Title: Protobuf message format for Bekcn Protocol **Draft**: 0.1

Proposal Date: 09-01-2022 **Authors:** Nikhil, Dheeraj Kumar, Pradip @Finarkein Analytics

Expires on: 09-06-2022 **State**: Proposal

Protobuf(Protocol Buffers) message format for Beckn Protocol

Proposal

Abstract

While JSON and XML are widely popular data transfer formats when it comes to REST APIs, they're not the only options available. There exist many other formats with varying degrees of serialization speed and serialized data size. Beckn protocol is intended to be form agnostic and transport layer agnostic. This proposal explains the motivation and working of protobuf message format in Beckn protocol. Its pros, cons and limitations.

Table of Contents

Title: Protobuf message format for Bekcn Protocol

Introduction

Protocol Buffers are language independent, platform independent mechanisms for serializing, deserializing structured data, developed by Google. Intent behind it was to have an optimized mechanism of sending data over the wire, keeping an eye on enhancing network performance and usage.

As per documentation[1] '*Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data — think XML, but smaller, faster, and simple'*. It also acts as a basis for a custom procedure call (gRPC) system, which is widely adapted for inter-machine communication at many organizations today. There are equivalent cross language serialization platforms, frameworks are available e.g. Apache Thrift, Microsoft Bond, etc.

1. Motivation

There are many online resources that prove that Protobuf performs more quickly than JSON, XML and others [2]. JSON payloads are simple messages that have relatively slower serialization and deserialization performance. Beckn being an open network of BAPs, BGs, and BPPs there could be a very large number of participants active in the system. Hence low latency and efficient communication across the participating nodes would be beneficial to the ecosystem. Communication on beckn enabled networks is server-to-server, and two systems on a beckn network does not involve the client application hence it is best suitable for protobuf.

a. Stricter schema

Protocol Buffer requires the definition of a schema(.proto) for the data it is representing. This schema can be defined in a .proto file. And this file uses the construct called as a "message" to define data objects. Messages can contain attributes related to the objects similar to JSON.
Protobuf messages are not self describing, hence entity decoding the binary representation of the message needs to refer to the equivalent .proto definition of the message to interpret the fields/message. Without an accompanying schema(.proto), the decoder can't make sense of a protocol buffer data.
Hence it guarantees type-safety and prevents schema-violations. Protocol Buffer can easily convert the strongly typed messages into client and server understandable programming language.

e.g. sample schema for Context bean,

```
syntax = "proto3";

package beckn;

message Context {
 // Describes the domain of an object
 string domain = 1;
 string country = 2;
 string city = 3;
 ActionEnum action = 4;
 // Version of Beckn core API specification being used
 string coreUnderscoreversion = 5;
 // Unique id of the BAP. By default it is the fully qualified domain name of the BAP
 string bapUnderscoreid = 6;
…..
```

b. Forward and Backwards compatibility

Fields are numbered in protobuf schema because of this schema version checks can be avoided. Also with numbered field callers don't have to change their behaviour in order to maintain backward compatibility.

Also new fields can be introduced easily, and intermediate entities that didn't need to inspect the data could simply parse it and pass through the data without needing to know about all the fields. This minimizes the errors because of schema mismatch and helps in rolling out the features in future and evolution of schema over the period.

c. Partial Deserialization

As stated above, because of the numbered schema entity can deserialize only a part of the message (cases where schema is easy and does not use repeat construct). Beckn gateway can leverage this feature to avoid deserialization/parsing of complete messages received from BAP/BPP. It can define the minimum schema required to extract routing related information from the received message.

e.g

Search request message schema for BAP and BPP	Search request message schema for BG to extract only header

`syntax = "proto3";`
 	
`package beckn;`
 	
`import public "context.proto";`
`import public "search_message.proto";`
 	
`message SearchRequest {`
 	
`Context context = 1;`
 	
`SearchMessage message = 2;`
 	
`}`
	`syntax = "proto3";`
	
	`package beckn;`
	
	`import public "context.proto";`
	
	`message SearchRequest {`
	
	`Context context = 1;`
	
	`}`

d. Interoperability

proto3 version of Protocol buffers supports generated code in Java, Python, Objective-C, and C++, Kotlin, Dart, Go, Ruby, and C# which makes its adoption wider and interoperable.

2. Scope

Scope of this proposal is to optimize communication between participating nodes in the network by enhancing the serialization, de-serialization efforts. Further extension could be adaptation of gRPC(relies on HTTP/2 protocol). gRPC APIs provide the unique combination of low latency and high throughput communication.

3. Limitations
    a. Large Messages

    Protobuf is great for handling individual messages within a large data set. It is not optimal to encode large messages [3]. Usually, large data sets are really just a collection of small pieces, where each small piece may be a structured piece of data. Even though Protocol Buffers cannot handle the entire set at once, using Protocol Buffers to encode each piece greatly simplifies your problem.

Functional Specification

1. Adoption in Beckn

    Protobuf is a binary serialization format, it will be added as payload in the REST request.
    Hence content type needs to be set to application/x-protobuf(widely followed) instead of application/json.
    Use “Accept” header to indicate alternative encodings,

    - application/json: to return json response
    - application/xml: to return xml response
    - application/x-protobuf: to return protocol buffer binary response
        - When the protocol buffer is returned the HTTP response also needs to include an x-protobuf-schema header containing the URI of the .proto schema file. Schema files can be hosted by BG.

    Protobuf comes with toolings like native code generation functionality compatible with many popular programming language schemas. Following sample shows how search api json will look like when it is converted to .proto schema

Title: Protobuf message format for Bekcn Protocol

json	.proto

<pre>{
  "context": {
  "domain": "string",
  "country": "string",
  "city": "string",
  "action": "search",
  "core_version": "string",
  "bap_id": "string",
  "bap_uri": "string",
  "bpp_id": "string",
  "bpp_uri": "string",
  "transaction_id": "string",
  "message_id": "string",
  "timestamp": "2022-01-08T20:09:33.661Z",
  "key": "string",
  "ttl": "string"
  },
  "message": {
  "intent": {
  "provider": {
  "id": "string",
  "descriptor": {
  "name": "string",
  "code": "string",
  "symbol": "string",
  ..so on</pre>
	<pre>syntax = "proto3";
	
	message InlineObject {
	 Context context = 1;
	 Message message = 2;
	
	 message Context {
	  string domain = 1;
	  string country = 2;
	  string city = 3;
	  string action = 4;
	  string core_version = 5;
	  string bap_id = 6;
	  string bap_uri = 7;
	  string bpp_id = 8;
	  string bpp_uri = 9;
	  string transaction_id = 10;
	  string message_id = 11;
	  string timestamp = 12;
	  string key = 13;
	  string ttl = 14;
	 }
	
	 message Message{
	  Intent intent = 1;
	 }
	 ..so on</pre>

Receiver and sender of the request needs to map content-type to its corresponding message type encoder and decoder.
For example in case of Java mapping can be,

- application\json → `MappingJackson2HttpMessageConverter`
- application\x-protobuf → `ProtobufHttpMessageConverter` and so on.

By this way protocol implementers would be able to support more applicable encoder and decoders.

References

[1] : Protocol Buffers | Google Developers
[2] : Is Protobuf 5x Faster Than JSON? (Part 1) - DZone Performance, Beating JSON performance with Protobuf (auth0.com)
[3] : Techniques | Protocol Buffers | Google Developers
[4] : Protocol Buffers Best Practices for Backward and Forward Compatibility - Earthly Blog