Compared to JSON, Protocol Buffers come with a schema and the client libraries that can warn you of typos as well as give you type mismatch errors in your data. Both protocol buffers and thrift are primarily designed to handle older or newer messages (than your program was compiled with) gracefully, but they do this with some penalty to speed and compactness.

The basic wire format is \([(tag, payload \text{ bytes})^*]\). The tag identifies both the field "number", and the wire-type of the field. Wire types include fixed 32 and 64, length-delimited ranges, and, for protos, variable length integers. The parser knows how to skip fields of all wire types, so any unknown tags are skipped (and retained in memory) during parsing.

Interana strongly prefers JSON as a native format, so we provide the following reference for how to perform the conversion coming from ProtoBuf.

**Step #0: Install the ProtoBuf Machinery**

Mostly this means the packages protobuf and python-protobuf are installed on your linux machine. Type `protoc` on the command line and if it works your install was successful.

**Step #1: Get the .proto files**

The first step is to get the schema. Unlike Avro files, .proto files are not self-describing so one must provide the definition files. For example, a web service might store user data in a protobuf schema that looks as follows:

```plaintext
message UserProto {
  optional ProfileProto profile = 1;
  optional ModelFeaturesProto model_features = 2;
  optional ActivityProto user_activity = 3;
  optional CalculatedProto calculated_values = 4;
  optional MatchingProto matching_activity = 5;
  optional TimelineProto user_timeline = 6;
  // Tracks the time when last FULL refresh of the
  // current user's data has been performed.
  // The date is stored in ISO8601 format.
  optional string last_refreshed_time = 7;
}
```

https://docs.interana.com/HowTos/Converting_Protocol_Buffers_(ProtoBuf)_to_JSON_for_ingestion
Updated: Wed, 16 Oct 2019 03:48:48 GMT
Powered by
**Step #2: Generate Python Code from .proto**

Protobuffs comes with target code generators for most languages. Since we work in Python, you'll want to compile the protobuf into our language as follows:

```
protoc --proto_path=protos User.proto --python_out=./Protos/build
```

As you can see, we give the path where we store the definitions and a output directory for python. In general, we recommend keeping all proto files to convert in a single directory called protos and run something like the following:

```
for i in `ls ./protos/*`; do protoc --proto_path=protos $i --python_out=.; done
```

Now the above simple generates python client code based on the proto definitions with no effort on your part. Do NOT manually edit any generated code files.

**Step 3: Run the Transformer**

Now that we have python code that tells us how to represent these objects we defined, we are ready to ingest data from the pickled/serialized format into JSON or any other format of our choice. The key trick in any transformer ought to be to first extract out the encoded string and run through the following lines which read into a python object from the raw form.

```python
from google.protobuf import text_format
from google.protobuf.message import DecodeError

def Parse2Protobuf(file_contents_string, protobuf, ascii_text=None):
    """Parse a string containing a protobuf.
    set ascii_text=True if ASCII representation is used
    Returns:
    True if the protobuf was read correctly.
    ""

    if file_contents_string is None:
        print('Unable to read string %s while trying to parse it as protobuf ' '
             'message %s.', file_contents_string, type(protobuf))
        return False
    try:
        if ascii_text:
            text_format.Merge(file_contents_string, protobuf)
        else:
            protobuf.MergeFromString(file_contents_string)
        return True
    except (DecodeError, text_format.ParseError) as err:
        print('Error while parsing file %s to protobuf message %s: %s', 
             file_contents_string, type(protobuf), err)
        return False
```

https://docs.interana.com/HowTos/Converting_Protocol_Buffers_(ProtoBuf)_to_JSON_for_ingestion

Updated: Wed, 16 Oct 2019 03:48:48 GMT
Powered by
That's it. After the above step you can convert the object to a dict and then JSON for ingestion into Interana.