Nulstar

MESSAGE PROTOCOL

V 1.0

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 Types

Only eight types of messages are currently defined: NegotiateConnection, NegotiateConnectionResponse, Request, Unsubscribe, Response, Notification, RegisterCompoundMethod and UnregisterCompoundMethod.

2.1] 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:

- 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.

- 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:

```
{
   ”NegotiateConnection”: {
       "ProtocolVersion": ”1.0”,
       "CompressionRate": ”3”
   }
}
```

2.2] 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 three fields:

- ProtocolVersion: Represents the protocol version that the target microservice is able to understand, it is sent in case the caller needs to make a special provision if the minor version is different.
- 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:

```
{
   ”NegotiateConnectionResponse”: {
       ”ProtocolVersion”: ”2.0”,
       ”NegotiationStatus”: ”0”,
       ”NegotiationComment”: ”Incompatible protocol version”
   }
}
```

2.3] 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 compound requests fails then the whole operation is considered a failure.

It is composed by eight fields:

- RequestID: This is a string that identifies a request. Its length should not surpass 256 characters and should not change.
- RequestInternalID: This is 64bit unsigned integer that identifies a request and it is filled only by internal microservices, the reason for this field to exist is to assure that a request is identified in a strictly unique manner because multiple applications can be connected to a single node, each of them having their own ID scheme. This field should no be filled by an external application and can be changed by services at will for internal control.
- RequestDate: The date the request was sent using the callers time zone
- RequestTime: The time the request was sent using the callers time zone
- RequestTimeZone: The time zone where the request was originated and it is represented as a number between -12 and 12
- 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.

- 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:

```
{
   "Request": {
      "RequestID": "sdj8jcf8899ekffEFefee"
      "RequestInternalID": "348022847492"
      "RequestDate": "2018-10-05",
      "RequestTime": "03:00:00",
      "RequestTimeZone": "-4",
      "SubscriptionEventCounter": "3",
      "SubscriptionPeriod": "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

2.4] 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 four fields:

- UnsubscribeDate: The date the unsubscribe request was sent using the callers time zone
- UnsubscribeTime: The time the unsubscribe request was sent using the callers time zone
- UnsubscribeTimeZone: The time zone where the unsubscribe request was originated and it is represented as a number between -12 and 12
- UnsubscribeMethods: An array that holds all methods that the caller wants to unsubscribe

Example:

```
{
   "Unsubscribe": {
      "UnsubscribeDate": "2018-10-05",
      "UnsubscribeTime": "03:00:00",
      "UnsubscribeTimeZone": "-4",
      "UnsubscribeMethods": [
          "GetBalance",
          "GetHeight"
       ]
   }
}
```

2.5] Response

When the target service finished processing a request, a Response should be sent with the result of the operation.

It is composed by eleven fields:

- ResponseID: This is 64bit unsigned integer that uniquely identifies a response
- RequestID: This is the original request ID referred by a Request message

- RequestInternalID: This is the original internal request ID referred by a Request message.
- ResponseDate: The date the response was sent.
- ResponseTime: The time the response was sent.
- ResponseTimeZone: The time zone where the response was originated and it is represented as a number between -12 and 12
- ResponseProcessingTimeM: 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:


```
{
   "Response": {
      "ResponseID": "965567634256"
      "RequestID": "sdj8jcf8899ekffEFefee"
      "RequestInternalID": "348022847492"
      "ResponseDate": "2018-10-05",
      "ResponseTime": "03:00:00",
      "ResponseTimeZone": "-4",
      "ResponseProcessingTimeM": "13",
      "ResponseStatus": "1"
      "ResponseComment": "Congratulations! Processing was completed successfully!"
      "ResponseMaxSize": "0"
      "ResponseData": [
         {
            "Balance": "25000"
         },
         {
            "Height": "45454655454"
         }
      ]
   }
}
```

2.6] Notification

This message type is sent when a service needs to notify some event to the connected components without expecting some kind of response, for example when an upgrade is about to be performed on a service.

It is composed by seven fields:

- NotificationID: This is 64bit unsigned integer that uniquely identifies a notification.
- NotificationDate: The date the notification was sent.
- NotificationTime: The time the notification was sent.
- NotificationTimeZone: The time zone where the notification was originated and it is represented as a number between -12 and 12
- 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 rquiered the target service to process this filed.

Example:

```
{
   "Notification": {
      "NotificationID": "797531667099"
      "NotificationDate": "2018-10-05",
      "NotificationTime": "03:00:00",
      "NotificationTimeZone": "-4",
      "NotificationType": "ServiceUpgrade"
      "NotificationComment": "A service upgrade is about to be performed!"
      "NotificationData": [
         {
            "Date": "2018-11-11"
         },
         {
            "Time": "23:00:00"
         },
         {
            "NewVersion": "1.1.6"
         }
      ]
   }
}
```

2.7] RegisterCompoundMethod

As explained in 2.3] 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 six 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.
- RegisterCompoundMethodDate: The date the registration request was sent.
- RegisterCompoundMethodTime: The time the registration request was sent.
- RegisterCompoundMethodTimeZone: The time zone where the registration request was originated and it is represented as a number between -12 and 12
- CompoundMethods: This is an array containing the methods with their respective parameter aliases that make up the virtual method.

Example:

```
{
   "RegisterCompoundMethod": {
      "CompoundMethodName": "GetMyInfo"
      "CompoundMethodDescription": "Get useful information."
      "RegisterCompoundMethodDate": "2018-11-05",
      "RegisterCompoundMethodTime": "04:00:03",
      "RegisterCompoundMethodTimeZone": "-4",
      "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.

2.7] UnregisterCompoundMethod

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

It is composed by four fields:

- UnregisterCompoundMethodName: This is the string that identifies the virtual method.
- UnregisterCompoundMethodDate: The date the request was sent.
- UnregisterCompoundMethodTime: The time the request was sent.
- UnregisterCompoundMethodTimeZone: The time zone where the registration request was originated and it is represented as a number between -12 and 12

Example:

```
{
   "UnregisterCompoundMethod": {
      "UnregisterCompoundMethodName": "GetMyInfo"
      "UnregisterCompoundMethodDate": "2018-10-05",
      "UnregisterCompoundMethodTime": "03:00:00",
      "UnregisterCompoundMethodTimeZone": "-4",
   }
}
```