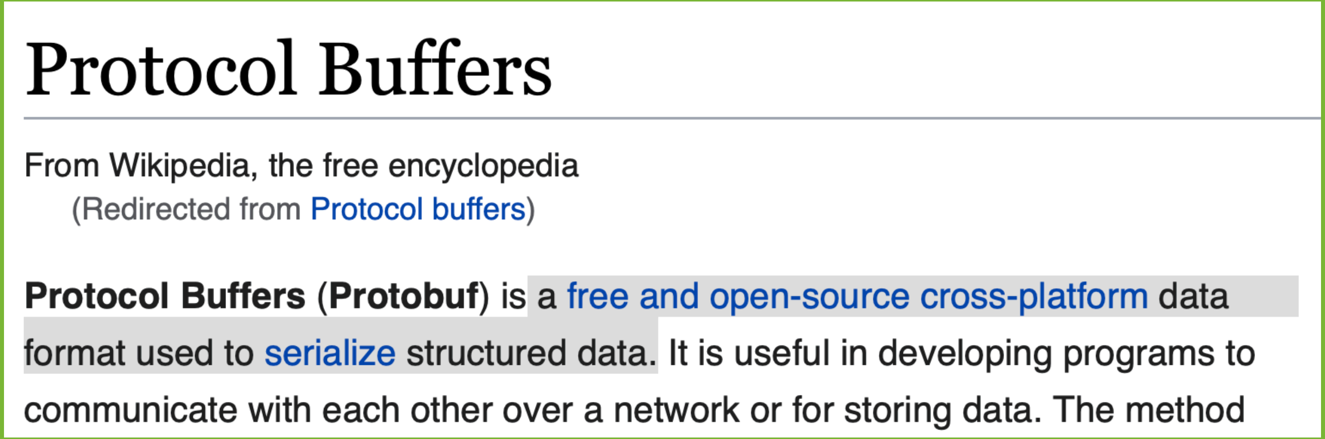
FH JOANNEUM University of Applied Sciences

Protocol Buffers



Protocol Buffers

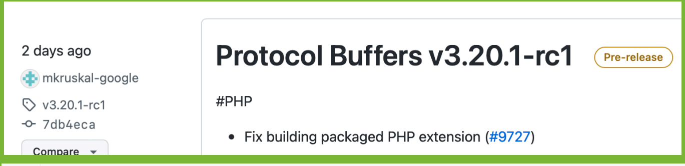
From Wikipedia, the free encyclopedia
(Redirected from Protocol buffers)

Protocol Buffers (**Protobuf**) is a free and open-source cross-platform data format used to serialize structured data. It is useful in developing programs to communicate with each other over a network or for storing data. The method

John Feiner

https://en.wikipedia.org/wiki/Protocol_Buffers

Protocol Buffers

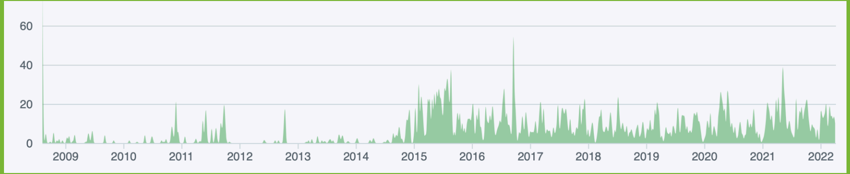


2 days ago
mkruskal-google
v3.20.1-rc1
7db4eca
Compare

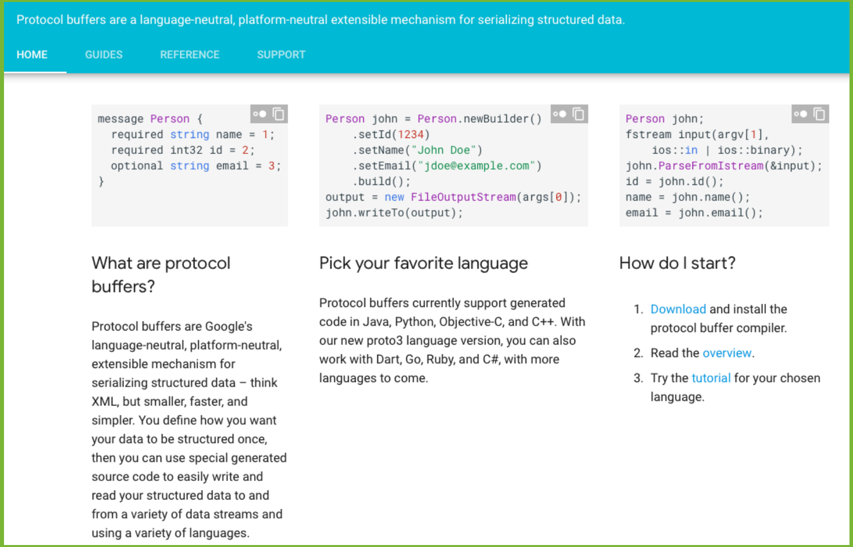
Protocol Buffers v3.20.1-rc1 Pre-release

#PHP

- Fix building packaged PHP extension (#9727)



https://github.com/protocolbuffers/protobuf/releases



Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data.

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
}
```

```
Person john = Person.newBuilder()
    .setId(1234)
    .setName("John Doe")
    .setEmail("jdoe@example.com")
    .build();
output = new FileOutputStream(args[0]);
john.writeTo(output);
```

```
Person john;
fstream input(argv[1],
    ios::in | ios::binary);
john.ParseFromIstream(&input);
id = john.id();
name = john.name();
email = john.email();
```

What are protocol buffers?

Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler. You define how you want your data to be structured once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages.

Pick your favorite language

Protocol buffers currently support generated code in Java, Python, Objective-C, and C++. With our new proto3 language version, you can also work with Dart, Go, Ruby, and C#, with more languages to come.

How do I start?

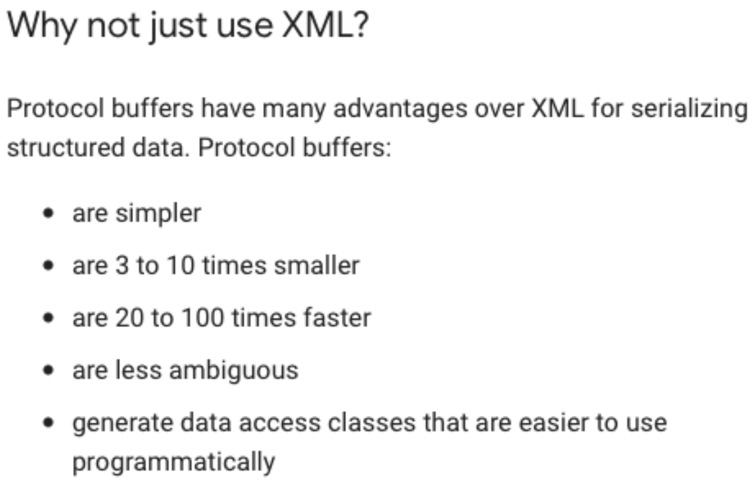
1. Download and install the protocol buffer compiler.
2. Read the overview.
3. Try the tutorial for your chosen language.

https://developers.google.com/protocol-buffers/

Why Protocol Buffers?

"Better" than text
(plain, cvs, html, XML, JSON),
because...

... fast, compact, **language and platform** agnostic serialisation transmission and preserve semantics.



Why not just use XML?

Protocol buffers have many advantages over XML for serializing structured data. Protocol buffers:

- are simpler
- are 3 to 10 times smaller
- are 20 to 100 times faster
- are less ambiguous
- generate data access classes that are easier to use programmatically

https://developers.google.com/protocol-buffers/docs/overview

Why Protocol Buffers?

Typical example:

Transfer
Server -> Client

Client
- generates log **objects (structured data)**
- **serialises into messages** (optionally include screen capture)
- sends log info "very often" to a server (allow to configure interval)

Server
- (dedicated "log" server, not the server hosting the large data)
- reads messages and **deserialises into objects**
- server writes to log file

How To – Step 1

Design Message Structure ("data class")

- **content:** (required/optional) **fields** (and defaults)
- **datatype: string**, enum, **int32,...**
- **structure:** e.g. **nested**

Example:

cracking md5 hashes

Define:

- Request
- Response
- Service

Request: **HashRequest**

With parameter: **md5hash** (The type should be a string)

Response: **PasswordReply**

With parameter: **password** (The type should be a string)

for a service: **MD5HashCracking**

method: **CrackTheMD5Hash**

How To – Step 2

Code Message (language agnostic):

...define protocol buffer message types in .proto files...

check allowed data types

Scalar (bool, uint64,...), Enums, Maps (key/value), ...

https://developers.google.com/protocol-buffers/docs/proto

Example:

cracking md5 hashes

cracking.proto

```
syntax = "proto3";
...
service MD5HashCracking {
   rpc CrackTheMD5Hash (HashRequest)
                              returns (PasswordReply) {}
}

message HashRequest {
   string md5hash = 1;
}

message PasswordReply {
   string password = 1;
}
```

How To – Step 3

Compile to get JavaScript, Ruby, Python, C#, ... stubs

For example,

```
    protoc ... --js_out ....
```

Example:

cracking md5 hashes

Compile
code for JavaScript
*(Using **protoc.js**)*

```
./node_modules/grpc-tools/bin/protoc.js \
  --js_out=import_style=commonjs,binary:./gen/ \
  --grpc_out=grpc_js:./gen cracking.proto
```

How To – Step 4

Use your message

For example in JS:

```
toObject()
serializeBinary()
deserializeBinary()
```

For example in Py

```
toObject()
SerializeToString()
ParseFromString()
```

Find a Tutorial at:
https://developers.google.com/protocol-buffers/docs/pythontutorial

Example:

cracking md5 hashes

Inspect ./gen/*.js:

- **cracking_grpc_pb.js**
- **cracking_pb.js**

```
...
proto.cracking.HashRequest.prototype.toObject = function(opt_includeInstance) {
        return proto.cracking.HashRequest.toObject(opt_includeInstance, this);
};
...
```

```
...
function deserialize_cracking_HashRequest(buffer_arg) {
  return cracking_pb.HashRequest.deserializeBinary(new Uint8Array(buffer_arg));
}
...
```

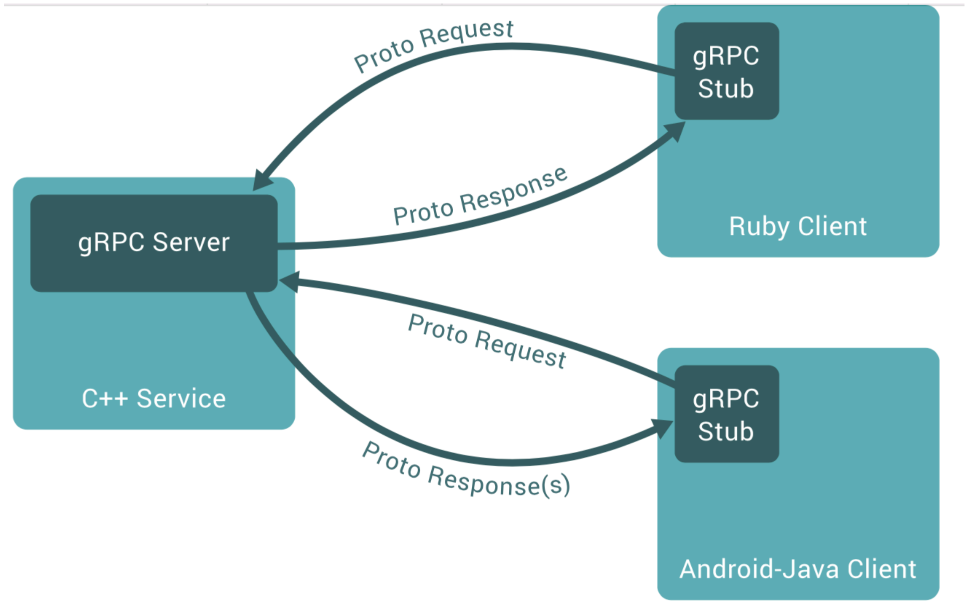
How To – Step 5

Transfer messages

....

Client/Server

....



(C++, Ruby, .. and many other options: Android, iOS, ...)

https://www.grpc.io/docs/ https://www.grpc.io/docs/what-is-grpc/introduction/

Example:
JS Client/Server

```
function crack(call, callback) {
        h = call.request.getMd5hash()
        repl = new messages.PasswordReply()
        reply.setPassword(`The cracked hash ${h} is for password ...')
        callback(null, reply)
}

var server = new grpc.Server();
server.addService(services.MD5HashCrackingService, {
      crackTheMD5Hash: crack
});

server.bindAsync(
        '0.0.0.0:50051',
        grpc.ServerCredentials.createInsecure(),
        () => {
        server.start();
    }
);
```

Example:
JS Client/Server

```
var serverWithPort = 'localhost:50051';

 var client = new services.MD5HashCrackingClient(
                serverWithPort,
                grpc.credentials.createInsecure());

 var request = new messages.HashRequest();
 request.setMd5hash('3acab568ca3c13728919f1c711e22afd');

 client.crackTheMD5Hash(request, function(err, response) {
     console.log('The password is:', response.getPassword());
 });
```

Example: JS Client/Server

```
./cracking_server.js
We start up the cracking server. CTRL-C to stop
...
```

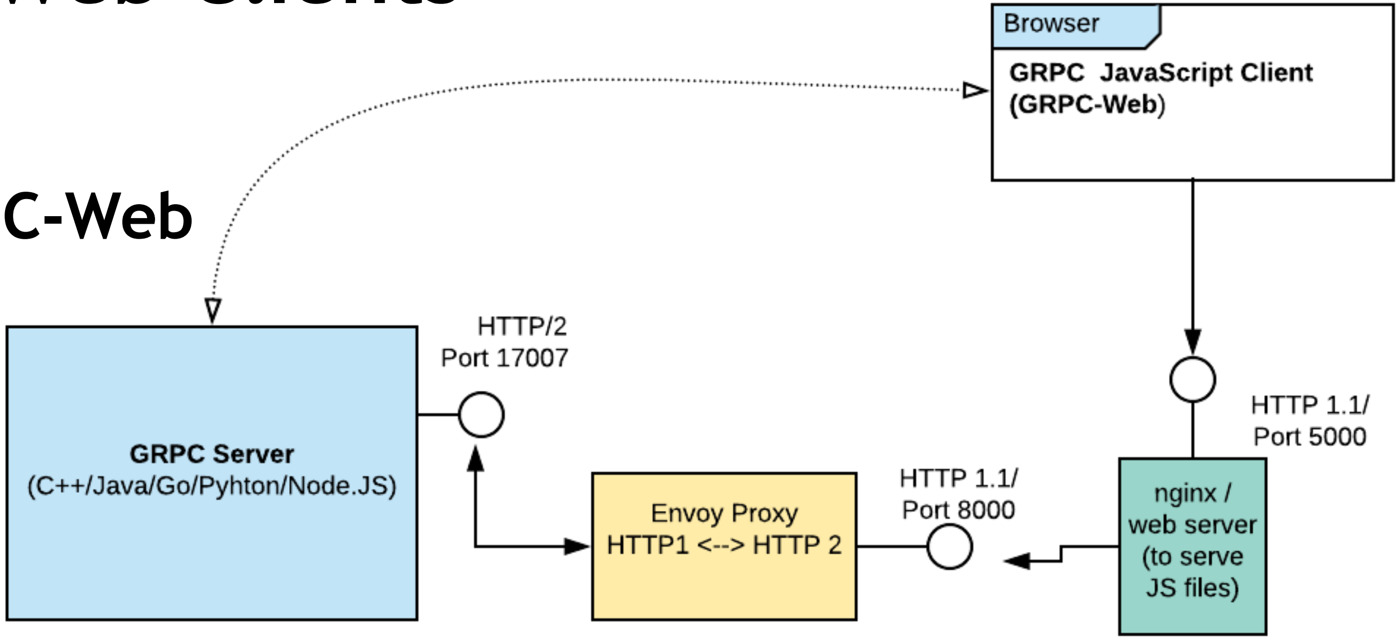
```
./cracking_client.js
We startup the MD5 Hash cracking gRPC client
...
```

```
We start up the cracking server. CTRL-C to stop
...
We try to crack the MD5 hash  3acab568ca3c13728919f1c711e22afd
```

```
./cracking_client.js
We startup the MD5 Hash cracking gRPC client
The password is: verySecure
```

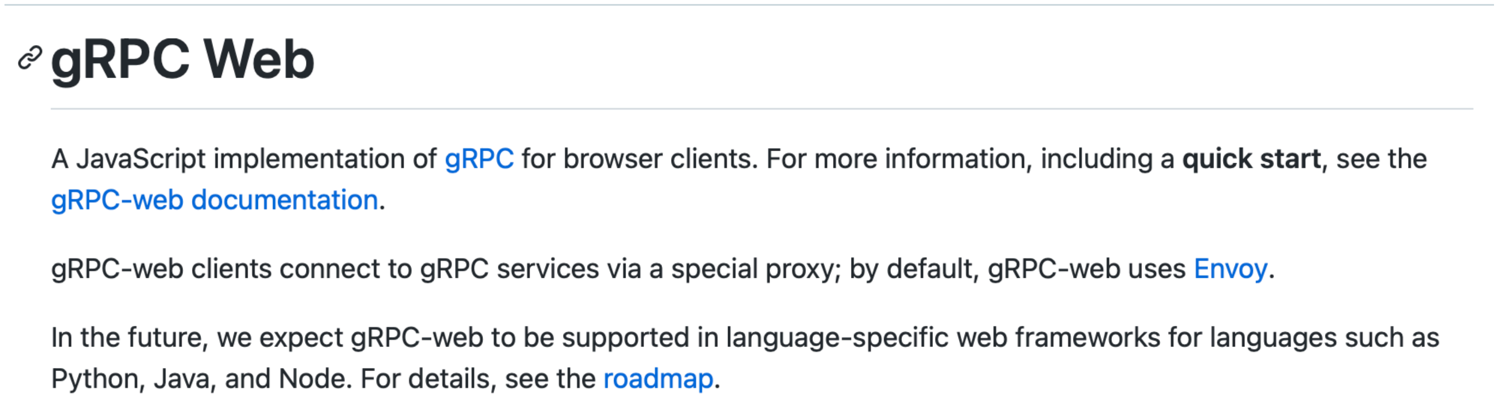
Special: Web Clients

In **Browser:**
using **gRPC-Web**


Browser
GRPC JavaScript Client (GRPC-Web)
HTTP 1.1/ Port 5000
nginx / web server (to serve JS files)
HTTP 1.1/ Port 8000
Envoy Proxy HTTP1 <--> HTTP 2
HTTP/2 Port 17007
GRPC Server (C++/Java/Go/Pyhton/Node.JS)

https://images.app.goo.gl/NF3DQXgifmtuREFL7

Special: Web Clients with gRPC Web



gRPC Web

A JavaScript implementation of gRPC for browser clients. For more information, including a **quick start**, see the gRPC-web documentation.

gRPC-web clients connect to gRPC services via a special proxy; by default, gRPC-web uses Envoy.

In the future, we expect gRPC-web to be supported in language-specific web frameworks for languages such as Python, Java, and Node. For details, see the roadmap.

https://github.com/grpc/grpc-web

https://www.grpc.io/docs/platforms/web/basics/