Laboratório 3 - Desafio complementar

Objetivos:

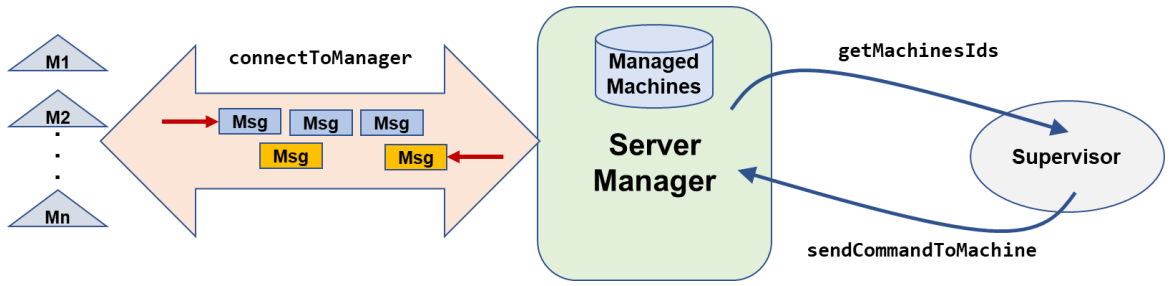
- Desenvolver serviços distribuídos com tecnologia *Google Remote Procedure Call* (gRPC)
- Chamadas com *stream* de cliente e *stream* de servidor e mensagens *oneof*

Considere um cenário de uma fábrica onde existem múltiplas máquinas (aplicação cliente) que estão em comunicação com uma aplicação gestora (aplicação servidora). Durante o arranque de cada máquina, esta liga-se ao servidor chamando uma operação gRPC duplex (streaming dos dois lados) por forma a que, tanto a máquina, como o servidor *Manager* possam enviar mensagens entre si:

- A máquina envia periodicamente mensagens com o valor da temperatura de funcionamento;
- Em qualquer altura, o servidor pode enviar à máquina comandos de controlo, por exemplo de *stop/restart*, ou pares (key, value) de configuração;
- O servidor não necessita de armazenar os valores de temperatura, verificando unicamente se os mesmos estão acima de um valor máximo admissível (MAXTEMP). No caso do valor estar acima de MAXTEMP o servidor deve enviar à máquina uma mensagem de controlo para esta se desligar (*stop*);
- Quando a máquina receber do servidor um comando de controlo específico para o efeito, a máquina envia para o servidor a sua configuração corrente como um conjunto de pares (key, value);

Como se indica na figura quando uma máquina se conecta ao servidor, este deve guardar numa estrutura de dados (*Managed Machines*), indexada pelo identificador da máquina, as informações relativas a poder posteriormente enviar-lhe mensagens.

Existe também uma aplicação de supervisão que pode obter os identificadores das máquinas sob gestão no servidor e, em seguida, enviar comandos para que o servidor os envie para a máquina indicada no comando, por exemplo, novas configurações ou ordens de *stop* ou *restart*..



Implemente as aplicações: (cliente-simulação de máquina), (server-manager) e (cliente-supervisor), considerando os dois contratos (entre as máquinas e o servidor, e entre a aplicação de supervisão e o servidor) que se indicam na página seguinte.
Note que o servidor vai ter de disponibilizar dois serviços gRPC, um para cada contrato.

```
ManagedMachines manMach=new ManagedMachines(); // partilha de estado e lógica de aplicação

io.grpc.Server svc = ServerBuilder
   .forPort(svcPort)
   .addService(new ServerMachine(manMach))
   .addService(new ServerSupervisor(manMach))
   .build();
svc.start();
// . . .
svc.awaitTermination();
```

Contratos:

```
service MachinesManagerContract {

   rpc connectToManager (stream Information)
                        returns (stream Information);
}
```

```
message Void {  }
message MachineID {   int32 ID = 1; }
// generic message between Machine-Manager
message Information {
      MachineID mID = 1;
      oneof MsgOptions {
         double temperature = 2;
         Control ctl = 3;
         Config conf = 4;
      }
}
message Control {
   int32 ctlNumber = 1;
   string ctltext = 2;
}
message Config {
   map<int32, string> configPairs = 1;
}
```

Nota: O nome da mensagem que transporta o identificador da máquina é diferente nos dois contratos propositadamente para evitar conflito de nomes entre os stubs dos dois contratos.

```
service SupervisorManager {
   // obter os Ids de todas as máquinas
   rpc getMachinesIds(Void) returns (AllMachineIsD);
   rpc sendCommandToMachine(Command) returns (Void);
}
```

```
message Void { }
message MachID { int32 ID = 1; }

message AllMachineIsD {
 repeated MachID IDs = 1;
}

message Command {
   MachID id =1;
   int32 ctlNumber = 2;
   string ctltext = 3;
}
```

Como uma mensagem *Information* pode transportar diferentes dados (**oneof**), note que a classe gerada pelo compilador de *protobuf* dá suporte para obtermos o tipo que a mensagem transporta.

```
// no stream observer que recebe as mensagens na máquina
public void onNext(Information msg) {
  if (msg.getMsgOptionsCase() == Information.MsgOptionsCase.CTL) {
      // Machime received control
      System.out.println("Control ("+msg.getCtl().getCtlNumber()+":"+msg.getCtl().getCtltext()+")");
  } else if(msg.getMsgOptionsCase() == Information.MsgOptionsCase.CONF) { // Machime received config
      Map<Integer,String> config=msg.getConf().getConfigPairsMap();
      config.forEach((key, cmd) -> {System.out.println(key+":"+cmd);});
  }
}
```