Microservices with gRPC

(Como aplicações podem se comunicar com outras)

Rafael Araujo dos Santos - Software Developer

Github: @araujorafael

1.

O que extamente é o gRPC?

O que o gRPC tem a oferecer?

- **Alta performance**
- **Segurança**
- **Load Balancing**
- **Serviços ao invés de objetos, mensagens ao invés de Referências**
- **Códigos de erro padronizados**
- **Lameducking**
- **Payload Agnostic**
- **Duplex Streamexing**
- **Código de cliente auto gerado***

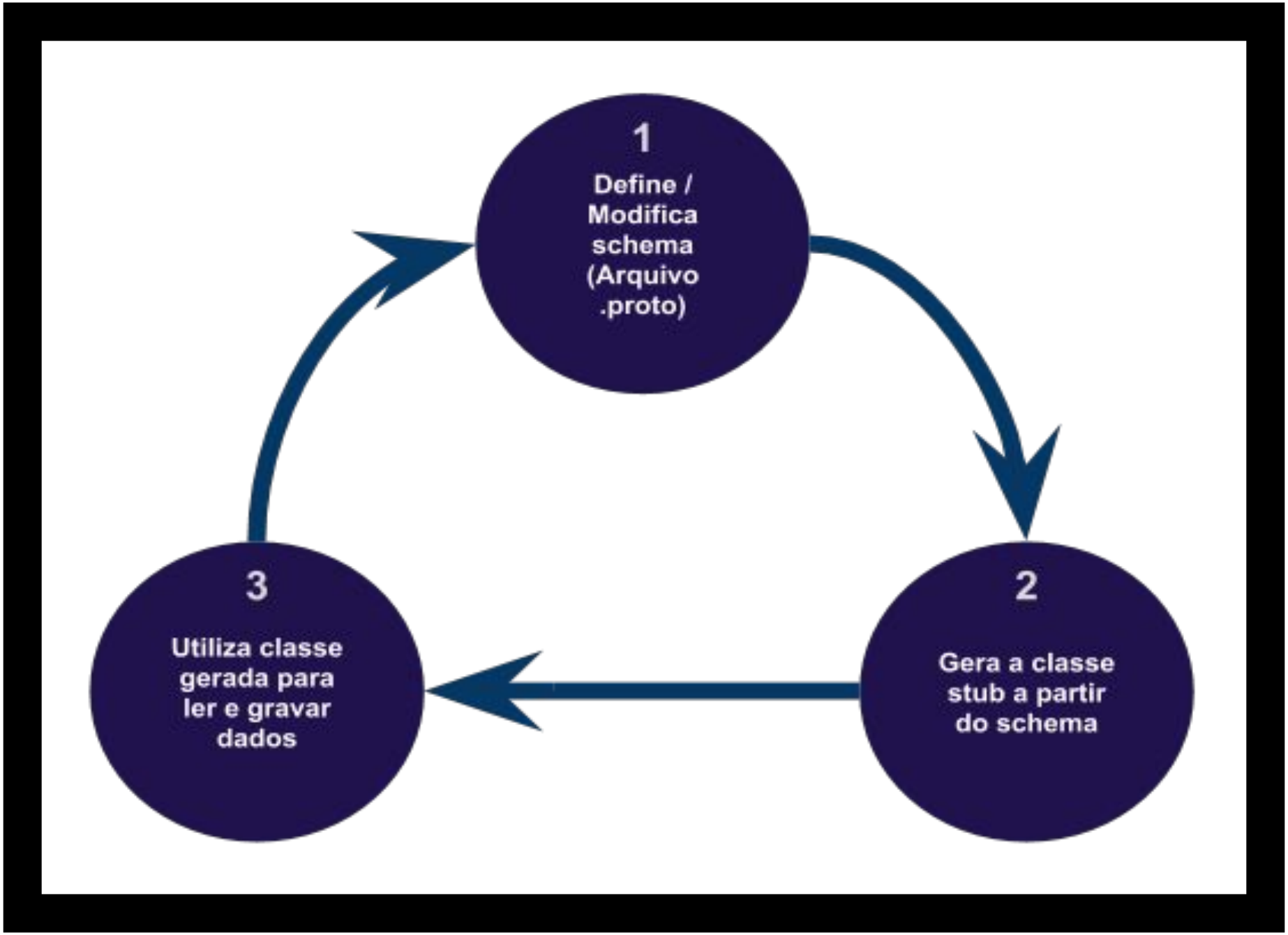
Mas também não é uma bala de prata...

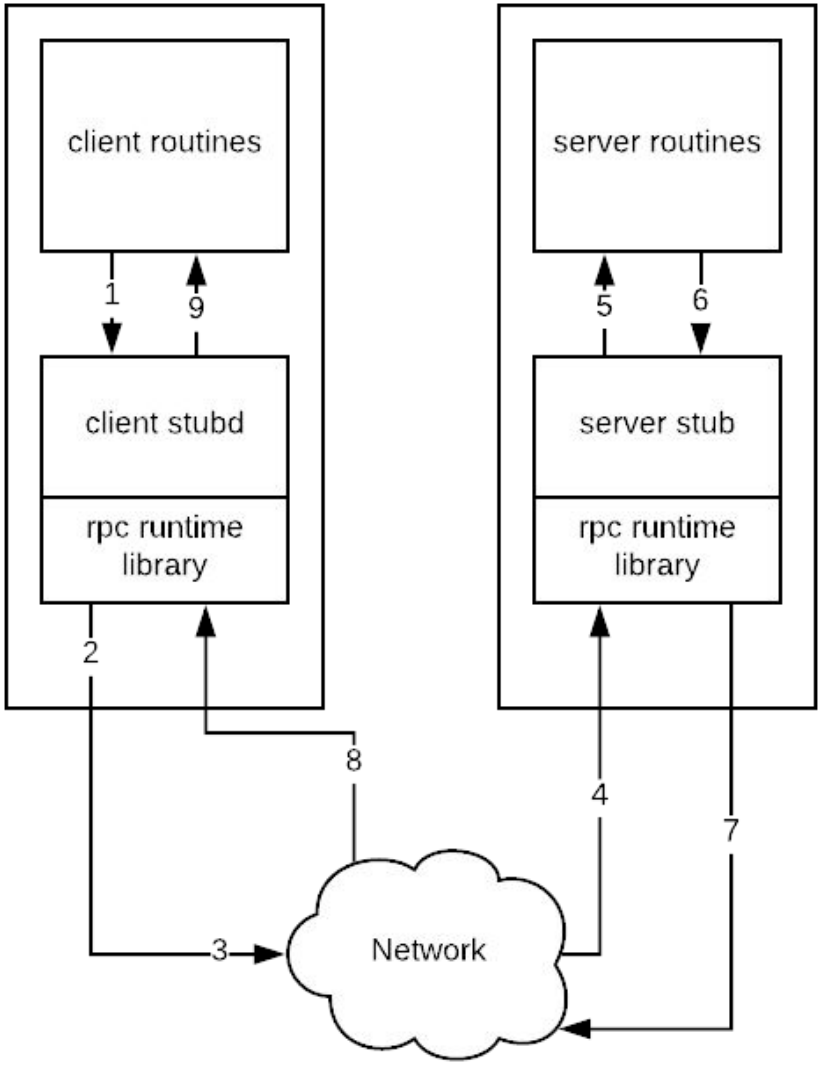
- **Não há suporte de browser***
- **Endpoints não podem ser testados com *postman* ou *curl***
- **Status customizados podem gerar conflitos**

Mas e o padrão REST?

Protobuf

- **Schemas**
- **Menos código duplicado**
- **Extensibilidade**
- **Fortemente tipado**
- **Interoperabilidade entre linguagens**
- **Geração de *“stubs”* clients e servers**

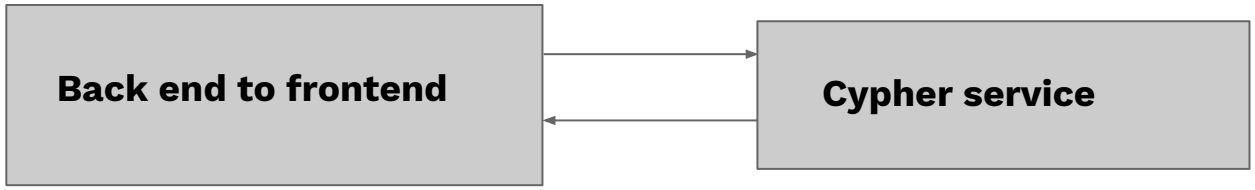

1
Define / Modifica schema (Arquivo .proto)
2
Gera a classe stub a partir do schema
3
Utiliza classe gerada para ler e gravar dados


client routines
server routines
1
9
5
6
client stubd
server stub
rpc runtime library
rpc runtime library
2
8
4
7
3
Network

2.

Como funciona na prática?

Serviços que vamos implementar:



Instalando o Protoc

```
$ brew install protobuff
```

Criando nosso arquivo proto...

```
syntax = "proto3";

package cypher;
option go_package = "proto";

service CypherService {
        rpc Encode(CypherRequest) returns (CypherResponse);
        rpc Decode(CypherRequest) returns (CypherResponse);
}

message CypherRequest {
        CypherOptions options = 1;
        string id = 2;
}

message CypherResponse {
        string encrypedText = 1;
        CypherOptions options = 2;
}

message CypherOptions {
        string text = 1;
        int32 shift = 2;
}
```

Gerando os stubs de client/server

```
$ protoc proto/cypher.proto --proto_path=./proto \
> --go_out=plugins=grpc:${PWD}/proto
```

```
...
// CypherServiceServer is the server API for CypherService service.
type CypherServiceServer interface {
  Encode(context.Context, *CypherRequest) (*CypherResponse, error)
  Decode(context.Context, *CypherRequest) (*CypherResponse, error)
}

func RegisterCypherServiceServer(s *grpc.Server, srv CypherServiceServer) {
  s.RegisterService(&_CypherService_serviceDesc, srv)
}
...
```

```go
package remoteprocedurecall

import (
  ...
  "google.golang.org/grpc"
  "google.golang.org/grpc/keepalive"
)

type RPCServer struct {
  listener net.Listener
  Grpc     *grpc.Server
}

...

func NewServer(port string) *RPCServer {
  listener, err := net.Listen("tcp", port)
  if err != nil {
     return nil
  }
  return &RPCServer{
     listener: listener,
     Grpc:     grpc.NewServer(grpc.KeepaliveEnforcementPolicy(kaep), grpc.KeepaliveParams(kasp)),
  }
}

func (this *RPCServer) Start() error {
  return this.Grpc.Serve(this.listener)
}
```

Arquivo main.go do Cypher Service

```go
package main

import (
  "github.com/labstack/gommon/log"
  "grpc-talk/libs/remoteprocedurecall"
  "grpc-talk/cypher/rpc"
)

func main() {
  port := ":8001"
  rpcServer := remoteprocedurecall.NewServer(port)
  if rpcServer == nil {
     log.Fatal("Nao consigo escutar na porta:", port)
  }

  rpc.NewCypherServer(rpcServer.Grpc)

  log.Info("Listening on", port)
  log.Fatal(rpcServer.Start())
}
```

```go
package rpc

import (
  "context"
  "google.golang.org/grpc"
  pb "grpc-talk/proto"
)

type CypherServer struct{}

// NewCypherServer cria o serviço que contera as funçoes do contrato
func NewCypherServer(s *grpc.Server) *CypherServer {
  server := &CypherServer{}
  if s != nil {
     pb.RegisterCypherServiceServer(s, server)
  }
  return server
}

// Encode encripta string de acordo com cifra de cesar
func (cypher CypherServer) Encode(ctx context.Context, req *pb.CypherRequest) (*pb.CypherResponse, error) {
  options := req.GetOptions()
  encode := rotate(options.GetText(), int(options.GetShift()))
  resp := &pb.CypherResponse{
     EncrypedText: encode,
     Options:      options,
  }
  return resp, nil
}

// Decode descripta string de acordo com cifra de cesar
func (cypher CypherServer) Decode(ctx context.Context, req *pb.CypherRequest) (*pb.CypherResponse, error) {...}
func rotate(text string, shift int) string {...}
```

Implementando o rpc_client

```go
import (
  ...
  "google.golang.org/grpc"
  "google.golang.org/grpc/keepalive"
)

// Client contains rpc connection confs
type Client struct {
  address    string
  connection *grpc.ClientConn
  ctx        context.Context
}

// NewClient Create a new rpc client
func NewClient(address string) Client {
  return Client{
     address: address,
     ctx:     context.Background(),
  }
}
```

```go
// Connect connect to another aplication via rpc
func (cl Client) Connect() *grpc.ClientConn {
  k := keepalive.ClientParameters{}
  opts := grpc.WaitForReady(false)
  conn, err := grpc.Dial(
     cl.address,
     grpc.WithInsecure(),
     grpc.WithKeepaliveParams(k),
     grpc.WithDefaultCallOptions(opts),
  )
  if err != nil {...}
  cl.connection = conn
  return conn
}
```

Adiciona os imports necessários ao main.go de nosso BFF...

```
import (
  rpc "grpc-talk/bff/rpc"
  pb "grpc-talk/proto"

  "github.com/gin-gonic/gin"
  "google.golang.org/grpc"
)

type CesarMessage struct {
  Message string `json:"message" binding:"required"`
  Shift   int    `json:"shift,omitempty" binding:"required"`
}

func getRpcConn(url string) *grpc.ClientConn {
  client := rpc.NewClient(url)
  return client.Connect()
}

func setupRouter() *gin.Engine {
  conn := getRpcConn("localhost:8001")
  cypherConn := pb.NewCypherServiceClient(conn)
  r := gin.Default()
  ...
}
```

```go
r.GET("/cesarcypher/encode", func(c *gin.Context) { // Adicionando rota e chamando stub
  // recebendo dados
  data := new(CesarMessage)
  err := c.BindJSON(data)
  if err != nil {
     c.AbortWithStatusJSON(http.StatusBadRequest, gin.H{"error": "Json incorrect values"})
     return
  }
  // Convertendo para formato do proto
  message := &pb.CypherRequest{
     Options: &pb.CypherOptions{
        Text:  data.Message,
        Shift: int32(data.Shift),
     },
  }
  // Executando a "função remota"
  encoded, err := cypherConn.Encode(c, message)
  if err != nil {
     c.AbortWithStatusJSON(http.StatusBadRequest, gin.H{"error": "Parsing error"})
     return
  }
  // Retornando resultados
  encodedMessage := CesarMessage{Message: encoded.GetEncrypedText()}
  c.JSON(http.StatusOK, gin.H{"data": encodedMessage})
})
```

Links uteis:

https://grpc.io/
https://github.com/golang/protobuf/tree/master/protoc-gen-go
https://developers.google.com/protocol-buffers/
https://github.com/protocolbuffers/protobuf

Obrigado!