

ADRIAN ERNESTO AGUILAR QUEJ

The screenshot shows a Windows desktop environment. On the left, a web browser window displays the gRPC quickstart page for Python. The page includes instructions for installing gRPC using pip, with code snippets for both virtual environments and system-wide installation. Below the installation instructions, there is a section titled 'gRPC tools' which explains the role of the protocol buffer compiler and the special plugin for generating server and client code from .proto files.

On the right, a Visual Studio Code window is open, showing the 'Welcome' page. The 'Terminal' panel at the bottom of the window displays the output of the command `python -m pip install grpcio`. The terminal output shows the collection of gRPC packages, the download of the necessary wheels, and the successful installation of gRPC version 1.62.0.

gRPC Quickstart Instructions:

```
$ source venv/bin/activate
$ python -m pip install --upgrade pip
```

gRPC Installation:

```
$ python -m pip install grpcio
```

gRPC Tools:

Python's gRPC tools include the protocol buffer compiler `protoc` and the special plugin for generating server and client code from `.proto` service definitions. For the first part of our quick-start example, we've already generated the server and client stubs from [helloworld.proto](#), but you'll need the tools for the rest of our quick start, as well as later tutorials and your own projects.

Terminal Output:

```
PS C:\Users\Adrian Quej> python -m pip install grpcio
Collecting grpcio
  Downloading grpcio-1.62.0-cp312-cp312-win_amd64.whl.metadata (4.2 kB)
  Downloading grpcio-1.62.0-cp312-cp312-win_amd64.whl (3.8 MB)
    3.8/3.8 MB 2.2 MB/s eta 0:00:00
Installing collected packages: grpcio
Successfully installed grpcio-1.62.0
PS C:\Users\Adrian Quej>
```

Quick start | Python | gRPC

grpcio/docs/languages/python/quickstart/

grpcio tools

Python's gRPC tools include the protocol buffer compiler `protoc` and the special plugin for generating server and client code from `.proto` service definitions. For the first part of our quick-start example, we've already generated the server and client stubs from [helloworld.proto](#), but you'll need the tools for the rest of our quick start, as well as later tutorials and your own projects.

To install gRPC tools, run:

```
$ python -m pip install grpcio-tools
```

Download the example

You'll need a local copy of the example code to work through this quick start. Download the example code from our GitHub repository (the following command clones the entire repository, but you just need the examples for this quick start and other tutorials):

```
$ git clone -b v1.62.0 --depth 1 --shallow-submodules https://github.com/grpc/grpc
```

Run a gRPC application

From the `examples/python/helloworld` directory:

1. Run the server:

```
$ python greeter_server.py
```

Visual Studio Code interface showing the terminal output:

```
PS C:\Users\Adrian Quej> python -m pip install grpcio
Collecting grpcio
  Downloading grpcio-1.62.0-cp312-cp312-win_amd64.whl.metadata (4.2 kB)
  Downloading grpcio-1.62.0-cp312-cp312-win_amd64.whl (3.8 MB)
    3.8/3.8 MB 2.2 MB/s eta 0:00:00
Installing collected packages: grpcio
Successfully installed grpcio-1.62.0
PS C:\Users\Adrian Quej> python -m pip install grpcio-tools
Collecting grpcio-tools
  Downloading grpcio-tools-1.62.0-cp312-cp312-win_amd64.whl.metadata (6.4 kB)
Collecting protobuf<5.0dev,>=4.21.6 (from grpcio-tools)
  Downloading protobuf-4.25.3-cp310-abi3-win_amd64.whl.metadata (541 bytes)
Requirement already satisfied: grpcio>=1.62.0 in c:\python312\lib\site-packages (from grpcio-tools) (1.62.0)
Collecting setuptools (from grpcio-tools)
  Downloading setuptools-69.1.1-py3-none-any.whl.metadata (6.2 kB)
  Downloading setuptools-1.62.0-cp312-cp312-win_amd64.whl (1.1 MB)
    1.1/1.1 MB 944.6 kB/s eta 0:00:00
  Downloading protobuf-4.25.3-cp310-abi3-win_amd64.whl (413 kB)
    413.4/413.4 kB 124.6 kB/s eta 0:00:00
  Downloading setuptools-69.1.1-py3-none-any.whl (819 kB)
    819.3/819.3 kB 1.1 MB/s eta 0:00:00
Installing collected packages: setuptools, protobuf, grpcio-tools
Successfully installed grpcio-tools-1.62.0 protobuf-4.25.3 setuptools-69.1.1
PS C:\Users\Adrian Quej>
```

Visual Studio Code interface showing the terminal output:

```
PS C:\Users\Adrian Quej\Downloads\grpc> git clone -b v1.62.0 --depth 1 --shallow-submodules https://github.com/grpc/grpc
Cloning into 'grpc'...
remote: Enumerating objects: 13417, done.
remote: Counting objects: 100% (13417/13417), done.
remote: Compressing objects: 100% (8020/8020), done.
remote: Total 13417 (delta 4535), reused 11148 (delta 3976), pack-reused 0
Receiving objects: 100% (13417/13417), 19.66 MiB | 1.41 MiB/s, done.
Resolving deltas: 100% (4535/4535), done.
Note: switching to 'f78a54c5ad4e058734aa9b2beb9459940e4de342'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

Updating files: 100% (12258/12258), done.
PS C:\Users\Adrian Quej\Downloads\grpc>
```

The image is a composite of two screenshots showing a gRPC application setup and execution. The top screenshot shows the initial state where the server is running and the client is being executed. The bottom screenshot shows the client output and the server being updated.

Top Screenshot:

Browser: The browser shows the gRPC quickstart page for Python. The title is "Run a gRPC application". The content says: "From the `examples/python/helloworld` directory: 1. Run the server: `$ python greeter_server.py` 2. From another terminal, run the client: `$ python greeter_client.py` Congratulations! You've just run a client-server application with gRPC."

VS Code: The VS Code interface shows the "Terminal" tab. The output shows the server starting and listening on 50051. The client is also running, and the output shows "Greeter client received: Hello, you!".

Bottom Screenshot:

Browser: The browser shows the same gRPC quickstart page. The content says: "From the `examples/python/helloworld` directory: 1. Run the server: `$ python greeter_server.py` 2. From another terminal, run the client: `$ python greeter_client.py` Congratulations! You've just run a client-server application with gRPC."

VS Code: The VS Code interface shows the "Terminal" tab. The output shows the server starting and listening on 50051. The client is also running, and the output shows "Greeter client received: Hello, you!".

Let's update this so that the `Greeter` service has two methods. Edit `examples/protos/helloworld.proto` and update it with a new `SayHelloAgain` method, with the same request and response types:

```
// 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;
}
```

Remember to save the file!

Generate gRPC code

Next we need to update the gRPC code used by our application to use the new service definition.

From the `examples/python/helloworld` directory, run:

```
$ python -m grpc_tools.protoc -I../protos --python_out=.
```

This regenerates `helloworld_pb2.py` which contains our generated request and response classes and `helloworld_pb2_grpc.py` which contains our generated client and server classes.

Update and run the application

We now have new generated server and client code, but we still need to implement and call the new method in the human-written parts of our example application.

```
helloworld.proto
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;
}
```

```
helloworld_pb2.py
# -*- coding: utf-8 -*-
# Generated by the protocol buffer compiler. DO NOT EDIT!
# source: helloworld.proto
# Protobuf Python Version: 4.25.1
"""Generated protocol buffer code."""
from google.protobuf import descriptor as _descriptor
from google.protobuf import descriptor_pool as _descriptor_pool
from google.protobuf import symbol_database as _symbol_database
from google.protobuf.internal import builder as _builder
import sys
import warnings

_sym_db = _symbol_database.Default()

DESCRIPTOR = _descriptor_pool.Default().AddSerializedFile(b'\n\thelloworld.proto\n\nservice Greeter {\n  rpc SayHello(HelloRequest) returns (HelloReply);\n  rpc SayHelloAgain(HelloRequest) returns (HelloReply);\n}\n\nmessage HelloRequest {\n  string name = 1;\n}\n\nmessage HelloReply {\n  string message = 1;\n}\n\n')
_HelloRequest = _builder._MessageClass('HelloRequest', _DESCRIPTOR, {})
```

```
helloworld_pb2_grpc.py
import grpc
import helloworld_pb2

class GreeterStub(object):
    """The Greeter service definition."""

    def SayHello(self, request, timeout=None):
        """Sends a greeting"""
        return self.Stub.SayHello(request, timeout)

    def SayHelloAgain(self, request, timeout=None):
        """Sends another greeting"""
        return self.Stub.SayHelloAgain(request, timeout)

class GreeterServicer(object):
    """The Greeter service definition."""

    def SayHello(self, request, context):
        """Sends a greeting"""
        return helloworld_pb2.HelloReply()

    def SayHelloAgain(self, request, context):
        """Sends another greeting"""
        return helloworld_pb2.HelloReply()

def add_GreeterServicer_to_server(servicer, server):
    rpc_method_handlers = {
        'SayHello': grpc.method_handlers.GenericMethodHandler(servicer.SayHello),
        'SayHelloAgain': grpc.method_handlers.GenericMethodHandler(servicer.SayHelloAgain),
    }
    server.add_rpc_handlers(rpc_method_handlers)
```

```
terminal
PS C:\Users\Adrian Quej\Downloads\grpc> python greeter_server.py
Server started, listening on 50051

PS C:\Users\Adrian Quej\Downloads\grpc> python greeter_client.py
Greeter client received: Hello, you!
```

The image is a composite of two screenshots showing a development environment. The left side displays a web browser with the gRPC documentation for Python, and the right side shows a VS Code editor with the corresponding Python code for a helloworld pb2 server and client.

Top Screenshot:

- Browser (Left):** Shows the gRPC documentation page for Python. The title is "Update and run the application". The text says: "We now have new generated server and client code, but we still need to implement and call the new method in the human-written parts of our example application." Below this, there are two sections: "Update the server" and "Update the client".
- VS Code (Right):** Shows the file `helloworld_pb2.py`. The code is generated by the protocol buffer compiler. It includes imports for `google.protobuf` and `grpc`. The code defines a `Greeter` class that implements the `GreeterService` interface. The `say_hello` method is implemented to return a `HelloReply` message with the message "Hello, %s!".

Bottom Screenshot:

- Browser (Left):** Shows the gRPC documentation page for Python. The title is "Update the server". The text says: "In the same directory, open `greeter_server.py`. Implement the new method like this:". Below this, there is a code block showing the implementation of the `Greeter` class.
- VS Code (Right):** Shows the file `greeter_server.py`. The code is a Python implementation of the `Greeter` class. It includes imports for `concurrent.futures`, `logging`, `grpc`, `helloworld_pb2`, and `helloworld_pb2_grpc`. The `say_hello` method is implemented to return a `HelloReply` message with the message "Hello, %s!".

The image is a composite of two screenshots. The top screenshot shows a web browser displaying the gRPC documentation page for updating the client. The bottom screenshot shows a VS Code editor with the same Python code as the top screenshot, but with a terminal window open showing the execution of the server and client programs.

Update the client

In the same directory, open `greeter_client.py`. Call the new method like this:

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

Run!

Just like we did before, from the `examples/python/helloworld` directory:

1. Run the server:

```
$ python greeter_server.py
```

Run!

Just like we did before, from the `examples/python/helloworld` directory:

1. Run the server:
2. From another terminal, run the client:

```
$ python greeter_client.py
```

What's next

- Learn how gRPC works in [Introduction to gRPC and Core concepts](#).

VS Code Editor:

The VS Code editor shows the `greeter_client.py` file with the following code:

```
import logging
import grpc
import helloworld_pb2
import helloworld_pb2_grpc

def run():
    print("Will try to greet world ...")
    with grpc.insecure_channel('localhost:50051') as channel:
        stub = helloworld_pb2_grpc.GreeterStub(channel)
        response = stub.SayHello(helloworld_pb2.HelloRequest(name='you'))
        print("Greeter client received: " + response.message)
        response = stub.SayHelloAgain(helloworld_pb2.HelloRequest(name='you'))
        print("Greeter client received: " + response.message)

if __name__ == "__main__":
    logging.basicConfig()
    run()
```

The terminal window shows the following output:

```
PS C:\Users\Adrian Quej\Downloads\grpc> cd g\
grpc\examples\python\helloworld
PS C:\Users\Adrian Quej\Downloads\grpc\g\
examples\python\helloworld> python greeter_
server.py
Server started, listening on 50051
PS C:\Users\Adrian Quej\Downloads\grpc\g\
examples\python\helloworld> python greeter_
client.py
Will try to greet world ...
Greeter client received: Hello, you!
Greeter client received: Hello, you!
```