mestrado em Engenharia Informática e de Computadores (MEIC) Mestrado em Engenharia Informática e Multimédia (MEIM)</p>
<p>14-12-2021</p>	<p>Computação Distribuída (Inverno 21/22)</p>

Sugestões Gerais

- Qualquer questão ou dúvida sobre os requisitos deve ser discutida com o professor;
- Antes de começar a escrever código desenhe a arquitetura do sistema, os contratos de interação bem como os diagramas de interação mais importantes;
- Quando tiver dúvidas sobre os requisitos, verifique no site *Moodle* se existem “*Frequently Asked Questions*” com esclarecimentos sobre o trabalho.

 ISEL <small>INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA</small>	Departamento de Engenharia de Eletrónica de Telecomunicações e Computadores Mestrado em Engenharia Informática e de Computadores (MEIC) Mestrado em Engenharia Informática e Multimédia (MEIM)
14-12-2021	Computação Distribuída (Inverno 21/22)

Anexo: Exemplo de conversão: Objeto -> byte[] -> Objeto

```

public static void main(String[] args) {
    SomeClass someObject=new SomeClass();
    someObject.setId(5); someObject.setName("ABCD");
    someObject.setResults(new String[]{"abc","def"});
    System.out.println(someObject.toString());
    // converter objeto em string JSON
    Gson js=new GsonBuilder().create();
    String jsonString=js.toJson(someObject);
    System.out.println(jsonString);
    // converter string em byte[]
    byte[] binData=jsonString.getBytes(StandardCharsets.UTF_8);
    for (byte b : binData) System.out.print(b+" ");
    System.out.println();
    // binData pode ser enviado como mensagem em binário
    // em qualquer plataforma por exemplo RabbitMQ ou Spread,...
    // Receção de binData e deserialização para objeto
    String newJsonString=new String(binData, StandardCharsets.UTF_8);
    SomeClass newSomeObject=js.fromJson(newJsonString,SomeClass.class);
    System.out.println(newSomeObject.toString());
}

```

```

public class SomeClass {
    private int id;
    private String name;
    private String[] results;

    public SomeClass(){}

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String[] getResults() { return results; }
    public void setResults(String[] results) { this.results = results; }

    @Override
    public String toString() {
        String strResults="["; boolean first=true;
        for (String s : getResults()) {
            strResults+= first? "\""+s+"\"": ","+"\""+s+"\""; first=false;
        }
        strResults+="]";
        return "SomeClass("+getId()+","+getName()+","+strResults+")";
    }
}

```

```

<dependency>
<groupId>com.google.code.gson</groupId>
<artifactId>gson</artifactId>
<version>2.8.9</version>
</dependency>

```