KubeCon | CloudNativeCon

North America 2019

KubeCon | CloudNativeCon

North America 2019

gRPC : An Introduction

Jayant Kolhe & Eric Anderson

Audience Poll

Summary of gRPC Talks

- Beginner Level:
  - gRPC: An Introduction: Jayant Kolhe & Eric Anderson
- Beginner/Intermediate Level:
  - Design Decisions for Communication Systems: Eric Anderson @3:20 pm on Tue. Nov 19
- Expert Level:
  - Prevent your service from taking over itself: Lidi Zheng @ 11:50 am on Wed. Nov. 20
- Intermediate Level:
  - Securing your services in Authentication, Authorization, and RBAC in gRPC: Luis Pabon @2:25 pm on Thurs. Nov. 21

What is gRPC?

gRPC stands for **g**RPC **R**emote **P**rocedure **C**alls.

A high performance, standards-based, open source general purpose feature-rich RPC framework

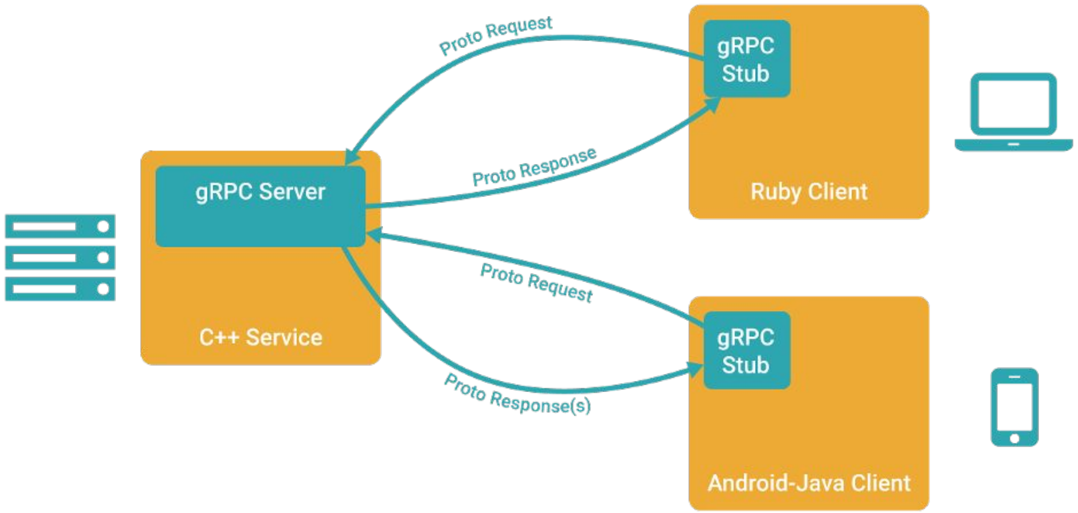
CNCF's RPC framework for building cloud native apps, next generation of Stubby RPC used in Google.

Actively developed and production-ready, current version is 1.25.

A high performance, open-source universal RPC framework

gRPC With Protocol Buffers

- Define a service in a .proto file using Protocol Buffers IDL
- Generate server and client stub code using the protocol buffer compiler
- Extend the generated server class in your language to fill in the logic of your service
- Invoke it using the generated client stubs



Quick Overview: Protocol Buffers

- gRPC Lingua Franca for serializing data: RPCs and storage
- Binary data representation
- Structures can be extended and maintain backward compatibility
- Code generators for many languages
- Strongly typed
- Not required for gRPC, but very handy

```
syntax = “proto3”;

message Person {
  string name = 1;
  int32 id = 2;
  string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    string number = 1;
    PhoneType type = 2;
  }

  repeated PhoneNumber phone = 4;
}
```

Lets walk through an example

- Route Guide Example

```
Example: RouteGuide : grpc/grpc/examples

Messages:
// Message Objects
// Point: location (lattitude, longitude)
// Feature: Feature at a location
// RouteNote: Note sent from point along a route

Service:
// Interface exported by the server
// Contains Methods for:
// GetFeature: Obtains the feature
//             at a given position.
// RouteChat: send RouteNotes while travelling
//            across a route and receive those
//            from other asynchronously
```

Start with a Protocol Buffer

- Start with defining messages you want to send

```
syntax = “proto3”;

message Point {
  int32 latitude = 1;
  int32 longitude = 2;
}

message Feature {
  string name = 1;
  Point location = 2;
}

message RouteNote {
  Point location = 1;
  string message = 2;
}
```

Add Service Definition

- **Unary RPC:**
  - Client sends a request
  - Server sends a response
- **Client Streaming RPC**:
  - Client sends multiple messages
  - Server sends one response
- **Server Streaming RPC**:
  - Client sends one message
  - Server sends multiple messages
- **Bidi Streaming RPC**:
  - Client and Server can independently send multiple messages to each other

```
syntax = “proto3”;

message Point {
  int32 latitude = 1;
  int32 longitude = 2;
}

message Feature {
  string name = 1;
  Point location = 2;
}

message RouteNote {
  Point location = 1;
  string message = 2;
}

service RouteGuide {
  rpc GetFeature(Point) returns (Feature);
  rpc RouteChat(stream RouteNote) returns
                       (stream RouteNote);
}
```

Generate code for your application

Code generator converts .proto idiomatically to your language.

- Idiomatic objects for messages
- with getters and setters for the message types
- And as an abstract interface class for the service type

```
syntax = “proto3”;

message Point {
  int32 latitude = 1;
  int32 longitude = 2;
}

message Feature {
  string name = 1;
  Point location = 2;
}

message RouteNote {
  Point location = 1;
  string message = 2;
}

service RouteGuide {
  rpc GetFeature(Point) returns (Feature);
  rpc RouteChat(stream RouteNote) returns
                          (stream RouteNote);
}
```

Generated Code Snippet

```
class RouteGuide {

  class Stub : public StubInterface{

   Public:

    Status GetFeature(ClientContext* context, const Point& request, Feature* response) override;
    unique_ptr<ClientReaderWriter<RouteNote,RouteNote>> RouteChat(ClientContext* context) override;
  };

  static unique_ptr<Stub> NewStub(const shared_ptr<ChannelInterface>& channel,
                                  const StubOptions& options = StubOptions());

  class Service : public ::grpc::Service {

   Public:

    virtual Status GetFeature(ServerContext* context, const Point& request, Feature* response);
    virtual Status RouteChat(ServerContext* context, ServerReaderWriter<RouteNote, RouteNote>* stream);
  };
}
```

Generated Code Snippet

```
class RouteGuide {

  class Stub : public StubInterface{

   Public:

    Status GetFeature(ClientContext* context, const Point& request, Feature* response) override;
    unique_ptr<ClientReaderWriter<RouteNote,RouteNote>> RouteChat(ClientContext* context) override;
  };

  static unique_ptr<Stub> NewStub(const shared_ptr<ChannelInterface>& channel,
                                   const StubOptions& options = StubOptions());

  class Service : public ::grpc::Service {

   Public:

    virtual Status GetFeature(ServerContext* context, const Point& request, Feature* response);
    virtual Status RouteChat(ServerContext* context, ServerReaderWriter<RouteNote, RouteNote>* stream);
  };
}
```

Write code for your service by creating a derived class that implements the RPC method handlers specified in the .proto file

Generated Code Snippet

```
class RouteGuide {

  class Stub : public StubInterface{

   Public:

    Status GetFeature(ClientContext* context, const Point& request, Feature* response) override;
    unique_ptr<ClientReaderWriter<RouteNote,RouteNote>> RouteChat(ClientContext& context) override;
  };

  static unique_ptr<Stub> NewStub(const shared_ptr<ChannelInterface>& channel,
                                  const StubOptions& options = StubOptions());

  class Service : public ::grpc::Service {

   Public:

    virtual Status GetFeature(ServerContext* context, const Point& request, Feature* response);
    virtual Status RouteChat(ServerContext* context, ServerReaderWriter<RouteNote, RouteNote>* stream);
  };
}
```

Write code for your client by creating a “Stub” and invoking RPCs as its member functions

Write code for your service by creating a derived class that implements the RPC method handlers specified in the .proto file

gRPC Advantages

Multi-language	On every platform	Strict Service contracts
Performant & Efficiency on wire	Extensible, Customizable	Easy to use
Streaming, BiDiStreaming APIs	Open & Standard compliant	Production Ready

gRPC Speaks Your Language

Multi-language	On every platform	Strict Service contracts
Performant & Efficiency on wire	Extensible, Customizable	Easy to use
Streaming, BiDiStreaming APIs	Open & Standard compliant	Production Ready

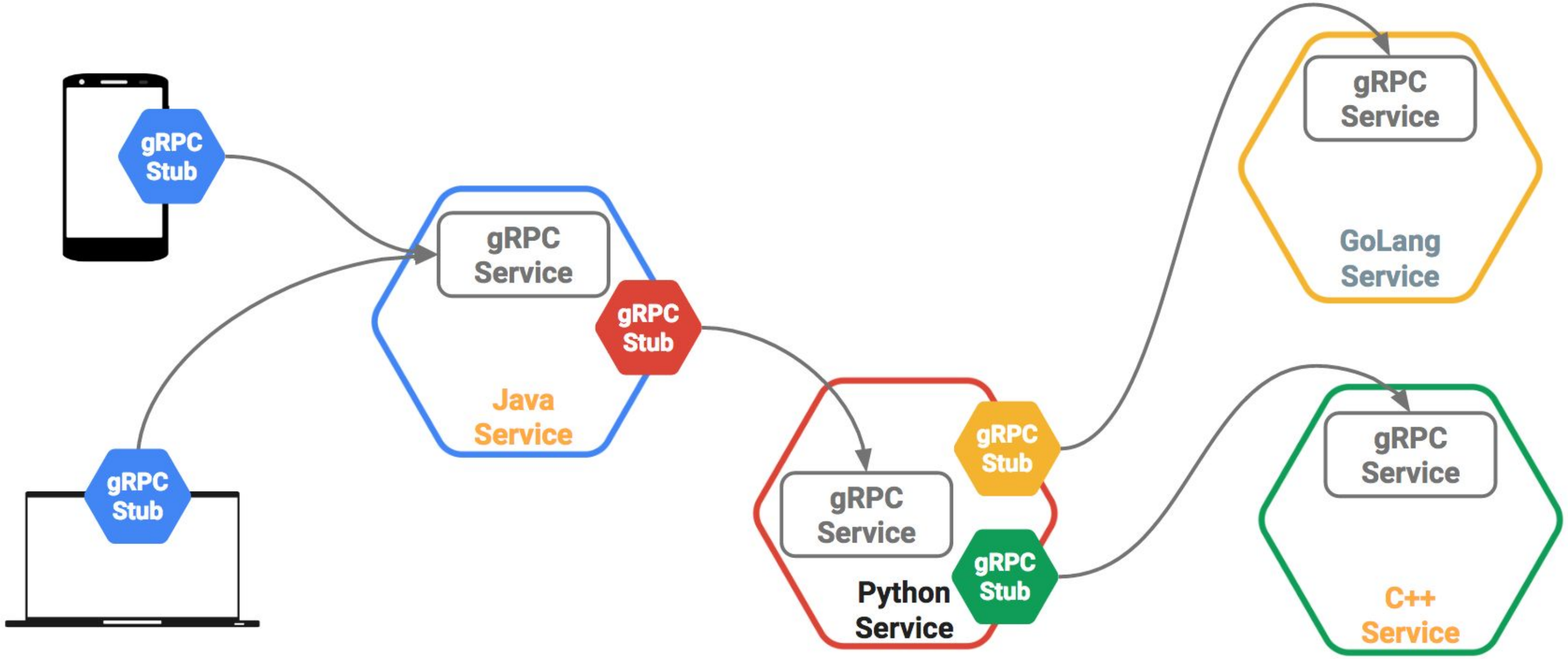
gRPC Speaks Your Language

Service definitions and client libraries

- Java
- Go
- C/C++
- C#
- Node.js
- PHP
- Ruby
- Python
- Objective-C
- Dart

More Languages...

- Swift
- Haskell
- Rust
- Typescript
- ....



Cross platform framework

Multi-language	**On every platform**	Strict Service contracts
Performant & Efficiency on wire	Extensible, Customizable	Easy to use
Streaming, BiDiStreaming APIs	Open & Standard compliant	Production Ready

Cross platform framework

More help from community on

- Bindings to languages such as clojure, kotlin, jruby
- New Environments and runtimes, e.g. GraalVM
- Supporting More platforms
- Testing on platforms that we do not have access to

Strongly Typed Service Contracts

Multi-language	On every platform	**Strict Service contracts**
Performant & Efficiency on wire	Extensible, Customizable	Easy to use
Streaming, BiDiStreaming APIs	Open & Standard compliant	Production Ready

Strongly Typed Protocol Buffers

- Strictly typed contract
- Conventions for Backward and forward compatibility of APIs
- Use your conventions for:
  - Semantic versioning
  - Stateless RESTful APIs
  - CRUD: enforce single service definition with Create, Read, Update, and Delete

```
syntax = “proto3”;

message Person {
  string name = 1;
  int32 id = 2;
  string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    string number = 1;
    PhoneType type = 2;
  }

  repeated PhoneNumber phone = 4;
}
```

Performant & Efficient

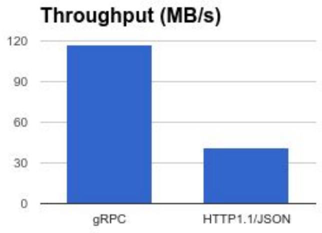
Multi-language	On every platform	Strict Service contracts
Performant & Efficiency on wire	Extensible, Customizable	Easy to use
Streaming, BiDiStreaming APIs	Open & Standard compliant	Production Ready

Performant & Efficient

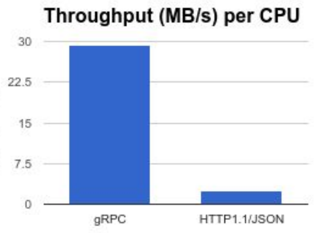
- HTTP/2 Performance:
  - Multiplexing, Header Compression, Binary Framing
- Binary compact protos: Serialization time, size of message on wire, client and server compute time, network throughput
- Libraries optimized for performance.

http://www.http2demo.io/

https://cloud.google.com/blog/big-data/2016/03/announcing-grpc-alpha-for-google-cloud-pubsub



3x increase in throughput



11x difference per CPU

Extensible, Customizable

Multi-language	On every platform	Strict Service contracts
Performant & Efficiency on wire	**Extensible, Customizable**	Easy to use
Streaming, BiDiStreaming APIs	Open & Standard compliant	Production Ready

Extensible, Customizable

- **Interceptors**
- **Transports**
- **Auth & Security**
    - Plugin auth mechanism for extensibility
- **Stats, Monitoring and Tracing**
    - Prometheus, Zipkin, OpenCensus, Opentracing integrations
- **Service Discovery**
    - Consul, Zookeeper, Eureka
- **Supported with Proxies**
    - Envoy, Nginx, linkerd, nghttp2, haproxy,...

Easy to use

Multi-language	On every platform	Strict Service contracts
Performant & Efficiency on wire	Extensible, Customizable	Easy to use
Streaming, BiDiStreaming APIs	Open & Standard compliant	Production Ready

Easy to use

- Single line installation
- Idiomatic APIs
- Error propagation
- Reconnect automatically on broken idle connections
- Cancellation propagation
- Deadline propagation

Stream is native to gRPC

Multi-language	On every platform	Strict Service contracts
Performant & Efficiency on wire	Extensible, Customizable	Easy to use
Streaming, BiDiStreaming APIs	Open & Standard compliant	Production Ready

Stream is native to gRPC

- **Unary RPC:**
  - Client sends a request
  - Server sends a response
- **Client Streaming RPC**:
  - Client sends multiple messages
  - Server sends one response
- **Server Streaming RPC**:
  - Client sends one message
  - Server sends multiple messages
- **Bidi Streaming RPC**:
  - Client and Server can independently send multiple messages to each other

Open & Standards Compliant

Multi-language	On every platform	Strict Service contracts
Performant & Efficiency on wire	Extensible, Customizable	Easy to use
Streaming, BiDiStreaming APIs	Open & Standard compliant	Production Ready

Open & Standards Compliant

- Developed on Github, in CNCF over an year
- Open RFC like process for Design changes
- HTTP2 based with gRPC wire protocol using HTTP2 published; standards based helps grpc traffic traverse network hops of proxies, firewalls

Production Ready

Multi-language	On every platform	Strict Service contracts
Performant & Efficiency on wire	Extensible, Customizable	Easy to use
Streaming, BiDiStreaming APIs	Open & Standard compliant	**Production Ready**

Production Ready

- Used in production by several large companies and projects!
- Well Tested:
  - Large number of tests for interoperability across languages
  - Large number of tests for portability across platforms
  - Fuzzing tests

gRPC Advantages in a nutshell

Multi-language *10+ languages*	**On every platform** *Linux, macosx, windows, Android, iOS, Embedded (iOT)*	**Strict Service contracts** *Define and enforce contracts, backward compatible*
Performant & Efficiency on wire *1m+ QPS - unary, 3m+ streaming (dashboard), 2-3X gains*	**Extensible, Customizable** *Interceptors, Auth, Transport, IDL, LB*	**Easy to use** *Single line installation, idiomatic APIs, Error propagation, cancellation propagation, deadline propagation*
Streaming, BiDiStreaming APIs *Large payloads, speech, logs*	**Open & Standard compliant** *Open source and growing community & HTTP/2*	**Production Ready** *Reliable, Well tested, Scalable*

Thank you