proto3 spec

protobuf - proto3 spec:

```
syntax = "proto3";

message SearchRequest {
  string query = 1;
  int32 page_number = 2;
  int32 result_per_page = 3;
}
```

Note - numbered fields, required vs. optional. All scalar types, composite ones come later.

Tags - 1 byte for numbers 1 - 15, and 2 bytes for tags 16 - 2047.

- So if you use a low tag number for an optional field, you've done something wrong.
- Also - leave room for additional frequently occuring fields in the future in your low tag numbers!
- 5k messages / second for 50 jobs = 21.6gb additional space per day per poorly labelled tag!

Singular and **required** for field frequency - singular is {0,1}, and repeated is an ordered {0,N}

- repeated numerics are "packed" by default, which is an encoding thing
- proto3 got rid of required! I'd say follow that wind.

Multiple message types can be defined in one .proto file (message, enum, service) for grouping purposes, but this can lead to dependency bloat.

You can have a **reserved** keyword for certain tag numbers if you don't want them to be re-used in the future!

```
message Foo {
  reserved 2, 15, 9 to 11;
  reserved "foo", "bar";
}
```

Data type notes:

- use "sint32" instead of "int32" for fields likely to have negative numbers
- "sfixed32" - signed fixed (always four bytes), good for fields that typically are larger than 2^28
- string - UTF8

Optional field defaults by type (when some other type-specific default is not provided) - worth knowing:

- string = ""
- bool = false

- numeric = 0
- enums = first value listed in enum definition, must be zero (SO BE CAREFUL WHEN MIGRATING AN ENUM'S DEFINITION!!)
- bytes = empty bytes
- repeated fields - empty list (in the appropriate language)

Things chosen to default don't serialize on the wire. Also, there's no way to differentiate a default from a hard-coded match to the default.

```
optional int32 result_per_page = 3 [default = 10];
```

Enums - numbering fields within an enum for more safety! Can specify enum aliases for overloading, just need an option to allow it.

- There MUST be a zero value in your enum for defaults.

```
message SearchRequest {
  required string query = 1;
  optional int32 page_number = 2;
  optional int32 result_per_page = 3 [default = 10];
  enum Corpus {
    UNIVERSAL = 0;
    WEB = 1;
    IMAGES = 2;
    LOCAL = 3;
    NEWS = 4;
    PRODUCTS = 5;
    VIDEO = 6;
  }
  optional Corpus corpus = 4 [default = UNIVERSAL];
}
```

You can import proto definitions from other files, and create more or less symbolic links from old files to new files in a different location. Another thing to ease migrations!

You can also nest message types and reference those accordingly.

```
message SearchResponse {
  message Result {
    required string url = 1;
    optional string title = 2;
    repeated string snippets = 3;
  }
  repeated Result result = 1;
}
```

Updating your message type:

- Added fields are read by your new code using defaults when not present in a message; old code will just ignore new fields.
- Remove fields by renaming with "OBSOLETE_" or adding the "reserved" keyword.

- int32, uint32, int64, uint64, and bool are forward/backward compatible
- sint32 and sint64 are compatible with each other AND NOTHING ELSE
- string and bytes are compatible if the bytes are UTF8
- Some other details, not crazy important

Unknown fields are NOT available upon deserialization, even if the deserialization is successful.

Any type - ability to fetch and unpack / pack any protobuf message at runtime, to then be used and parsed more specifically, I guess. Sorta like generics? Example (is this...C++?):

```
ErrorStatus status = ...;
for (const Any& detail : status.details()) {
  if (detail.Is<NetworkErrorDetails>()) {
    NetworkErrorDetails network_error;
    detail.UnpackTo(&network_error);
    ... processing network_error ...
  }
}
```

Oneof - only one field of a list of possible fields will be set at any given time (shares underlying memory). Not sure when you'd use this, isn't it like an enum?

Maps - map<string, Project> projects = 3;

- Can't be repeated
- Not ordered on the wire

Options - only notable ones I've seen:

- optimize_for (file option): Can be set to SPEED, CODE_SIZE, or LITE_RUNTIME. We should do speed.

Proto3 - supports JSON mappings! Just class/method calls within a particular language's library.