KubeCon | CloudNativeCon

Europe 2020

Virtual

gRPC Easy

Richard Belleville

Software Engineer, Google

[@gnossen](https://twitter.com/gnossen)

gRPC

About Me

- 2 years at Adtran
  - Developer/SM working on Cloud-Native Network Orchestration Systems
  - Maintained custom messaging framework
  - Wrote pre-Kubernetes container orchestrator
- ~1.5 years at Google, Sunnyvale
  - Developer on gRPC Team
  - Focus on Python and Usability

gRPC, A Recap

A modern open source high performance RPC framework

Multi-Language

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

Pluggable

+ auth, tracing, resolver, load balancing, IDL, health checking

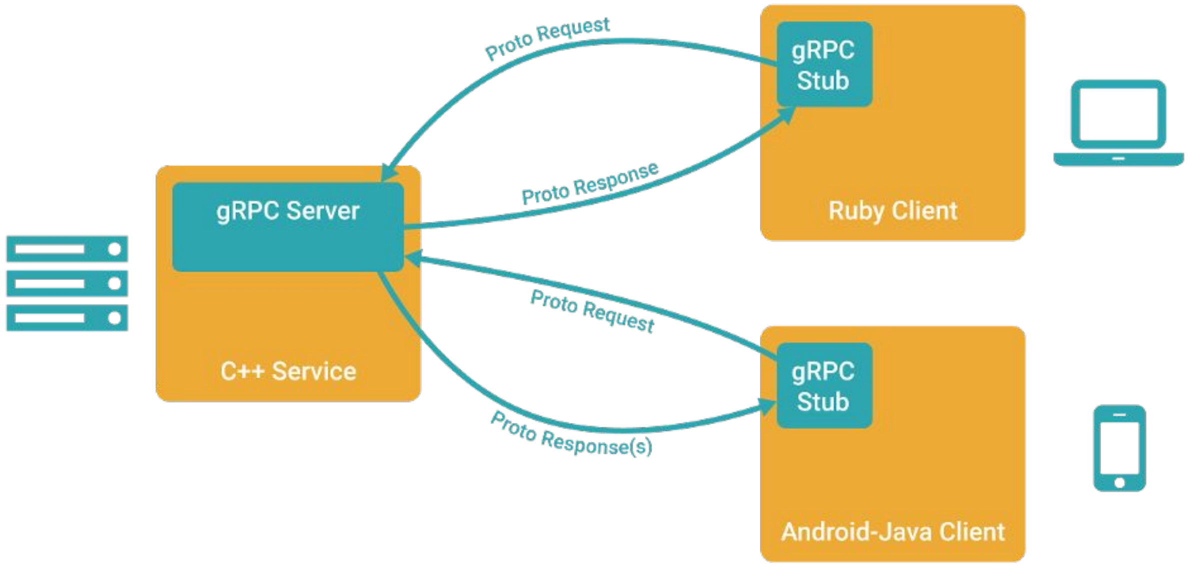
Multi-Platform

+ Linux, Windows, Mac OS X, iOS, Android

Feature-rich

+ bi-directional streaming, flow control
+ binary logging, channelz, tracing, retry, service config

gRPC, A Recap


gRPC Server
C++ Service
Proto Request
Proto Response
gRPC Stub
Ruby Client
Proto Request
Proto Response(s)
gRPC Stub
Android-Java Client

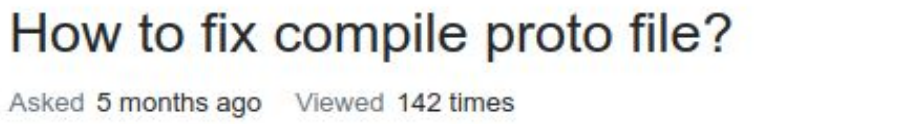
gRPC, A Recap

```
service KeyValueStore {
 rpc GetRecord(GetRecordRequest) returns (Record) {}
 rpc CreateRecord(CreateRecordRequest) returns (Record) {}
 rpc UpdateRecord(UpdateRecordRequest) returns (Record) {}
}
```

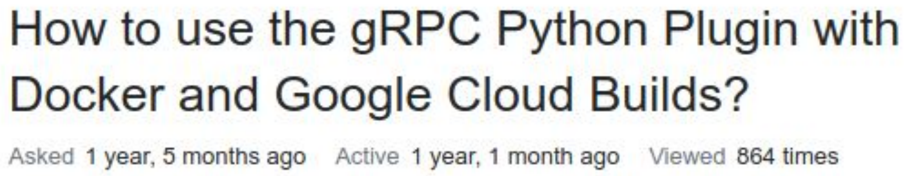
What is using gRPC like today?

- Well-Supported
- Performant
- Robust
- Safe
- Easy?

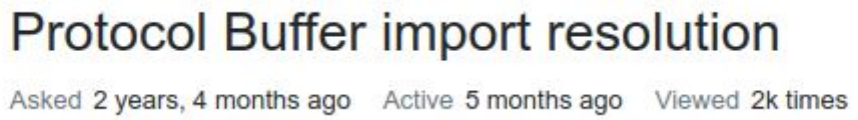
Easy?


How to fix compile proto file?

Asked 5 months ago   Viewed 142 times

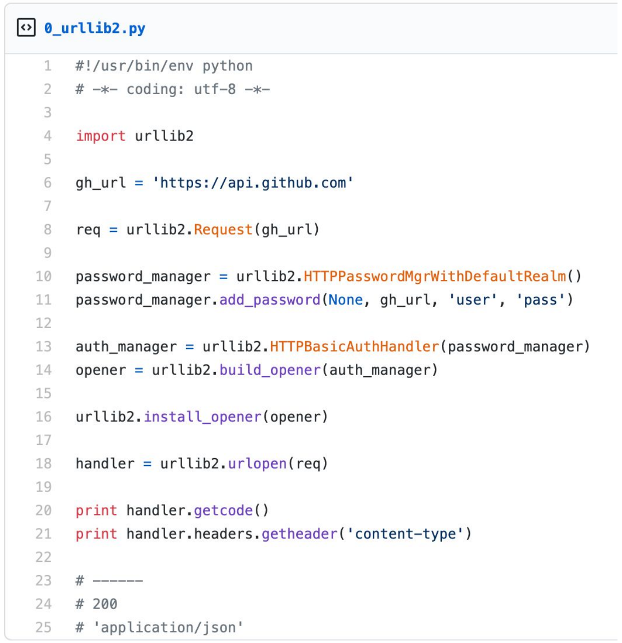

How to use the gRPC Python Plugin with Docker and Google Cloud Builds?

Asked 1 year, 5 months ago   Active 1 year, 1 month ago   Viewed 864 times


Protocol Buffer import resolution

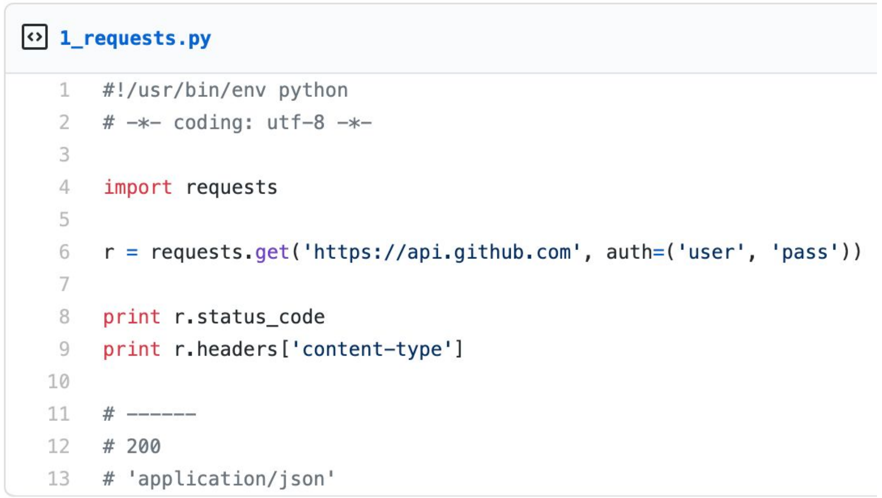
Asked 2 years, 4 months ago   Active 5 months ago   Viewed 2k times

The Inspiration: requests



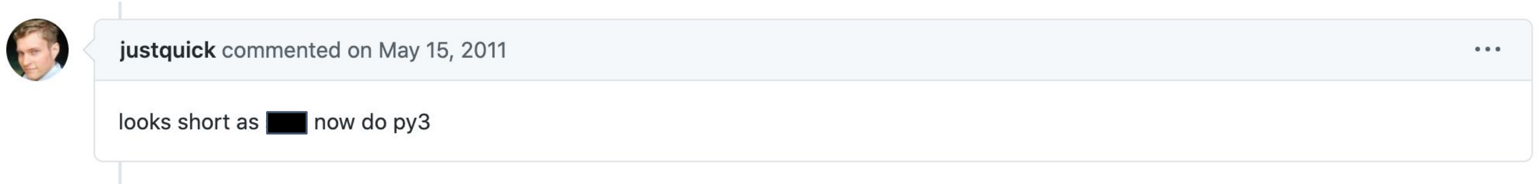
0_urllib2.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import urllib2

gh_url = 'https://api.github.com'

req = urllib2.Request(gh_url)

password_manager = urllib2.HTTPPasswordMgrWithDefaultRealm()
password_manager.add_password(None, gh_url, 'user', 'pass')

auth_manager = urllib2.HTTPBasicAuthHandler(password_manager)
opener = urllib2.build_opener(auth_manager)

urllib2.install_opener(opener)

handler = urllib2.urlopen(req)

print handler.getcode()
print handler.headers.getheader('content-type')

# ------
# 200
# 'application/json'
```



1_requests.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import requests

r = requests.get('https://api.github.com', auth=('user', 'pass'))

print r.status_code
print r.headers['content-type']

# ------
# 200
# 'application/json'
```



justquick commented on May 15, 2011

looks short as [redacted] now do py3

Approaches to Sharing Protobufs

- Monorepo
- Git submodule.
- Language-level package manager.
- Simple file server.
- ~~Checking your generated code into source control.~~ (Please don't)

Making gRPC... Easier

Before

```
import grpc
import helloworld_pb2
import helloworld_pb2_grpc


def main():
   with grpc.insecure_channel('localhost:50051') as channel:
       stub = helloworld_pb2_grpc.GreeterStub(channel)
       request = helloworld_pb2.HelloRequest(name='you')
       response = stub.SayHello(request)
   print("Greeter client received: " + response.message)
```

+

```
$ python -m grpc_tools.protoc \
      -I. \
      --python_out=. \
      --grpc_python_out=. \
      helloworld.proto
```

After

```
import grpc
protos = grpc.protos('helloworld.proto')
services = grpc.services('helloworld.proto')


def main():
   request = protos.HelloRequest(name='you')
   response = services.Greeter.SayHello(request,
                                        'localhost:50051')
```

Stubs and Messages

KubeCon CloudNativeCon Europe 2020 Virtual

Before

```python
import grpc
import helloworld_pb2
import helloworld_pb2_grpc

def run():
   with grpc.insecure_channel('localhost:50051') as channel:
       stub = helloworld_pb2_grpc.GreeterStub(channel)
       request = helloworld_pb2.HelloRequest(name='you')
       response = stub.SayHello(request)
   print("Greeter client received: " + response.message)
```

+

```
$ pip install grpcio-tools
$ python -m grpc_tools.protoc \
      -I. \
      --python_out=. \
      --grpc_python_out=. \
      helloworld.proto
```

After

```python
import grpc
protos = grpc.protos('helloworld.proto')
services = grpc.services('helloworld.proto')

def run():
   with grpc.insecure_channel('localhost:50051') as channel:
       stub = services.GreeterStub(channel)
       request = protos.HelloRequest(name='you')
       response = stub.SayHello(request)
   print("Greeter client received: " + response.message)
```

Channels

Before

```python
import grpc
import helloworld_pb2
import helloworld_pb2_grpc


def run():
   with grpc.insecure_channel('localhost:50051') as channel:
       stub = helloworld_pb2_grpc.GreeterStub(channel)
       request = helloworld_pb2.HelloRequest(name='you')
       response = stub.SayHello(request)
   print("Greeter client received: " + response.message)
```

After

```python
import grpc
import helloworld_pb2
import helloworld_pb2_grpc


def run():
   request = helloworld_pb2.HelloRequest(name='you')
   response = hello_world_pb2_grpc.Greeter.SayHello(request,
                                                    'localhost:50051',)
   print("Greeter client received: " + response.message)
```

Testing it out

```python
class KeyValueStore:
    def __init__(self):
        self._data = {}

    def store(self, key, value):
        self._data[key] = value

    def get(self, key):
        return self._data[key]

    def exists(self, key):
        return key in self._data
```

Testing it out

```
message Record {
 string name = 1;
 string value = 2;
}

message GetRecordRequest {
 string name = 1;
}

message CreateRecordRequest {
 Record record = 1;
}
```

```
message UpdateRecordRequest {
 Record record = 1;
}

service KeyValueStore {
 rpc GetRecord(GetRecordRequest) returns (Record) {}
 rpc CreateRecord(CreateRecordRequest) returns (Record) {}
 rpc UpdateRecord(UpdateRecordRequest) returns (Record) {}
}
```

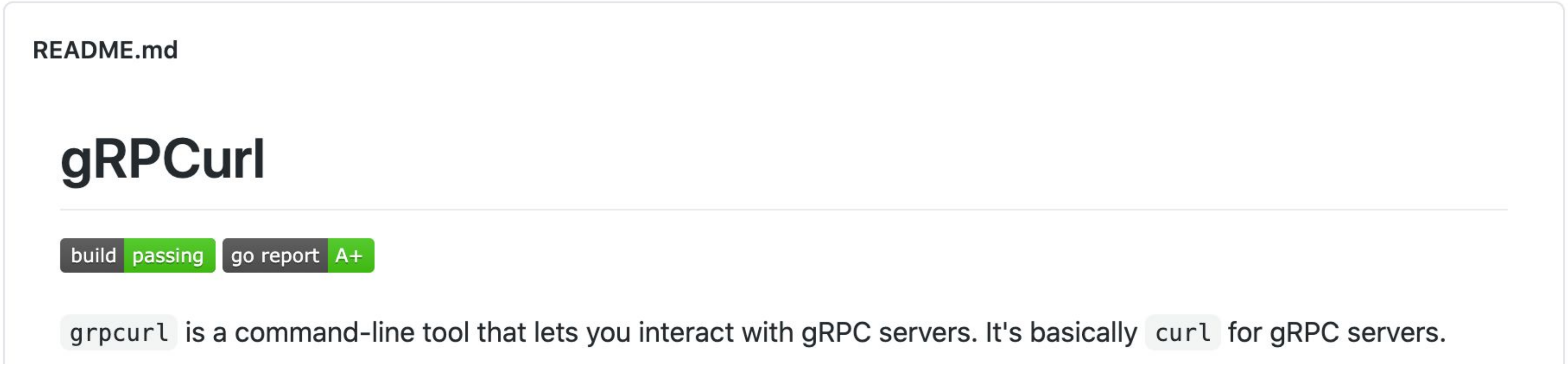
Sending an RPC

```
import grpc

protos = grpc.protos("key_value.proto")
services = grpc.services("key_value.proto")
KeyValueStore = services.KeyValueStore


def get(args):
   record = KeyValueStore.GetRecord(
            protos.GetRecordRequest(name=args.key),
            args.server,
            insecure=True)
   print(record.value)
```

```
def create(args):
   KeyValueStore.CreateRecord(
        protos.CreateRecordRequest(
             record=protos.Record(name=args.key,
                                  value=args.value)),
        args.server,
        insecure=True)
```

An Aside - grpcurl

- Great CLI tool by Joshua Humphries
- You don't necessarily need to write a client yourself
- Reflection is a big plus



README.md

gRPCurl

build passing | go report A+

`grpcurl` is a command-line tool that lets you interact with gRPC servers. It's basically `curl` for gRPC servers.

An Aside - grpcurl

```
$ grpcurl -plaintext \
     -d '{"record": {"name": "foo", "value": "1"}}' \
     localhost:50051 \
     key_value.KeyValueStore/CreateRecord

{
  "name": "foo",
  "value": "1"
}
```

KubeCon | CloudNativeCon

Europe 2020

Virtual

KEEP CLOUD NATIVE

CONNECTED