ポリシーベースのテレメトリ ストリーミングの主要コンポーネント

ポリシーベースのテレメトリ データのストリーミングに使用される主要コンポーネントは次のとおりです。

テレメトリ ポリシー ファイル

テレメトリ ポリシー ファイルは、生成して受信者にプッシュするテレメトリ データの種類を指定するためにユーザによって定義されます。ポリシーは .policy 拡張子付きのテキスト ファイルに保存する必要があります。複数のポリシー ファイルを定義して、ルータ ファイル システムの `/telemetry/policies/` フォルダにインストールできます。

ポリシー ファイルには以下が含まれます。

- 1 つ以上の収集グループ。1 つの収集グループには、異なるインターバルでストリーミングされるさまざまなタイプのデータが含まれています。
- グループごとの秒単位の時間。
- グループごとの 1 つ以上のパス。
- ポリシーに関するバージョン、説明、およびその他の詳細情報を含むメタデータ。

ポリシー ファイルの構文

次に、ポリシー ファイルの例を示します。

```
{
"Name": "NameOfPolicy",
 "Metadata": {
     "Version": 25,
     "Description": "This is a sample policy to demonstrate the syntax",
     "Comment": "This is the first draft",
```

```
    "Identifier": "<data that may be sent by the encoder to the mgmt stn"
},
 "CollectionGroups": {
     "FirstGroup": {
         "Period": 10,
         "Paths": [
             "RootOper.MemorySummary.Node",
             "RootOper.RIB.VRF",
             "..."
         ]
     },
     "SecondGroup": {
         "Period": 300,
         "Paths": [
             "RootOper.Interfaces.Interface"
         ]
     }
 }
}
```

ポリシー ファイルの構文には以下が含まれています。

- **Name**：ポリシーの名前。以前の例では、ポリシーは `NameOfPolicy.policy` という名前のファイルに保存されます。ポリシーの名前はファイル名と一致する必要があります（.policy 拡張子を除く）。大文字のアルファベット、小文字のアルファベット、および数字を含めることができます。ポリシー名には大文字と小文字の区別があります。
- **Metadata**：ポリシーに関する情報。メタデータには、バージョン番号、日付、説明、作成者、著作権情報、その他のポリシーを特定する詳細情報を含めることができます。次のフィールドはポリシーを特定するのに重要です。
  - Description は **show policies** コマンド内に表示されます。
  - Version および Identifier は、テレメトリ メッセージのメッセージ ヘッダーの一部として受信側に送信されます。
- **CollectionGroups**：グループ名をそれらに関する情報にマップするエンコーダ オブジェクト。コレクション グループの名前には、大文字のアルファベット、小文字のアルファベット、および数字を含めることができます。グループ名では、大文字と小文字が区別されます。
- **Period**：各収集グループのパターン。この期間で、データが照会されてレシーバに送信される頻度を秒単位で指定します。この値は 5 ~ 86400 秒の範囲内に収める必要があります。
- **Paths**：データをストリーミングして受信者に送信するための 1 つ以上のスキーマ パス、許可されたリスト エントリまたはネイティブ YANG パス（コンテナ用の）。次に例を示します。

  スキーマ パス：

  ```
  RootOper.InfraStatistics.Interface(*).Latest.GenericCounters
  ```

  YANG パス：

  ```
  /Cisco-IOS-XR-infra-statsd-oper:infra-statistics/interfaces/interface=*/latest/generic-counters
  ```

  許可されたリスト エントリ

```
"RootOper.Interfaces.Interface(*)":
{
                "IncludeFields": ["State"]
}
```

スキーマ パス

テレメトリ データの収集場所を指定するには、スキーマ パスを使用します。参考のため、いくつかのパスのリストを次のテーブルに示します。

表 *1:* スキーマ パス

動作	パス
インターフェイス運用データ	RootOper.Interfaces.Interface(*)
パケット/バイト カウンタ	RootOper.InfraStatistics.Interface(*).Latest.GenericCounters
パケット/バイト レート	RootOper.InfraStatistics.Interface(*).Latest.DataRate
IPv4 パケット/バイト カウンタ	RootOper.InfraStatistics.Interface(*).Latest.Protocol(['IPV4_UNICAST'])
MPLS の統計情報	• RootOper.MPLS_TE.Tunnels.TunnelAutoBandwidth • RootOper.MPLS_TE.P2P_P2MPTunnel.TunnelHead • RootOper.MPLS_TE.SignallingCounters.HeadSignallingCounters
QOS の統計情報	• RootOper.QOS.Interface(*).Input.Statistics • RootOper.QOS.Interface(*).Output.Statistics
BGP データ	RootOper.BGP.Instance({'InstanceName': 'default'}).InstanceActive.DefaultVRF.Neighbor([*])
インベントリ データ	RootOper.PlatformInventory.Rack(*).Attributes.BasicInfo RootOper.PlatformInventory.Rack(*).Slot(*).Card(*).Sensor(*).Attributes.BasicInfo

テレメトリ エンコーダ

テレメトリ エンコーダは、生成したデータを目的の形式にカプセル化し、レシーバに送信します。

エンコーダはストリーミング テレメトリ API を呼び出して、以下を実行します。

- 明示的に定義するポリシーを指定する
- 対象のすべてのポリシーを登録する

テレメトリは次の 2 つのタイプのエンコーダをサポートしています。

- **JavaScript Object Notation（JSON）エンコーダ**

このエンコーダは IOS XR ソフトウェアとともにパッケージ化されており、デフォルト形式のテレメトリ データのストリーミングを提供します。CLI および XML で設定でき、特定のポリシーに合わせて登録することができます。設定はポリシー グループにグループ化されます。各ポリシー グループには、1 つ以上のポリシーと 1 つ以上の宛先が含まれます。JSON エンコーディングは、TCP トランスポート サービス経由でのみサポートされています。

JSON エンコーダは、2 つのエンコーディング形式をサポートしています。

- **Restconf 形式のエンコーディング**は、デフォルトの JSON エンコーディング形式です。
- **埋め込みキー エンコーディング**は、パス内の命名情報をキーとして扱います。

- **Google Protocol Buffers（GPB）エンコーダ**

  このエンコーダは代替の符号化メカニズムを提供し、UDP または TCP を使用して GPB 形式でデータをストリーミングします。CLI および XML で設定でき、JSON と同じポリシー ファイルを使用します。また、データを GPB 形式に変換するには、コンパイルされた .proto ファイル形式のメタデータが GPB エンコーダに必要です。

  GPB エンコーダは、2 つのエンコーディング形式をサポートしています。

  - **コンパクト エンコーディング**は、ストリーミングされるポリシーに固有のコンパクト GPB 構造にデータを格納します。この形式は、UDP と TCP の両方のトランスポート サービスで使用できます。結果ファイルを復号化するために受信者が使用するポリシー ファイル内のパスごとに、`.proto` ファイルを生成する必要があります。
  - **キー値エンコーディング**は、単一の `.proto` ファイルを使用して汎用キー値形式でデータを格納します。このエンコーディングは、キーがメッセージに含まれているので自己記述的です。この形式は、UDP および TCP トランスポート サービスで使用できます。受信者がデータを解釈できるため、`.proto` ファイルは各ポリシー ファイルには必要ありません。

TCP ヘッダー

JSON または GPB エンコーダのいずれかを使用して TCP 接続を介してデータをストリーミングし、必要に応じて zlib によって圧縮することで、データの各バッチの最後でストリームがフラッシュされます。これにより、受信者は受信したデータを展開できるようになります。データが zlib を使用して圧縮されている場合、圧縮はポリシー グループ レベルで行われます。受信側の圧縮解除プログラムの初期状態が空であるため、受信側から新しい接続が確立されると、コンプレッサがリセットされます。

各 TCP メッセージのヘッダーは次のとおりです。

タイプ	フラグ	長さ	メッセージ
4 バイト	4 バイト • **default**：フラグを設定しない場合は 0x0 の値を使用します。 • **zlib compression**：メッセージに zlib 圧縮を設定するには、0x1 の値を使用します。	4 バイト	変数

値は次のとおりです。

- タイプはビッグエンディアン値としてエンコードされます。
- 長さ（バイト単位）はビッグエンディアン値としてエンコードされます。
- フラグは、ビッグエンディアン形式の修飾子（圧縮など）を示します。
- メッセージには、JSON または GPB オブジェクトのストリーミングされたデータが含まれています。

メッセージのタイプは次のとおりです。

タイプ	名前	長さ	値
1	リセット コンプレッサ	0	値なし
2	JSON メッセージ	変数	JSON メッセージ（任意の形式）
3	GPB コンパクト	変数	コンパクト形式の GPB メッセージ
4	GPB キー値	変数	キー値形式の GPB メッセージ

JSON メッセージ形式

JSON メッセージは TCP を介して送信され、TCP ヘッダー （4 ページ） で説明されているヘッダー メッセージを使用します。

このメッセージは、次の JSON オブジェクトで構成されています。

```
{
"Policy": "<name-of-policy>",
```

```
"Version": "<policy-version>",
"Identifier": "<data from policy file>"
"CollectionID": <id>,
"Path": <Policy Path>,
"CollectionStartTime": <timestamp>,
"Data": { … object as above … },
"CollectionEndTime": <timestamp>,
}
```

値は次のとおりです。

- `Policy`、`Version` および `Identifier` は、ポリシー ファイルで指定されます。
- `CollectionID` は、単一パスのデータが複数のメッセージに分割されている場合にメッセージをグループ化するための整数です。
- `Path` は、ポリシー ファイルで指定されている対応するデータのベース パスです。
- `CollectionStartTime` と `CollectionEndTime` は、データの収集時点を示すタイムスタンプです。

JSON メッセージはルータのデータ モデルの階層を反映しています。階層は次のように構成されています。

- コンテナ：タイプに応じて異なるノードがあります。
- テーブル：テーブルにもノードがありますが、子ノードの数は異なる場合があり、同じタイプである必要があります。
- リーフ ノード：整数や文字列などのデータ値を含んでいます。

スキーマ オブジェクトは次のように JSON にマップされます。

- 各コンテナは JSON オブジェクトにマップされます。キーはノードのスキーマ名を表す文字列です。この値はノードの値を表します。
- また、テーブルを表すために JSON オブジェクトも使用されます。この場合は、キーは文字列形式に変換される命名情報に基づきます。命名情報をエンコードするための 2 つのオプションが提供されます。
  - デフォルトは restconf 形式のエンコーディングです。このエンコーディングでは、ネーミング パラメータは参照先の子ノード内に含まれます。
  - 埋め込みキー オプションは、命名情報を JSON ディクショナリのキーとして使用し、対応する子ノードが値を形成します。
- リーフのデータ型は次のようにマップされます。
  - 簡単な文字列、整数、およびブール値は直接マッピングされます。
  - 列挙値は値の文字列表現として保存されます。
  - IP アドレスなどのその他のシンプルなデータ型は文字列としてマップされます。

例：Rest-conf エンコーディング

たとえば、次のパスを検討します。

```
Interfaces(*).Counters.Protocols(“IPv4”)
```

これには、インターフェイス名とプロトコル名の2つのネーミング パラメータがあり、パケット カウンタとバイト カウンタであるリーフ ノードを保持するコンテナを表しています。これは次のように表されます。

```
{
  "Interfaces": [
    {
      "Name": "GigabitEthernet0/0/0/1"
      "Counters": {
        "Protocols": [
          {
            "ProtoName": "IPv4",
            "CollectionTime": 12345678,
            "InputPkts": 100,
            "InputBytes": 200,
          }
        ]
      }
    },{
      "Name": "GigabitEthernet0/0/0/2"
      "Counters": {
        "Protocols": [
          {
            "ProtoName": "IPv4",
            "CollectionTime": 12345678,
            "InputPkts": 400,
            "InputBytes": 500,
          }
        ]
      }
    }
  ]
}
```

複数のキーを含むネーミング パラメータ（たとえばFoo.Destination(IPAddress=1.1.1.1, Port=2000)）は、次のように表されます。

```
{
  "Foo":
    {
    "Destination": [
      {
        "IPAddress": 1.1.1.1,
        "Port": 2000,
        "CollectionTime": 12345678,
        "Leaf1": 100,
      }
    ]
  }
}
```

例：埋め込みキー エンコーディング

埋め込みキーエンコーディングは、パス内の命名情報をJSONディクショナリのキーとして扱います。キー名情報は失われ、階層には余分なレベルがありますが、解析時にコレクタの役に

立つ可能性があるキーがどのデータによって構成されているかがより明確になります。このオプションは主に 6.0 との下位互換性のために提供されます。

```
{
  "Interfaces": {
      "GigabitEthernet0/0/0/1": {
        "Counters": {
          "Protocols": {
            "IPv4": {
              "CollectionTime": 12345678,
              "InputPkts": 100,
              "InputBytes": 200,
            }
          }
        }
      },
      "GigabitEthernet0/0/0/2": {
        "Counters": {
          "Protocols": {
            "IPv4": {
              "CollectionTime": 12345678,
              "InputPkts": 400,
              "InputBytes": 500,
            }
          }
        }
      }
    }
  }
}
```

複数のキーを含むネーミング パラメータ（たとえばFoo.Destination(IPAddress=1.1.1.1, Port=2000)）は、各キーを順番にネストすることによって次のように表されます。

```
{
  "Foo":
    {
    "Destination": {
        1.1.1.1: {
          2000: {
            Leaf1": 100,
          }
        }
      }
    ]
  }
}
```

GPB メッセージ形式

GPB エンコーダの出力は GPB によって全体的に構成され、単一のパケット内に複数のテーブルを許可して拡張性を持たせています。

GPB（Google プロトコル バッファ）エンコーダには、コンパイル済みの `.proto` ファイル形式のメタデータが必要です。`.proto` ファイルはデータのストリーミングに使用する GPB メッセージ形式を記述します。

UDP の場合、データは単なる GPB です。メッセージが `TelemetryHeader` メッセージとして解釈できるように、コンパクト形式のみがサポートされています。

TCP の場合、メッセージ本文は、次のエンコーディング タイプのどれが設定されているかに応じて、`Telemetry` メッセージまたは `TelemetryHeader` メッセージのいずれかです。

- **コンパクト GPB 形式**は、データを圧縮された非自己記述形式で格納します。結果ファイルを復号化するために受信者が使用するポリシー ファイル内のパスごとに、`.proto` ファイルを生成する必要があります。
- **キー値 GPB 形式**は、単一の `.proto` ファイルを使用して、データを自己記述形式でエンコードします。このエンコーディングでは、各パスに `.proto` ファイルは必要ありません。キー名が含まれているため、ワイヤ上のデータはさらに大きくなります。

次の例では、ポリシーグループ *alpha* は、コンパクト エンコーディングと UDP トランスポートのデフォルト設定を使用します。ポリシー グループ *beta* は、圧縮された TCP とキー値エンコーディングを使用します。ポリシーグループ *gamma* は圧縮されていない TCP 上でコンパクト エンコーディングを使用します。

```
telemetry policy-driven encoder gpb
  policy group alpha
    policy foo
    destination ipv4 192.168.1.1 port 1234
    destination ipv4 10.0.0.1 port 9876
  policy group beta
    policy bar
    policy whizz
    destination ipv4 10.20.30.40 port 3333
    transport tcp
      compression zlib
  policy group gamma
    policy bang
    destination ipv4 11.1.1.1 port 4444
    transport tcp
      encoding-format gpb-compact
```

コンパクト GPB 形式

コンパクト GPB 形式は、大量のデータを頻繁な間隔でストリーミングするためのものです。この形式は、ワイヤ上のメッセージのサイズを最小に抑えます。拡張性のために、複数のテーブルを単一のパケットで送信できます。

（注） テーブルは複数のパケットに分割できますが、行の断片化はサポートされていません。テーブル内の行が大きすぎて単一のUDPフレームに収まらない場合は、ストリーミングできません。代わりに、TCP に切り替えるか、MTU を増やすか、または .proto ファイルを変更します。

次の .proto ファイルは、エンコーダによって送信されるすべてのパケットに共通するヘッダーを示しています。

```
message TelemetryHeader {
  optional uint32 encoding = 1

  optional string policy_name = 2;
  optional string version = 3;
  optional string identifier = 4;
```

```
  optional uint64 start_time = 5;
  optional uint64 end_time = 6;

  repeated TelemetryTable tables = 7;
  }

message TelemetryTable {
  optional string policy_path = 1;
  repeated bytes  row = 2;
  }
```

値は次のとおりです。

- encoding は、レシーバが使用してパケットが有効かどうかを確認します。
- policiy_name、version、および identifier は、ポリシー ファイルから取得したメタデータです。
- start_time と end_time は、データが収集された時間を示します。
- tables はパケット内のテーブルのリストです。この形式は、1 つのパケットで複数のスキーマ パスの結果を受け取ることができることを示します。
- 各テーブルの値は、次のとおりです。
  - policy_path はスキーマ パスです。
  - row は、符号化された GPB を表す 1 つ以上のバイト配列です。

キー値 GPB 形式

自己記述型キー値 GPB 形式では、汎用の .proto ファイルが使用されます。このファイルは、データを一連のキーと値のペアとしてエンコードします。フィールド名は、受信者がデータを解釈できるように出力に含まれています。

次の .proto ファイルには、キーと値のペアを含むフィールドが示されています。

```
message Telemetry {
  uint64   collection_id = 1;
  string   base_path = 2;
  string   subscription_identifier = 3;
  string   model_version = 4;
  uint64   collection_start_time = 5;
  uint64   msg_timestamp = 6;
  repeated TelemetryField fields = 14;
  uint64   collection_end_time = 15;
}

message TelemetryField {
  uint64         timestamp = 1;
  string         name = 2;
  bool           augment_data = 3;
  oneof value_by_type {
    bytes          bytes_value = 4;
    string         string_value = 5;
    bool           bool_value = 6;
    uint32         uint32_value = 7;
    uint64         uint64_value = 8;
    sint32         sint32_value = 9;
    sint64         sint64_value = 10;
```

```
    double          double_value = 11;
    float           float_value = 12;
  }
  repeated TelemetryField fields = 15;
}
```

値は次のとおりです。

- collection_id、base_path、collection_start_time、および collection_end_time は、ストリーミングの詳細を提供します。
- subscription_identifier は、パターン駆動型テレメトリの固定値です。イベント駆動型データから区別するために使用されます。
- model_version には、必要に応じて、データ モデルのバージョンに使用される文字列が含まれています。

テレメトリ レシーバ

テレメトリ レシーバは、宛先として使用してストリーミングされたデータを保存します。

JSON と GPB の両方のエンコーディングを処理するサンプル レシーバについては、https://github.com/cisco/bigmuddy-network-telemetry-collector をご覧ください。

GPB レシーバ用のコードをコンパイルするには、`cisco.proto` ファイルのコピーが必要です。`cisco.proto` ファイルは http://github.com/cisco/logstash-codec-bigmuddy-network-telemetry-gpb/tree/master/resources/xr6.0.0 で入手できます。

独自のコレクタを構築する場合は、標準の protoc コンパイラを使用します。たとえば、GPB コンパクト エンコーディングの場合は次のようになります。

```
protoc --python_out . -I=/sw/packages/protoc/current/google/include/:.
generic_counters.proto ipv4_counters.proto
```

値は次のとおりです。

- --python_out <out_dir> で、結果の生成されたファイルの場所を指定します。これらのファイルの形式は `<name>_pb2.py` です。
- -I <import_path> で、インポートを検索するパスを指定します。これには、Google からの `descriptor.proto` の場所を含める必要があります。（in `/sw/packages`）および `cisco.proto` とコンパイルされた .proto ファイル。

上記の例で示したすべてのファイルがローカル ディレクトリに存在します。

テレメトリ レシーバ