1. Install Dependencies

- Install **gRPC** and **Protocol Buffers** for your chosen programming language. For example, in Python, you can install the necessary packages like this:

```
pip install grpcio grpcio-tools
```

For other languages like Go, Java, or Node.js, the process is similar but with their respective package managers.

2. Define the Service with Protobuf

- Create a `.proto` file to define the service and message types. This will act as the contract between the client and server.

Example: `service.proto`

```
syntax = "proto3";

service MyService {
   rpc GetData (Request) returns (Response);
}

message Request {
   string query = 1;
}

message Response {
   string data = 1;
}
```

This file defines a service `MyService` with a single method `GetData` that takes a `Request` and returns a `Response`.

3. Generate Code from `.proto` File

Use the `grpcio-tools` (or equivalent for your language) to generate server and client code from the `.proto` file.

For Python:

```
python -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. service.proto
```

This will generate the `service_pb2.py` and `service_pb2_grpc.py` files, which contain the necessary code to implement both the server and the client.

4. Implement the Server

Create a server that implements the service you defined in the `.proto` file.

Example: `server.py`

```
import grpc
from concurrent import futures
import time
import service_pb2
import service_pb2_grpc

class MyService(service_pb2_grpc.MyServiceServicer):
    def GetData(self, request, context):
        return service_pb2.Response(data=f"Received: {request.query}")

def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    service_pb2_grpc.add_MyServiceServicer_to_server(MyService(), server)
    server.add_insecure_port('[::]:50051')
    print("Server is listening on port 50051...")
    server.start()
    try:
        while True:
            time.sleep(86400)
    except KeyboardInterrupt:
        server.stop(0)

if __name__ == '__main__':
    serve()
```

5. Implement the Client

Create a client that can communicate with the server using the gRPC service.

Example: `client.py`

```
import grpc
import service_pb2
import service_pb2_grpc

def run():
    channel = grpc.insecure_channel('localhost:50051')
    stub = service_pb2_grpc.MyServiceStub(channel)
    response = stub.GetData(service_pb2.Request(query="Hello, gRPC!"))
    print(f"Server response: {response.data}")

if __name__ == '__main__':
    run()
```

6. Run the Server and Client

- First, start the server:

```
python server.py
```

- Then, in another terminal window, run the client:

```
python client.py
```

The client should output the server's response: `Received: Hello, gRPC!`.

7. Next Steps

- **Security**: Consider implementing secure communication via TLS for gRPC services, especially for production environments.
- **Load Balancing**: If you plan to scale your services, look into gRPC load balancing solutions.
- **Error Handling**: gRPC has robust support for handling errors and status codes. Check the gRPC status codes to handle different scenarios.
- **Streaming**: If you need continuous real-time updates, explore gRPC's support for bidirectional streaming.

gRPC is a powerful framework that fits very well with microservices and can be scaled horizontally to handle a large number of requests efficiently.

Let me know if you need more help!