15-10-2023	Computação Distribuída (Inverno 23/24)

Trabalho Prático de Avaliação (TPA1)

Objetivos: Utilização do *middleware* gRPC na implementação de um sistema com múltiplos servidores de suporte a balanceamento de carga na execução de tarefas de processamento de imagens. Cada tarefa é executada por um *container* Docker lançado pelos servidores gRPC.

Notas prévias:

- Nas aulas de 19 e 26 de Outubro de 2023, na turma diurna e a aula de 24, 31 de Outubro de 2022, na turma noturna a 2ª parte das aulas serão dedicadas para realizarem e esclarecerem dúvidas sobre o trabalho. No entanto, é pressuposto e faz parte dos ECTS da Unidade Curricular que têm de dedicar horas de trabalho fora das aulas para a realização do trabalho. 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* disponibilizados pelos respetivos professores);
- De acordo com as regras de avaliação definidas no slide 5 do conjunto *CD-01 Apresentação.pdf*, este trabalho tem um peso de 20% na avaliação final;
- 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. Por exemplo, podem juntar um ficheiro PDF ou tipo *readme.tx*t que explica os pressupostos que usaram bem como a forma de configurar e executar o sistema;
- Nas aulas de 16 de novembro de 2023 (turma diurna) e 14 de novembro de 2023 (turma noturna), cada grupo terá de apresentar e demonstrar, durante 10 a 15 minutos, para toda a turma, a funcionalidade do trabalho realizado;
- **A entrega no Moodle por cada grupo é até 13 de novembro de 2023 (23:59h)**

Pretende-se o desenvolvimento de um sistema distribuído (**ImageMarks**) onde existem múltiplos servidores para suportar distribuição de carga perante os múltiplos pedidos de múltiplos clientes, como se apresenta na Figura 1.

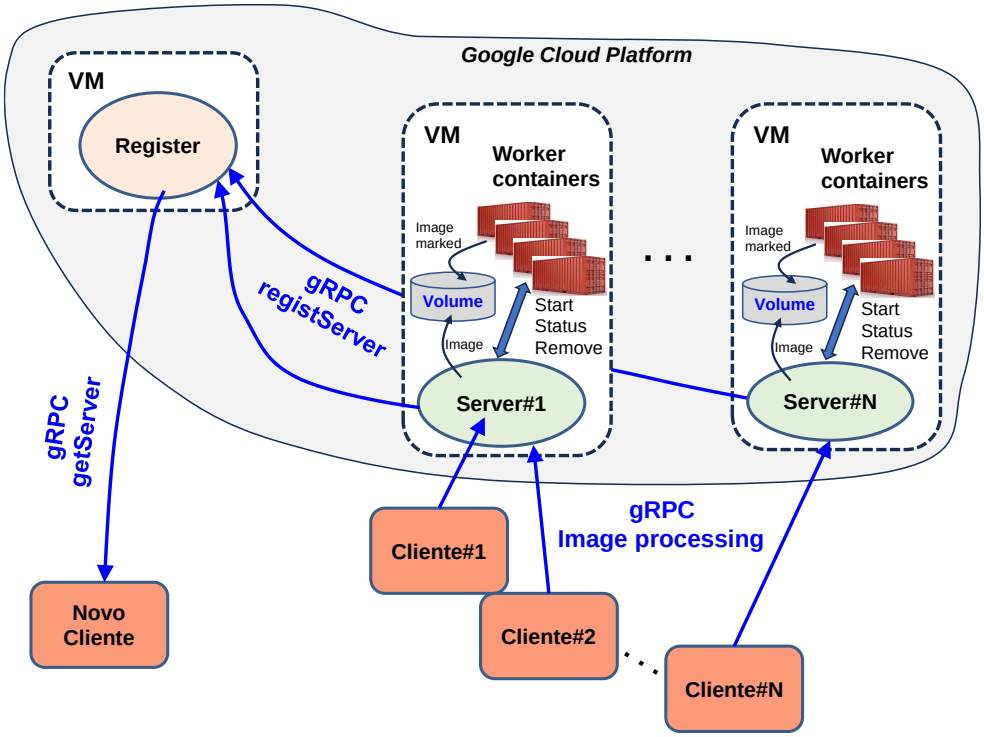

Figura 1 - Diagrama geral do sistema ***ImageMarks***

Requisitos funcionais

- Existe um servidor *Register* com localização (IP, porto) bem conhecida que funciona como serviço de registo dos N servidores de processamento de imagens. Os clientes recorrem a este servidor para obterem a localização (IP, porto) de um servidor (*Server#N*) de processamento de imagens, ao qual irão submeter pedidos;
- Uma aplicação Cliente usa dois contratos: i) operação de acesso ao servidor *Register* para obter a localização de um dos servidores registados; ii) operações que permitem realizar *upload* de imagens para um servidor, uma lista de *keywords* a serem marcadas na imagem inicial e a obtenção posterior da correspondente imagem marcada;
- Um Cliente pode submeter um conjunto de imagens e só posteriormente pedir ao servidor para obter (*download*) uma imagem marcada específica. Assim o servidor após receber uma imagem a processar deve retornar um identificador único que identifique a imagem. Através desse identificador o Cliente pode também interrogar o servidor se a imagem correspondente já foi processada, isto é, se já pode fazer o *download* da imagem marcada;
- A transferência dos ficheiros imagem (upload e download) entre os clientes e os servidores deve ser realizada por *streaming* de blocos, por exemplo, de 32 Kbyte;
- Cada marcação de imagens é realizada por um *container* onde se executa uma aplicação para o efeito (veja no Anexo 1, uma aplicação Java simples de marcar imagens);
- Quando o servidor recebe um ficheiro imagem deve armazená-la num volume da VM que seja acessível pelo *container* que irá processar a imagem. O *container* deve guardar a imagem marcada nesse mesmo volume;
- O servidor interage com a interface REST API do *daemon Docker* para executar, obter o estado de execução e remover os *containers*. Como se ilustra na aplicação Java do Anexo 2, a interação entre o servidor e o *daemon Docker* pode ser realizada através da biblioteca `com.github.docker-java` que encapsula a REST API;

Requisitos não funcionais

- Por questões de isolamento entre as partes (*Loose Coupling*) devem existir 3 contratos:
  1. Dos servidores *Server#N* para o servidor *Register* para registo dos múltiplos servidores de processamento de imagens;
  2. Do Cliente para o servidor *Register* para obtenção do *EndPoint* de um servidor *Server* para a partir daí o cliente submeter pedidos;
  3. Do Cliente para o servidor *Server para que o Cliente possa submeter pedidos*
- Assume-se que o servidor *Register* nunca falha e que está localizado num *EndPoint (IP, port)* sempre igual e bem conhecido;
- Cada VM pode disponibilizar um ou mais servidores Server;
- A construção do protótipo de demonstração deve ter pelo menos 3 servidores *Server* em execução em pelo menos duas VM, com pelo menos um Cliente em cada servidor;
- As várias instâncias da aplicação Cliente podem executar-se tanto nas máquinas locais dos elementos do grupo (computadores pessoais) como nas instâncias de VM onde se executam os vários servidores;
- Deve ser considerada na implementação e na demonstração que a qualquer momento podem ser adicionados dinamicamente novos servidores melhorando assim o balanceamento de carga.

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 dos serviços bem como os diagramas de interação mais importantes;
- Tenha em atenção o tratamento e propagação de exceções para assim o sistema ser mais fiável e permitir tratar algumas falhas;
- Quando tiver dúvidas sobre os requisitos, verifique no site *Moodle* se existem "*Frequently Asked Questions*" com esclarecimentos sobre o trabalho.

ANEXOS

Anexo 1: Aplicação Java que recebe um ficheiro imagem JPG e produz outro ficheiro com a imagem marcada com uma lista de palavras (*keywords*) passadas como parâmetro.

```
public class MarkApp {

    public static void main(String[] args) {
        // args[0] - image pathname; args[1] - image result pathname
        // args[2]...args[n] keywords to mark image
        String inputPath=args[0];
        String outputPath=args[1];
        ArrayList<String> keywords=new ArrayList<>();
        for (int i=2; i < args.length; i++)  keywords.add(args[i]);
        BufferedImage img = null;
        try {
            img = ImageIO.read(Path.of(inputPath).toFile());
            annotateImage(img, keywords);
            ImageIO.write(img, "jpg", Path.of(outputPath).toFile());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void annotateImage(BufferedImage img, ArrayList<String> keywords) {
        Graphics2D gfx = img.createGraphics();
        gfx.setFont(new Font("Arial", Font.PLAIN, 18));
        gfx.setColor(new Color(0x00ff00));

        String sentence="";
        for (String s : keywords) sentence += s+" ";
        gfx.drawString(sentence, 10, 20);
        Polygon poly = new Polygon();
        poly.addPoint(3, 3);
        poly.addPoint(10*sentence.length(), 3);
        poly.addPoint(10*sentence.length(), 25);
        poly.addPoint(3, 25);
        poly.addPoint(3, 3);
        gfx.setColor(new Color(0xff0000));
        gfx.draw(poly);
    }
}
```

Considerando que esta aplicação é usada na criação de uma imagem Docker de nome *markimage*, é possível executar um *container* de nome *contMarkimage* que marca a imagem *isel-view.jpg* existente na diretoria */usr/images* da VM associada como *volume* à diretoria do *container* */usr/datafiles* com as *keywords* *<2023/10/12, ISEL, aerial, view>* gerando a imagem *isel-view-marked.jpg*.


```
docker run -d -v /usr/images:/usr/datafiles –name countMarkimage markimage isel-view.jpg
isel-view-marked.jpg 2023-10-09 ISEL aerial view
```

Anexo II: A biblioteca *docker-java* que encapsula a REST API do *runtime* do Docker permite a qualquer aplicação Java executar comandos Docker, nomeadamente lançar e eliminar *containers* bem como obter o estado `{running, exited,…}` de execução de um *container*.

A seguir apresenta-se uma aplicação Java que executa um *container* de forma equivalente ao comando atrás ilustrado, verifica o seu estado e termina o *container*.

```
public class DockerAPI {
    public static void main(String[] args) {
        // arg0 windows: tcp://localhost:2375   // arg0 linux: unix:///var/run/docker.sock
        // arg1 : container name
        // arg2 : volume name or filesystem directory
        try {
           String HOST_URI = args[0];   String containerName = args[1];
           String pathVolDir = args[2]; String imageName = args[3];
           List<String> command=new ArrayList<>();
           for (int i=4; i < args.length; i++) command.add(args[i]);
           DockerClient dockerclient = DockerClientBuilder
                              .getInstance()
                              .withDockerHttpClient(
                                        new ApacheDockerHttpClient.Builder()
                                                .dockerHost(URI.create(HOST_URI)).build()
                               )
                               .build();
           HostConfig hostConfig = HostConfig.newHostConfig()
                     .withBinds(new Bind(pathVolDir, new Volume("/usr/datafiles")));
           CreateContainerResponse containerResponse = dockerclient
                     .createContainerCmd(imageName)
                     .withName(container Name)
                     .withHostConfig(hostConfig)
                     .withCmd(command)
                     .exec();
           System.out.println("ID:" + containerResponse.getId());
           dockerclient.startContainerCmd(containerResponse.getId()).exec();

           InspectContainerResponse inspResp = dockerclient
                     .inspectContainerCmd(containerName).exec();
           System.out.println("Container Status: " + inspResp.getState().getStatus());
           // if container is running
           dockerclient.killContainerCmd(containerName).exec();
           // remove container
           dockerclient.removeContainerCmd(containerName).exec();
        } catch (Exception ex) {
                ex.printStackTrace();
        }
    }
}
```

A aplicação anterior tem as seguintes dependências Maven:

```
<dependency>
    <groupId>com.github.docker-java</groupId>
    <artifactId>docker-java</artifactId>
    <version>3.3.3</version>
</dependency>
<dependency>
    <groupId>com.github.docker-java</groupId>
    <artifactId>docker-java-transport-httpclient5</artifactId>
    <version>3.2.14</version>
</dependency>
```