Nulstar

MESSAGE PROTOCOL

V 1.2

1] Communication Protocol – Json/WebSockets

Microservices do not behave like a standard client-server infrastructure as every microservice is a client and a server at the same time therefore a full duplex communication protocol is needed, which allows the implementation of a special type of publish-subscription pattern; in this implementation methods can be invoked just once and the caller could receive constant updates afterwards in two different ways:

- Event based: When the method sends notifications after a predefined number of events
- Period based: When the method sends notifications after a predefined number of seconds

Web Sockets is a mature option that offers full duplex connectivity natively, other alternatives as standard Json-RPC do not offer bidirectional channels.

The messages will be encoded using JSon format as it is the most widely used for message interchange and it is easy to debug.

2] Message Structure

All messages will have a common base structure composed of six fields:

- ProtocolVersion: Represents the protocol version that the caller needs the service to understand, it is composed by two numbers, major and minor and follows semantic rules, which means that if the major number is different the connection is refused, if the minor number varies then a successful connection can be established.
- MessageID: This is a string that identifies a request. Its length should not surpass 256 characters.
- Timestamp: Number of seconds since epoch (January 1, 1970 at 00:00:00 GMT)
- TimeZone: The time zone where the request was originated and it is represented as a number between -12 and 12
- MessageType: The message type, these are specified on section 3]
- MessageData: A Json object that holds the payload of the message.

Example:

```
{
    "ProtocolVersion": "1.0",
    "MessageID": "45sdj8jcf8899ekffEFefee",
    "Timestamp": "1542102459",
    "TimeZone": "-4",
    "MessageType": "NegotiateConnection",
    "MessageData": {
        "CompressionRate": "3"
        "CompressionAlgorithm": "zlib"
    }
}
```

3] Message Types

Only nine types of messages are currently defined: NegotiateConnection, NegotiateConnectionResponse, Request, Unsubscribe, Response, Ack, Notificatioin, RegisterCompoundMethod and UnregisterCompoundMethod.

3.1] NegotiateConnection

This should be the first object that should be sent when establishing a connection with a microservice, only if the negotiation is successful the service may process further requests otherwise a NegotiateConnectionResponse object should be received with Status set to 0 (Failure) and disconnect immediately.

It is composed by two fields:

- CompressionAlgorithm (Default: zlib): A String that represents the algorithm that will be used to receive and send messages if CompressionRate is greater than 0. The default is zlib which a library is available in most development languages.
- CompressionRate: An integer between 0 and 9 that establishes the compression level in which the messages should be sent and received for this connection. 0 means no compression while 9 maximum compression

Example:

```
{
  ......
  "MessageType": "NegotiateConnection",
  "MessageData": {
        "CompressionAlgorithm": "zlib",
        "CompressionRate": "3"
    }
}
```

3.2] NegotiateConnectionResponse

A message of this type should be sent by a service only in response to a previous incoming NegotiateConnection message. It is composed by two fields:

- NegotiationStatus: An unsigned small integer value, 0 if negotiation was a failure and 1 if it was successful
- NegotiationComment: A string value, useful to describe what exactly went wrong when the connection was rejected.

Example:

```
{
  ......
  "MessageType": "NegotiateConnectionReponse",
  "MessageData": {
        "NegotiationStatus": "0",
        "NegotiationComment": "Incompatible protocol version"
    }
}
```

3.3] Request

A caller service must send a Request object to execute a method offered by some service inside the Nulstar microservice's network.

If two or more methods are enclosed in a single request object, then the methods should be executed in sequential order and a Response should be sent consolidating

all responses, if one of the requests fails then the whole operation is considered a failure.

It is composed by five fields:

- RequestAck (Default: 0): This is a boolean value.
    - 0: The Microserver that made the request expects only a Response message, if it subscribed to the function then it may expect many Response messages.
    - 1: The Microserver that made the request expects exactly one Ack message and also a Response message, if it subscribed to the function then it may expect many Response messages.
- SubscriptionEventCounter: This is an unsigned integer that specifies how many events do the target methods need to process before sending back another Response request, the first Response is always sent as soon as possible.

  For example, if the requested method is GetHeight and this parameter is set to 5 then the service will send back responses only after 5 blocks have been processed.

  0 means the method should send a Response only once; this is the default value.
- SubscriptionPeriod: This is an unsigned integer that specifies how many seconds do the target methods need to wait before sending back another Response request, the first Response is always sent as soon as possible.

  For example, if the requested method is GetHeight and this parameter is set to 5 then the service will send back responses only after 5 seconds have passed.

  0 means the method should send a Response only once; this is the default value.
- SubscriptionRange: If the event defined in the target microservice returns a number, this is a string that represents the set of numbers that will trigger a Response. The string is a pair of signed decimal numbers, the first one is the lower bound, empty if not available and the second one is the higher bound. If the pair starts or ends with "(" or ")" respectively then it means that the number is not included, if the pair pair starts or ends with "[" or "]" respectively then it means that the number is included.

Example: Assume we only want to be notified only when the balance is equal or greater to 1000. Then the getbalance request should be sent with "[1000,)" string as SubscriptionRange parameter.

- ResponseMaxSize: An unsigned integer which specifies the maximum number of objects that the method should return, a value of zero (the default) means no limit
- RequestMethods: An array that holds all methods being requested with their respective parameters

Example:

```
{
 ......
 "MessageType": "Request",
 "MessageData": {
       "RequestAck": "0",
       "SubscriptionEventCounter": "3",
       "SubscriptionPeriod": "0",
       "SubscriptionRange": "0",
       "ResponseMaxSize: "0",
       "RequestMethods": [
          {
             "GetBalance": {
                  "Address": "N234rFr4Rtgg5ref4$45tgg5f43335emcnd"
             }
          },
          {
             "GetHeight": {
             }
          }
       ]
   }
}
```

In this example the caller will get one response after the target service has processed 3 blocks, an 'event' is defined by the target service and corresponds always to the first method requested

3.4] Unsubscribe

When a service no longer wants to receive Responses from the method it subscribed then it must send an Unsubscribe message to the target service.

It is composed by one field:

- UnsubscribeMethods: An array that holds all methods that the caller wants to unsubscribe

Example:

```
{
  ……
  "MessageType": "Unsubscribe",
  "MessageData": {
        "UnsubscribeMethods": [
            "GetBalance",
            "GetHeight"
        ]
    }
}
```

3.5] Response

When the target service finished processing a request, a Response should be sent with the result of the operation. Responses should be sent ONLY when a Request message was received previously, if the Request message subscribes to a function then many Responses may be sent referring to the same Request.

It is composed by six fields:

- RequestID: This is the original request ID referred by a Request message
- ResponseProcessingTime: The time that the target service took to process the request in miliseconds.
- ResponseStatus: The response status, 1 if successful, 0 otherwise.
- ResponseComment: A string that could offer more clarification about the result of the process.
- ResponseMaxSize: The maximum number of objects that the response contains per request.
- ResponseData: An object array that contains the result of the method processed, one object per request.

Example:

```
{
 ......
 "MessageType": "Response",
 "MessageData": {
       "RequestID": "sdj8jcf8899ekffEFefee",
       "ResponseProcessingTime": "13",
       "ResponseStatus": "1"
       "ResponseComment": "Congratulations! Processing was completed successfully!"
       "ResponseMaxSize": "0"
       "ResponseData": [
          {
             "Balance": "25000"
          },
          {
             "Height": "45454655454"
          }
       ]
   }
}
```

3.6] Ack

This message type is sent when the RequestType is 2 or 3 (See section 3.3). Its only purpose is to notify the caller that the request was received successfully.

It is composed by one field:

- RequestID: This is the original request ID referred by a Request message

Example:

```
{
 ......
 "MessageType": "Ack",
 "MessageData": {
       "RequestID": "sdj8jcf8899ekffEFefee",
   }
}
```

3.7] Notification

This message type is sent when a service needs to notify some event to the connected components without expecting a Response message, for example when an upgrade is about to be performed on a service. Notifications pushes information to other components instead of pulling, therefor Notifications should be used only by the Manager, Controller and Connector modules.

It is composed by four fields:

- NotificationAck: (Default: 0): This is a boolean value.
    - 0: The Microserver that made the notification does not expect any kind of message in return.
    - 1: The Microserver that made the notification expects exactly one Ack message.
- NotificationType: The category of the notification, each service may define its own types so it is not required that the target service processes this field.
- NotificationComment: A string comment that provides more information about the reason of the notification
- NotificationData: Data relevant to the notification, it is not required the target service to process this field. Example:

```
{
  ……
  "MessageType": "Notification",
  "MessageData": {
        "NotificationAck": "1",
        "NotificationType": "SystemUpgrade",
        "NotificationComment": "A system upgrade is about to be performed!"
        "NotificationData": [
           {
              "Date": "2018-11-11"
           },
           {
              "Time": "23:00:00"
           },
           {
             "NewVersion": "1.1.6"
           }
        ]
   }
}
```

3.8] RegisterCompoundMethod

As explained in 3.3] a request may be composed by several methods, with this message type we register a virtual method that will execute its individual real methods in sequential order, if one of its child methods fails then the virtual method returns failure.

It is possible that some child methods share the same parameter name, therefore aliases may be created as shown in the example.

It is composed by three fields:

- CompoundMethodName: This is a string identifying the virtual method.
- CompoundMethodDescription: A string describing the functionality of the method, the description will be available when querying the API for help.
- CompoundMethods: This is an array containing the methods with their respective parameter aliases that make up the virtual method.

Example:

```
{
    ……
   "MessageType": "RegisterCompoundMethod":,
   "MessageData": {
      "CompoundMethodName": "GetMyInfo"
      "CompoundMethodDescription": "Get useful information."
      "CompoundMethods": [
         {
            "GetBalance": {
                  "Address": "GetBalanceAddress"
            }
         },
         {
            "GetHeight": {
            }
         }
      ]
   }
}
```

In the example a virtual method called GetMyInfo is being registered that it is composed by GetBalance and GetHeight methods, also an alias was created for the Address parameter called GetBalanceAddress.

Requests for GetMyInfo may be sent as it was a standard method.

3.9] UnregisterCompoundMethod

This message type is used to unregister a compound (virtual) method.

It is composed by one field:

- UnregisterCompoundMethodName: This is the string that identifies the virtual method. If it is empty then all virtual methods registered by the caller should be unregistered.

Example:

```
{
    ……
   "MessageType": "UnregisterCompoundMethod",
   "MessageData": {
     "UnregisterCompoundMethodName": "GetMyInfo"
   }
}
```