Laboratório de Desenvolvimento de Aplicações Móveis e Distribuídas
gRPC

Prof. Hugo Bastos de Paula

gRPC

- Sistema RPC multiplataforma da Google
- Open source on Github:http://grpc.io
- Independência de linguagem: Android/Java, C#/.NET, C++, Dart , Go, Java, Kotlin/JVM, Node.js , Objective-C , PHP, Python, Ruby
- Utiliza Google Protobuffers para ser independente da carga
- Robusto: timeouts, atrasos, balanceamento de carga, cancelamentos, etc.)

Aplicações do gRPC

Nuvens públicas e privadas

Clientes e servidores em plataformas:

- Web, Móvel, Nuvem e IoT.

API síncrona:

- cliente espera resposta
- Mensagem ou stream

API assíncrona:

- Thread do cliente ou do servidor utilizam CompletionQueue para esperar por eventos.

Fluxo de trabalho

- Instalação:
  - apt-get install protobuf-compiler
  - pip install grpcio grpcio-tools
- Escrever as definições das mensagens: protos
- Usar o protoc. Para gerar as interfaces dos serviços e os stubs.
- Implementar os serviços no servidor.
- Instanciar o stub no cliente.
- Testar e disponibilizar.

Protocol Buffers

- IDL (*Interface Definition Language*): Descreve estrutura de dados para troca de informação
- Modelo de dados: estrutura o formato das mensagens de requisição e resposta
- Formato para transmissão na rede
- Compilação para geração dos stubs

python -m grpc_tools.protoc -I . --python_out=. --grpc_python_out=. arquivo.proto

```
message SubscribeRequest {
  string topic = 1;
}
message Event {
  string details = 1;
}
service Topics {
  rpc Subscribe(SubscribeRequest)
  returns (stream Event);
}
```

Definição do serviço

```
service RouteGuide {

// Obtains the feature at a given position.
rpc GetFeature(Point) returns (Feature) {}

  // Obtains the Features available within the given Rectangle. Results are
  // streamed rather than returned at once (e.g. in a response message with a
  // repeated field), as the rectangle may cover a large area and contain a
  // huge number of features.
  rpc ListFeatures(Rectangle) returns (stream Feature) {}

  // Accepts a stream of Points on a route being traversed, returning a
  // RouteSummary when traversal is completed.
  rpc RecordRoute(stream Point) returns (RouteSummary) {}

 // Accepts a stream of RouteNotes sent while a route is being traversed,
 // while receiving other RouteNotes (e.g. from other users).
 rpc RouteChat(stream RouteNote) returns (stream RouteNote) {}
}
// Points are represented as latitude-longitude pairs in the E7 representation
// (degrees multiplied by 10**7 and rounded to the nearest integer).
// Latitudes should be in the range +/- 90 degrees and longitude should be in // the range +/- 180 degrees (inclusive).
message Point {
  int32 latitude = 1;
  int32 longitude = 2;
}
```

Chamada simples: mensagem -> mensagem

Chamada request-streaming RPC: mensagem → stream

Chamada response-streaming RPC: stream → mensagem

Chamada bidirecional streaming RPC: stream → stream

Definição da mensagem

https://grpc.io/docs/languages/python/basics/

RPC simples

```
def GetFeature(self, request, context):
  feature = get_feature(self.db, request)
  if feature is None:
    return route_guide_pb2.Feature(name="", location=request)
  else:
    return feature
```

https://grpc.io/docs/languages/python/basics/

Response-streaming RPC

```
def ListFeatures(self, request, context):
  left = min(request.lo.longitude, request.hi.longitude)
  right = max(request.lo.longitude, request.hi.longitude)
  top = max(request.lo.latitude, request.hi.latitude)
  bottom = min(request.lo.latitude, request.hi.latitude)
  for feature in self.db:
    if (feature.location.longitude >= left and
        feature.location.longitude <= right and
        feature.location.latitude >= bottom and
        feature.location.latitude <= top):
      yield feature
```

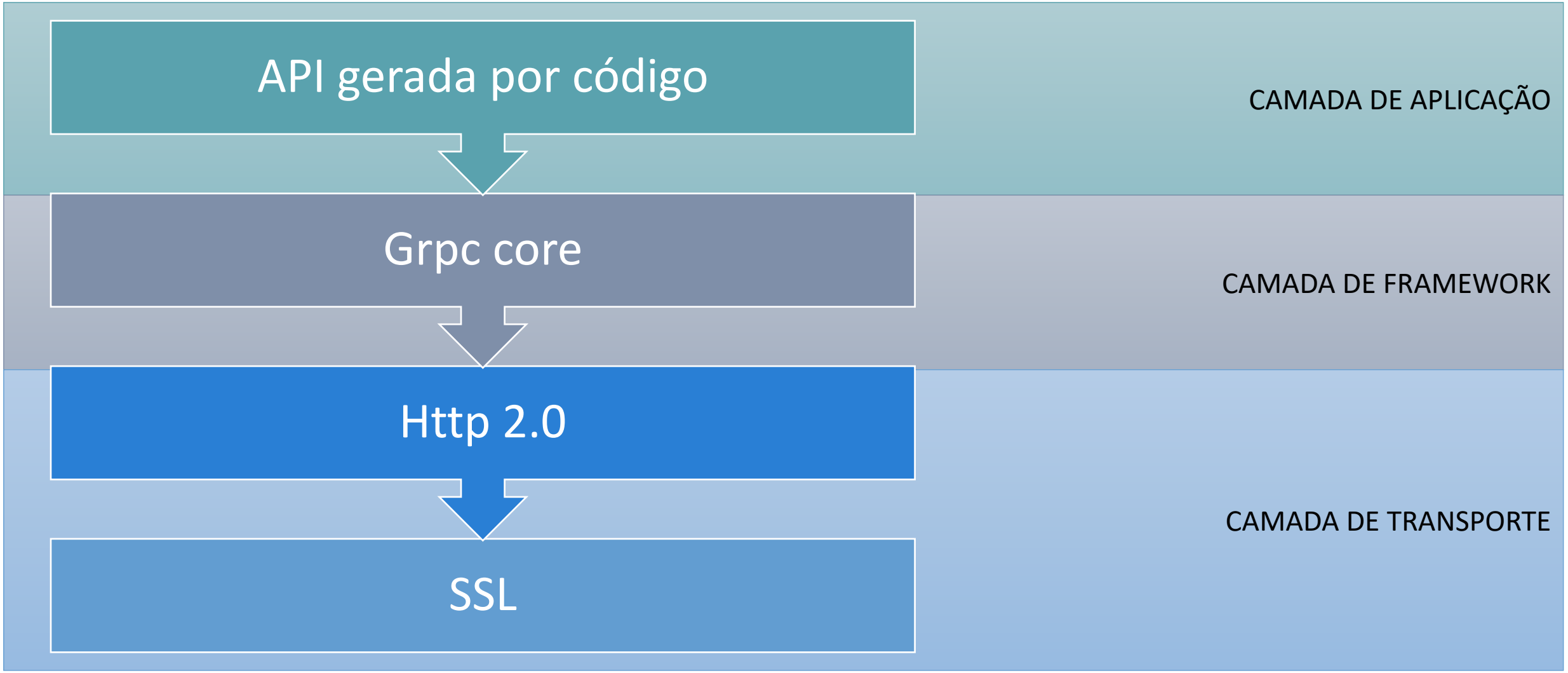
Request-streaming RPC

```
def RecordRoute(self, request_iterator, context):
  point_count = 0
  feature_count = 0
  distance = 0.0
  prev_point = None

  start_time = time.time()
  for point in request_iterator:
    point_count += 1
    if get_feature(self.db, point):
      feature_count += 1
    if prev_point:
      distance += get_distance(prev_point, point)
    prev_point = point

  elapsed_time = time.time() - start_time
  return route_guide_pb2.RouteSummary(point_count=point_count,
                                      feature_count=feature_count,
                                      distance=int(distance),
                                      elapsed_time=int(elapsed_time))
```

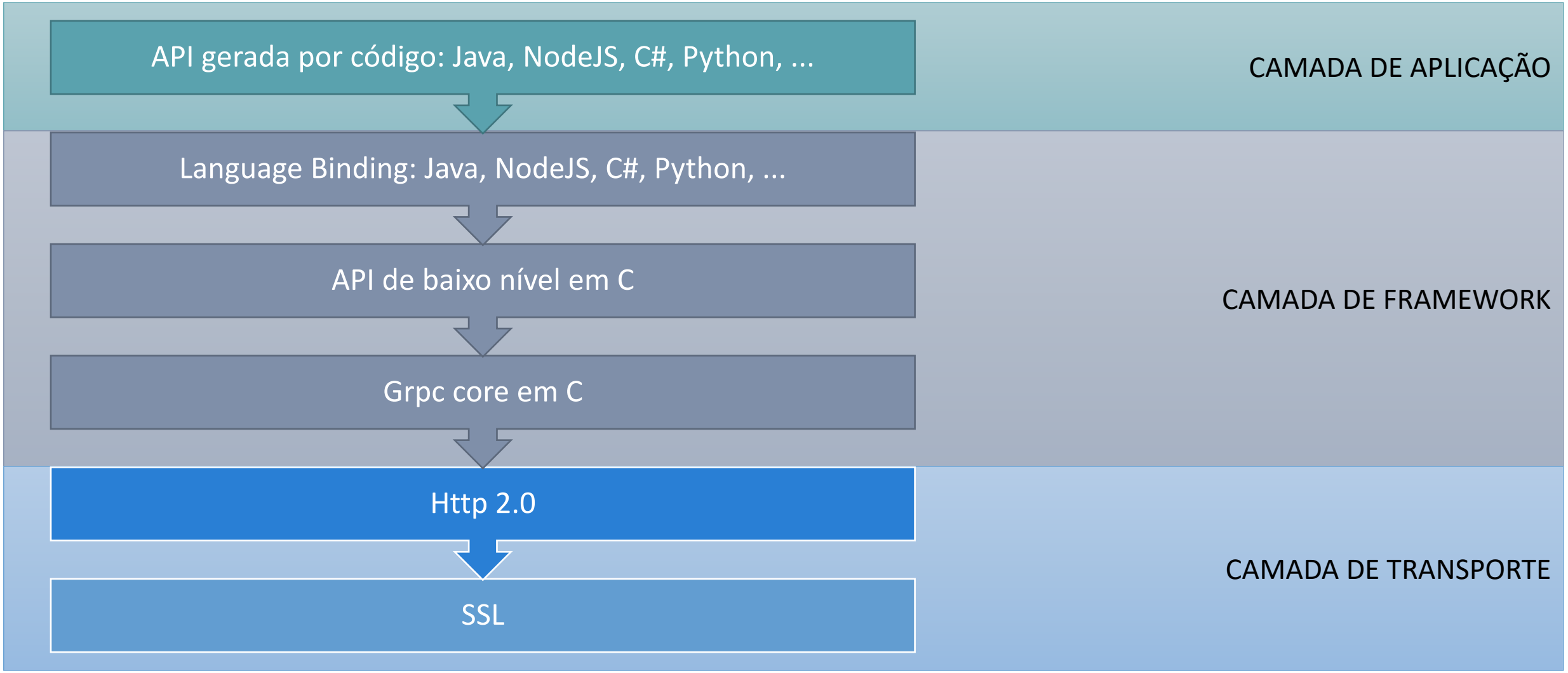
Bidirectional streaming RPC

```
def RouteChat(self, request_iterator, context):
  prev_notes = []
  for new_note in request_iterator:
    for prev_note in prev_notes:
      if prev_note.location == new_note.location:
        yield prev_note
    prev_notes.append(new_note)
```

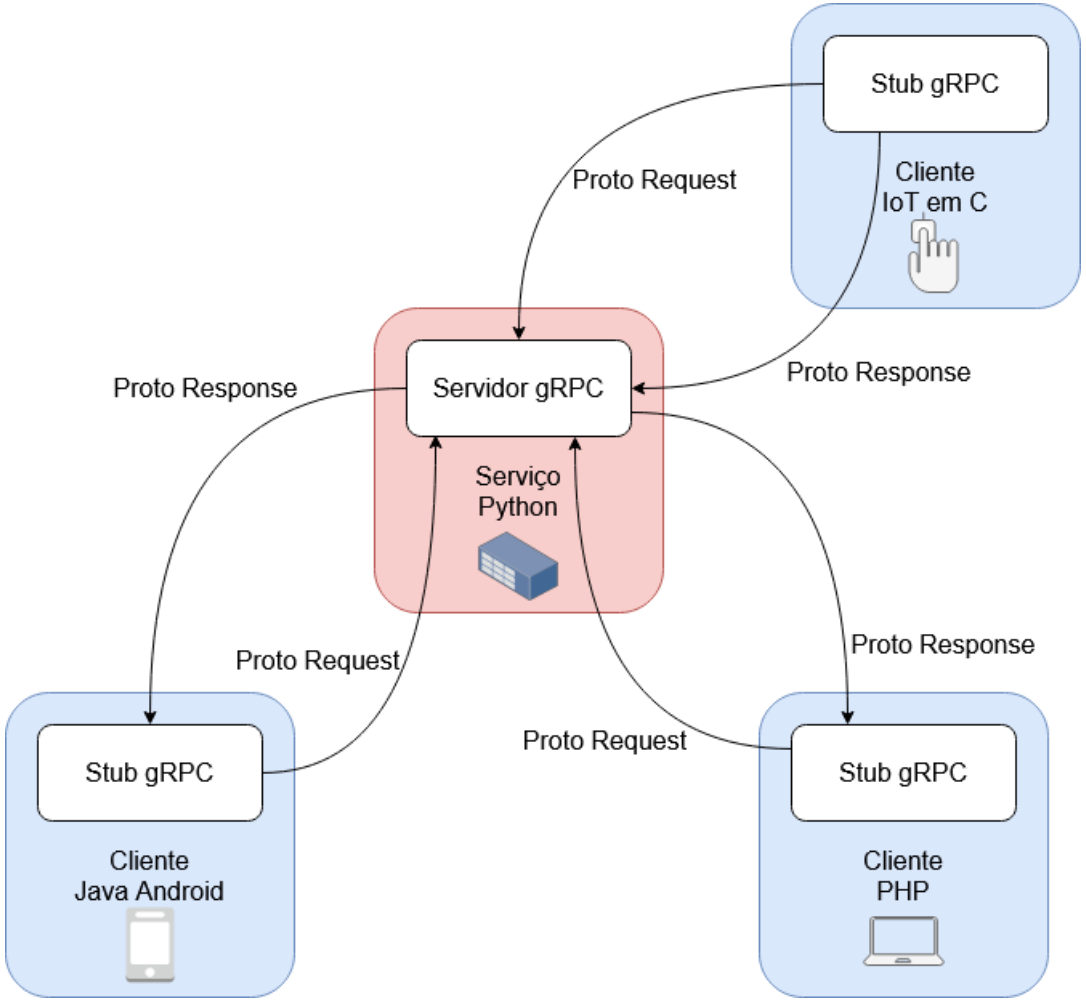
Aquitetura do gRPC



Aquitetura do gRPC

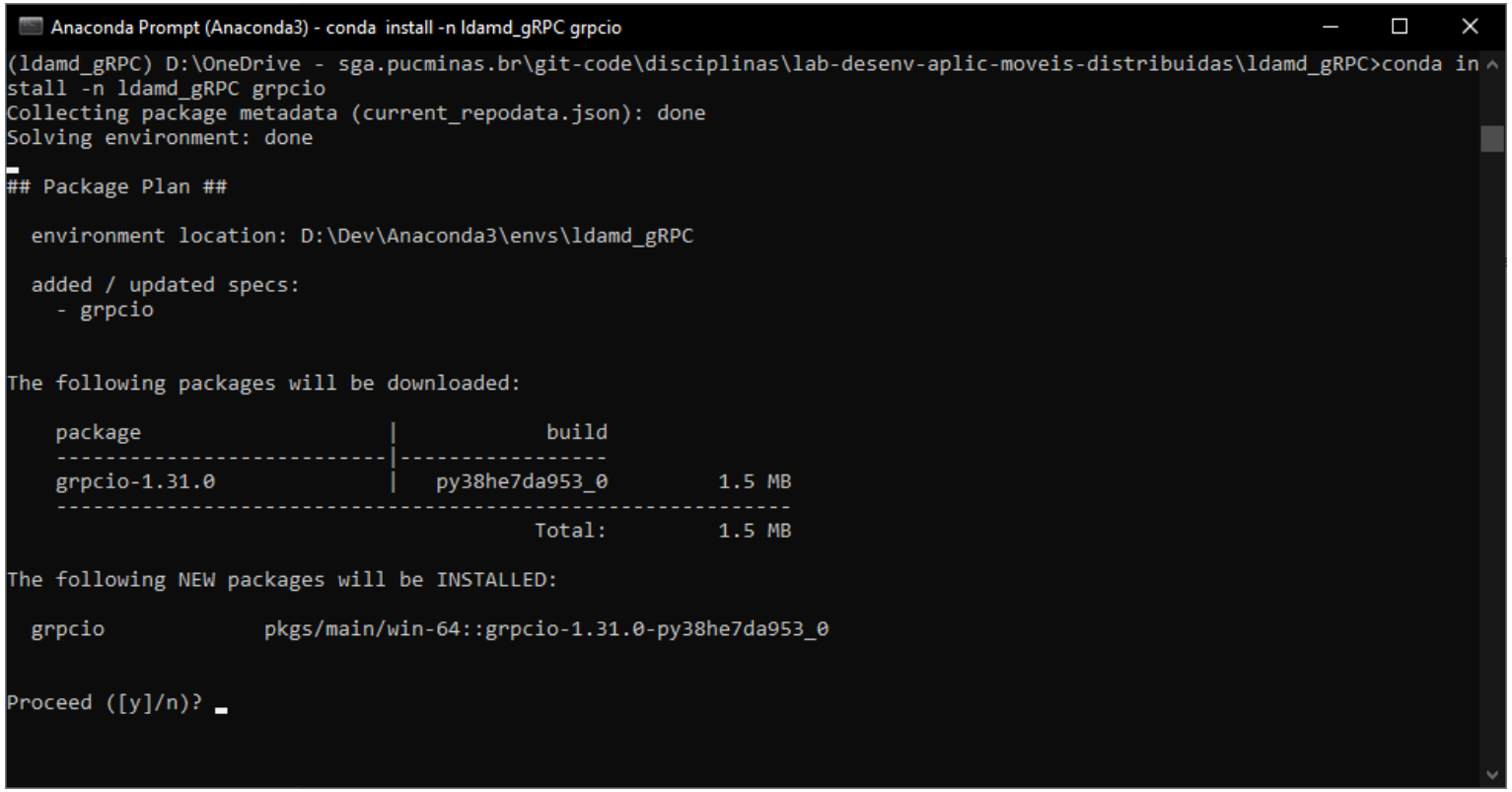


Exemplo de arquitetura Cliente/Servidor com gRPC



Criando uma aplicação gRPC em Python

- Criar o ambiente:
  - conda create -n ldamd_gRPC python=3.8 anaconda
  - source activate ldamd_gRPC
  - conda install –n ldamd_gRPC grpcio



Criando uma aplicação gRPC em Python

- gRPC tools inclui o compilador protoc., que irá gerar os códigos para o cliente e o servidor.
  - conda install -n ldamd_gRPC grpcio-tools
  - git clone -b v1.31.0 https://github.com/grpc/grpc

- Execute o servidor e depois o cliente.

```
(ldamd_gRPC) D:\OneDrive - sga.pucminas.br\git-code\disciplinas\ldamd\ldamd_gRPC\grpc\examples\python\helloworld>python
greeter_server.py
```

```
(ldamd_gRPC) D:\OneDrive - sga.pucminas.br\git-code\disciplinas\ldamd\ldamd_gRPC\grpc\examples\python\helloworld>python
greeter_client.py
Greeter client received: Hello, you!
```

Gerando os stubs

python -m grpc_tools.protoc

-I . ← Caminho para a IDL

--python_out=. ← Saída dos stubs

--grpc_python_out=. ← Saída para cliente/server

helloworld.proto ← Arquivo IDL do serviço

- Arquivos gerados:
  - helloworld_pb2.py – request/response
  - helloworld_pb2_grpc.py – cliente/server stubs

```
syntax = "proto3";

option java_multiple_files = true;
option java_package = "io.grpc.examples.helloworld";
option java_outer_classname = "HelloWorldProto";
option objc_class_prefix = "HLW";

package helloworld;

// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
  // Sends another greeting
  rpc SayHelloAgain (HelloRequest) returns (HelloReply) {}

}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```