Schemaless HTTP Design

Motivation

Currently, ekuiper's external services only support schema files of protobuf type and strictly enforce parameter type checks during usage. However, there are instances where users may prefer to utilize REST services of a schemaless nature, without the need for defining APIs. Therefore, in order to cater to the users' requirements, we need to implement HTTP external services that support schemaless functionality.

Design

Option one

1. Add a "schemaless" field to the JSON configuration file that describes service information

```
{
        "about":{},
        "interfaces": {
                "bookshelf": {
                "address": "http://localhost:51234/bookshelf",
                "protocol": "rest",
                "options": {
                        "insecureSkipVerify": true,
                        "headers": {
                                "Accept-Charset": "utf-8"
                        }
                },
                "schemaType": "protobuf",
                "schemaFile": "http_bookstore.proto",
                "schemaless": false,
                "functions": [
                        {
                                "name": "createBook",
                                "serviceName": "CreateBook"
                        }
                ]
                },
        }

}
```

2. Introduce a new JSON format to represent the mapping of external functions to HTTP services

```
[
  {
    "serviceName": "hello",
    "functions": [
      {
        "name": "hello_get",
        "method": "get",
        "uri": "/hello"
      },
      {
        "name": "hello_post",
        "method": "post",
        "uri": "/hello"
      }
    ]
  }
...
]
```

Implementation

1. Modify the structure "schemaInfo" and its related functions

```
type schemaInfo struct {
        Schemaless bool
        SchemaType schema
        SchemaFile string
}


func parse(schema schema, file string, schemaless bool) (descriptor, error) {

        info := &schemaInfo{
                Schemaless: schemaless,
                SchemaType: schema,
                SchemaFile: file,
        }
        switch schema {
                case PROTOBUFF:
                        if schemaless {
                                return nil, fmt.Errorf("unsupported schema %s for schemaless",
schema)
                        }
                        ...
                case JSON:
                        if !schemaless {
                                return nil, fmt.Errorf("unsupported schema %s for schema", schema)
                        }
                        // Parse the JSON file representing the mapping of external functions to HTTP
services into the "wrappedJsonDescriptor" structure.
                default:
                        return nil, fmt.Errorf("unsupported schema %s", schema)
        }
}
```

2. Add the "wrappedJsonDescriptor" structure to implement the "descriptor" and "multiplexDescriptor" interfaces

```
func (d *wrappedJsonDescriptor) ConvertParamsToJson(_ string, params []interface{}) ([]byte, error) {
        if len(params) == 0 {
                return []byte{}, nil
        } else if len(params) == 1 {
                return json.Marshal(params[0])
        }

        return json.Marshal(params)
}

func (d *wrappedJsonDescriptor) ConvertHttpMapping(method string, params []interface{}) (*httpConnMeta,
error) {
        hcm := &httpConnMeta{}
        var (
                json []byte
                err error
        )

        json, err = d.ConvertParamsToJson(method, params)
        if err != nil {
                return nil, err
        }
        hcm.Body = json
        if d.Methods[method] == nil {
                return nil, fmt.Errorf("cannot find method %s", method)
        }
        hcm.Method = "POST"
        if d.Methods[method].method != "" {
                hcm.Method = strings.ToUpper(d.Methods[method].method)
        }
        hcm.Uri = "/" + method
        if d.Methods[method].uri != "" {
                hcm.Uri = d.Methods[method].uri
        }
        return hcm, nil
}

func (d *wrappedJsonDescriptor) ConvertReturnText(_ string, returnVal []byte) (interface{}, error) {
        var result interface{}
        err := json.Unmarshal(returnVal, &result)
        if err != nil {
                return returnVal, nil
        }
        return result, nil
}
```

3. Introduce the "schemalessService" structure to read the JSON file that represents the mapping of external functions to HTTP services

```
type schemalessService struct {
        ServiceName string `json:"serviceName"`
        Functions   []*schemalessFunction
}

type schemalessFunction struct {
        Name   string `json:"name"`
        Method string `json:"method"`
        Uri    string `json:"uri"`
}

func parseSchemalessFiles(file string) (*wrappedJsonDescriptor, error) {
...
}
```

Option Two

1. Add a "schemaless" field to the JSON configuration file that describes service information

```
"messaging": {
     "address": "http://localhost:51234/messaging",
     "protocol": "rest",
     "options": {
       "insecureSkipVerify": true,
       "headers": {
         "Accept-Charset": "utf-8"
       }
     },
         "schemaless": true
   }
```

2. The user directly invokes schemaless external services by specifying the service name, followed by the method, route, and the incoming data as arguments

```
SELECT messaging("POST", "", *) FROM demo;
```