Version History

The version numbering is as follows:

The initial version is 1.0

VERSION NUMBER	REVISION DATE	AUTHOR	SUMMARY OF CHANGE
1.0	09/07/2017	VG411H	Initial Version

Message Router APIs

HTTP Service APIs:

DMaaP Message Router utilizes an HTTP REST API to service all Publish and Consume transactions. HTTP and REST standards are followed so clients as varied as CURL, Java applications and even Web Browsers will work to interact with Message Router. Message Router uses AAF for user's authentication and authorization.

General HTTP Requirements:

A DMaaP Message Router transactions consists of 4 distinct segments, HTTP URL, HTTP Header, HTTP Body (POST) and HTTP Response. The general considerations for each segment are as follows and are required for each of the specific transactions described in this section.

HTTP URL

http[s]://Username:Password@serverBaseURL{/routing}{resourcePath}

- The Username:Password utilizes HTTP Basic Authentication and HTTPS/TLS to securely transmit the authorization and authentication credentials that AAF needs to validate the client's access to the requested resource.
- The serverBaseURL points to DMaaP Message Router host/port that will service the request. Optionally DME2 service end points for Message Router can be used.
- The resourcePath specifies the specific service, or Topic, that the client is attempting to reach

HTTP Header

Specifies HTTP Headers, such as Content-Type, that define the parameters of the HTTP Transaction

HTTP Body

The HTTP Body contains the topic content when Publishing or Consuming. The Body may contain topic messages in several formats (like below) but it must be noted, that, except in very specific circumstances, messages are not inspected for content.

<table>
<tr><th>Content-Type</th><th>Description</th></tr>
<tr><td>text/plain</td><td>Each line in the POST body is treated as a separate message. No partition key is specified, and therefore no order is guaranteed. This format is mainly for test, as messages are highly likely to be re-ordered when delivered through the Kafka cluster.</td></tr>
<tr><td>application/json</td><td>The payload maybe a single JSON object or a JSON array of JSON objects. Each object is handled as an individual message. .
Note that use of this format may result in equivalent but altered JSON objects sent to consumers. That's because MR uses a standard JSON parser to read each object into memory before pushing the object to the Kafka system. At that point, the JSON object is re-written from the in-memory object. This can result in re-ordered fields or changes in whitespace. If you want to preseve JSON objects exactly, use application/cambria.

Recommended to follow the JSON format after validating the message in https://jsonformatter.curiousconcept.com/</td></tr>
<tr><td>application/cambria</td><td>@deprecated: The application/json payload requires Cambria to parse every inbound message for a partition name. The application/cambria format speeds things up a bit by having the client write a quick-to-parse version of the same information. In this format, messages start with two length values that designate the length of the key string and the length of the message. Length values are positive base-10 integers and are terminated with a period (aka dot, "."). The key and message body follow the second length without delimiters (their size is known). Whitespace leading the first length value is ignored.
For example:

1.11.AMessageBody

3.3.123Foo3.3.123Bar

0.16.You can do that..8.Or that.

This post contains 5 messages:
<table>
<tr><th>Partition Key Length</th><th>Message Length</th><th>Partition Key</th><th>Message</th></tr>
<tr><td>1</td><td>11</td><td>A</td><td>MessageBody</td></tr>
<tr><td>3</td><td>3</td><td>123</td><td>Foo</td></tr>
<tr><td>3</td><td>3</td><td>123</td><td>Bar</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Content-Type</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><td>0</td><td>16</td><td></td><td>You can do that.</td></tr>
<tr><td>0</td><td>8</td><td></td><td>Or that.</td></tr>
</table>
Things to note about the example:

New lines are whitespace. Because whitespace before the first length value is ignored, we can optionally put messages on separate lines for (human) readability.
Key length can be 0 or even empty, which is treated as 0. A zero-length key is equivalent to not specifying a key in the other POST body formats: there's no order guarantee.
Message length can be 0 as well, but this isn't very useful.
The examples use plain text but JSON objects are normal.

Finally, this Content-Type may be chunked.
</td></tr>
<tr><td>application/cambria-zip</td><td>@deprecated: Identical to application/cambria except that the POST body is GZip compressed. This POST body can also be chunked.</td></tr>
<tr><td>Other valid http Content-Types</td><td>Message Router assumes them as raw content type and will not validate the formats. This needs to be supported by Java streaming standards. Binary messages needs to be base64 encoded before publishing on to the MR endpoints</td></tr>
</table>

DME2 Service endpoints:

Message Router supports DME2 clients. That is , Client application may use DME2Client and DME2 service address to call the MessageRouter service.

Example DME2 service address:

TEST: http://hostname/events?version=XXX&envContext=XXX&partner=XX

PROD: https://hostname/events?version=XXX &envContext=XXX&partner=XX

The values of version/envContext/routerOffer may change based upon the environment.

The specific details for each API are described as below

Publishers

Description: .

Publishes data to Kafka server on the topic mentioned in the URL. Messages will be in the request body

The MessageRouter service has no requirements on what publishers can put onto a topic. The messages are opaque to the service and are treated as raw bytes. In general, passing JSON messages is preferred, but this is due to higher-level features and related systems, not the MessageRouter broker itself. The only constraint placed on messages is their on their size – messages must be under the maximum size, which is currently configured at 1 MB.

Request URL:

POST http(s)://{HOST:PORT}/**events/{topicname}**

Request Parameters

Name	Description	Param Type	Data type	Max Len	Req'd	Format	Valid/Example Values
Topicname	topic name to be posted	Path	String	40	Y	<app namespace>.<topicname>	Org.onap.crm.empdetails
content-type	To specify type of message content(json,text or cambria)	Header	String	20	N		application/json text/plain application/cambria
Username	userid	Header	String		N	Basic Authentication Header	
Password		Header	String		N	Basic Authentication Header	

partition Key		QueryParam	String		N	String value	?partitionKey =123

Note:

Publishers/user should have access on the topics. The user (id) and permissions details needs to be in AAF.

Response Parameters:

Name	Description	Type	Format	Valid/Example Values
httpStatusCode				200, 201 etc.
mrErrorCode	Numeric error code			200, 201 etc.
errorMessage				SUCCESS, or error message.
helpURL	helpurl			
transactionid	transaction-id value			

Response /Error Codes:

Response statusCode	Response statusMessage
200-299	Success
400-499	the client request has a problem
500-599	the DMaaP service has a problem

Error code	HTTPCode	Description	Issue Reason
DMaaP_MR_ERR_3001	413	Request Entity too large	Message size exceeds the batch limit <limit>.Reduce the batch size and try again
DMaaP_MR_ERR_3002	500	Internal Server Error	Unable to publish messages. Please contact administrator
DMaaP_MR_ERR_3003	400	Bad Request	Incorrect Batching format. Please correct the batching format and try again
DMaaP_MR_ERR_3004	413	Request Entity too large	Message size exceeds the message size limit <limit>.Reduce the message size and try again
DMaaP_MR_ERR_3005	400	Bad Request	Incorrect JSON object. Please correct the JSON format and try again
DMaaP_MR_ERR_3006	504	Network Connect Timeout Error	Connection to the DMaaP MR was timed out.Please try again
DMaaP_MR_ERR_3007	500	Internal Server Error	Failed to publish messages to topic <topicName>. Successfully published <count > number of messages.
	503	Service Unavailable	Service Unavailable

Sample Request:

POST http://<hostname>/events/org.onap.dmaap.mr.sprint

Payload- {"message":"message description"} Content-Type: application/json

Example:

curl -u XXXX@abc.com:X -H 'Content-Type:text/plain' -X POST -d @sampleMsg.txt http://hostname /events/org.onap.ecomp_test.crm.preDemo

{ "count": 1,

"serverTimeMs": 19"

```
}
```

Sample Response:

```
HTTP/1.1 200 OK
   Server: Apache-Coyote/1.1
   transactionId: 28-12-2015::08:18:50:682::<IP>::28122015552391
   Content-Type: application/json
   Content-Length: 42
   Date: Mon, 28 Dec 2015 13:18:50 GMT
```

Subscribers:

Description: To subscribe to a MessageRouter topic, a subscriber issues a GET to the RESTful HTTP endpoint for events.

Request URL:

GET http(s)://{HOST:PORT}}/**events/{topicname}/{consumegroup}/{consumerid}?{timeout=x}**

Request Parameters:

Name	Description	Param Type	Data type	MaxLen	Req'd	Format	Valid/Example Values
Topicname	topic name to be posted	Path	String	40	Y	namespace.string	
Consumergroup	A name that uniquely identifies your subscriber's group	Path	String		Y		CG1
consumerId	Within your subscriber's group, a name that uniquely identifies your subscriber's process	Path	String		Y		C1

content-type	To specify type of message content(json,text or cambria)	Header	String	20	N		application/json text/plain application/cambria
Username	userid	Header	String	1	N	Basic Authentication Header	
Password		Header	String	1	N	Basic Authentication Header	
Timeout	The number of milliseconds to wait for messages if none are immediately available. This should normally be used, and set at 15000 or higher. This is referred to as long-polling timeout Because MR API is built over HTTP, subscribers collect messages by polling the API servers. Having a subscriber poll the API server repeatedly is fine when a topic is busy, but doesn't work well on either end of the transaction when the topic's content is sparse. To help with this, DMaaP API supports a "long poll" mechanism. A subscriber can (and should) add a timeout argument to the GET path and provide a value in milliseconds. If there are no messages waiting for the consumer when the GET call is	Query Param	String	NA	N	?timeout=t value	<apiurl> ?timeout =15000

	received, the server will keep the response open for up to the timeout value before closing it as empty. If traffic arrives during the long poll wait period, it is immediately sent to the subscriber and the response is closed. As a result, the subscriber gets messages promptly without swamping the API server with frequent requests. To prevent subscribers from overwhelming an API server, the server will rate limit empty replies. A subscriber who polls frequently on a busy topic and receives messages on most transactions will not be rate-limited. On the other hand, a subscriber that receives empty replies too frequently may receive 429 responses some of the time. These responses are delayed in order to slow the subscriber down. To prevent 429 replies, subscribers should set the GET timeout to 15000 ms or higher. It does not impact latency						
Limit	The maximum number of messages to return.	Query Param	String	NA	N	?limit=value	<apiurl>?limit=10
Filter	A server-side filter built from the standard Highland Park filter library. Refer to Filter section below	Query Param	String	NA	N	?filter=msg	<apiurl>?timeout=15000&filter=%7B%22foo%22%3D%22bar%22%7D

Note1:

Subscribers /user should have access on the topics. The user () and permissions details needs to be in AAF.

Note 2:

A few consumer client app team who does continuous polling (by multiple rest calls) complained that they receive "503 consumer lock exception". The messages will not be lost from the topic when this exception occurs and it will be still in the topic. This happens when the concurrent rest call requests are made between several nodes in a cluster. Consumer lock is done at zookeeper level to ensure the offset for each consumer group is maintained to avoid losing the message in multiple concurrent calls of same consumer Group. We are looking otherways to resolve this issue. But for these continuous polling scenario, the suggested workaround is to ensure the request is made to only one node of the cluster. Session stickiness in DME2 is example of handling this. If this exception occurs , waiting for few minutes with out making api calls will release the lock in zookeeper.

Response Parameters:

Name	Description	Type	Format	Valid/Example Values
httpStatusCode				200, 201 etc.
mrErrorCode	Numeric error code			200, 201 etc.
errorMessage				SUCCESS, or error message.
helpURL	helpurl			
tranactionid	transaction-id value			28-12-2015::08:18:50:682::135.25.227.66::28122015552391
ResponseBody	Messages consumed from topic	Json	Json	

Response statusCode	Response statusMessage

200-299	Success
400-499	the client request has a problem
500-599	the DMaaP service has a problem

Error code	HTTP Code	Description	Issue Reason
DMaaP_MR_ERR_3008	413	Request Entity too large	Message size exceeds the batch limit <limit>.Reduce the batch size and try again
DMaaP_MR_ERR_3009	500	Internal Server Error	Unable to publish messages. Please contact administrator
DMaaP_MR_ERR_3010	400	Bad Request	Incorrect Batching format. Please correct the batching format and try again
DMaaP_MR_ERR_3011	413	Request Entity too large	Message size exceeds the message size limit <limit>.Reduce the message size and try again
DMaaP_MR_ERR_5012	429	Too many requests	This client is making too many requests. Please use a long poll setting to decrease the number of requests that result in empty responses.
	503	Service Unavailable	Service Unavailable

Sample Request:

GET http://<hostname>/events/org.onap.dmaap.mr.sprint/mygroup/mycus

Content-Type: application/json

Example:

curl -u XXXx@abc.com:map2016$ -X GET -d 'MyfirstMessage' http://hostname**/events/org.onap.ecomp_test.crm.preDeo/myG/C1**

[I am xxx sending first msg,I am XXX sending first msg]

Provisioning:

Description: To create , modify or delete the MessageRouter topics. Generally Invenio application will use these below apis to create , assign topics to the users. These APIs can also be used by other applications to provision topics in MessageRouter

Create Topic

Request URL:

POST http(s)://{HOST:PORT}**/topics/create**

Name	Description	Param Type	Data type	Max Len	Req'd	Format	Valid/Example Values
topicName	topicname to be created in MR	Body	String	20	Y	Json	Org.onap.dmaap.mr.metrics
topicDescription	description for topic	Body	String	15	Y		
partitionCount	Kafka topic partition	Body	String	1	Y		
replicationCount	Kafkatopic replication	Body	String	1	Y		3 (Default - for 3 node Kafka broker cluster)
transactionEnabled	to create transaction id for each	Body	Boolean	1	N		true

	message transaction						
Content-Type	application/json	Header	String		Y		application/json

Response/Error codes

Response statusCode	Response statusMessage
200-299	Success
400-499	the client request has a problem
500-599	the DMaaP service has a problem

Error code	HTTP Code	Description
DMaaP_MR_ERR_5001	500	Failed to retrieve list of all topics
DMaaP_MR_ERR_5002	500	Failed to retrieve details of topic:<topicName>
DMaaP_MR_ERR_5003	500	Failed to create topic:<topicName>
DMaaP_MR_ERR_5004	500	Failed to delete topic:<topicName>

Response Parameters

Name	Description	Type	Format	Valid/Example Values
httpStatusCode				200, 201 etc.
mrErrorCode	Numeric error code			5005
errorMessage				SUCCESS, or error message.

helpURL	helpurl			
ResponseBody	Topic details (owner,trxEnabled=true)	Json	Json	

Sample Request:

POST http://<hostname>/topic/create

Request Body

{"topicName":"org.onap.dmaap.mr.topicname","topicDescription":"This is a SAPTopic ","partitionCount":"1","replicationCount":"3","transactionEnabled":"true"}

Content-Type: application/json

Example:

curl -u XXXx@abc.com:xxxxx$ -H 'Content-Type:application/json' -X POST -d @topicname.txt http://hostname/topics/create

```
{
    "writerAcl": {
        "enabled": false,
        "users": []
    },
    "description": "This is a TestTopic",
    "name": "org.onap.ecomp_test.crm.Load9",
    "readerAcl": {
        "enabled": false,
        "users": []
```

GetTopic Details

Request URL:

GET http(s)://{HOST:PORT}/**topics** **:** To list the details of all the topics in Message Router. (UEB / Cambria format)

GET http(s)://{HOST:PORT}/**topics/{topicname}** **:** To list the details of specified topic .

GET http(s)://{HOST:PORT}/**topics/listAll** **:** To list the details of all the topics (name and owner of each topic)

Request Parameters

Name	Description	Param Type	Data Type	Max Len	Req'd	Format	Valid/Example Values
topicName	topicname details	Body	String	20	Y	Json	Org.onap.dmaap.mr.metrics

Response Parameters:

Name	Description	Param Type	Data type	Format	Valid/Example Values
topicName	topicname details	Body	String	Json	Org.onap.dmaap.mr.metrics
description			String		
Owner	user id who created the topic				
txenabled	true or false		Boolean		

Response/Error Code

Response statusCode	Response statusMessage
200-299	Success
400-499	the client request has a problem
500-599	the DMaaP service has a problem

Error code	Description	HTTPCode
DMaaP_MR_ERR_5001	Failed to retrieve list of all topics	500

DMaaP_MR_ERR_5002	Failed to retrieve details of topic:<topicName>	500

Sample Request:

GET http://<hostname>/topic/org.onap.dmaap.mr.testtopic

Example

curl -u XXXX@abc.com:x$ -X GET http://localhost/topics


```
{"topics": [
   {
      "txenabled": true,
      "description": "This is a TestTopic",
      "owner": "XXXX@abc.com",
      "topicName": "org.onap.ecomp_test.crm.Load9"
   },
   {
      "txenabled": false,
      "description": "",
      "owner": "",
      "topicName": "org.onap.ecomp_test.crm.Load1"
   },
   {
```


Service Specifications

maximum message size	1 MB
durability: time	up to 7 days
durability: storage	up to 70 GB
default topic replica count	3

default topic partition count	8
expected ingest rate in cluster	at least 100,000 1K msgs/sec
concurrent transactions in cluster	thousands (configurable at API server; can scale-out)

API Inventory

	API Name	**API Method**	**REST API Path**		**Comments**
Topics	Get All Topics List	getTopics()	/topics	GET	
	Get All Topics List with details	getAllTopics()	/topics/listAll	GET	
	Get individual Topic Details	getTopic(String topicName)	/topics/{topicName}	GET	
	Create Topic	createTopic(TopicBean topicBean)	/topics/create	POST	
	Delete Topic	deleteTopicString topicName)	/topics/{topicName}	DELETE	Not used in current MR version
	Get Publishers for a Topic	getPublishersByTopicName(String topicName)	/topics/{topicName}/producers	GET	UEB Backward

	Add a Publisher to write ACL on a Topic	permitPublisherForTopic(String topicName, String producerId)	/topics/{topicName}/producers/{producerId}	PUT	Compatibility
	Remove a Publisher from write ACL on a Topic	denyPublisherForTopic(String topicName,String producerId)	/topics/{topicName}/producers/{producerId}	DELETE	
	Get Consumers for a Topic	getConsumersByTopicName (String topicName)	/topics/{topicName}/consumers	GET	
	Add a Consumer to read ACL on a Topic	permitConsumerForTopic(String topicName,String consumerId)	/topics/{topicName}/consumers/{consumerId}	PUT	
	Remove a Consumer from read ACL on a Topic	denyConsumerForTopic(String topicName,String consumerId)	/topics/{topicName}/consumers/{consumerId}	DELETE	
API Keys	Get All API Keys List	getAllApiKeys()	/apiKeys	GET	UEB Backward Compatibility
	Get individual API Key Details	getApiKey(String apiKeyName)	/apiKeys /{apiKey}	GET	
	Create API Key	createApiKey(ApiKeyBeansaApiKey)	/apiKeys /create	POST	
	Update API Key	updateApiKey (String apiKeyName)	/apiKeys /{apiKey}	PUT	
	Delete API Key	deleteApiKey(String apiKeyName)	/apiKeys /{apiKey}	DELETE	

Events (Publish and Subscribe)	Consume Messages	getEvents(String topic, String consumergroup,String consumerid)	/events/{topic}/{consumergroup}/{consumerid}	GET	
	Publish Messages	pushEvents(@PathParam("topic") String topic, InputStream msg,@QueryParam("partitionKey") String partitionKey)	/events /{topic}	POST	
	Publish Messages with a transaction id header	pushEventsWithTransaction(@PathParam("topic") String topic, @QueryParam("partitionKey") String partitionKey)	/events /transaction/{topic}	POST	
Metrics	Get All Metrics	getMetrics()	/metrics	GET	UEB Backward Compatibility
	Get Metrics by name	getMetricsByName(@PathParam("metricName") String metricName)	/metrics/{metricName}	GET	
Admin	Get a list of all the registered consumers	getConsumerCache()	/admin /consumerCache	GET	
	Clears Consumer cache	dropConsumerCache()	/admin /dropConsumerCache	POST	
	Get List of blacklisted IPs	getBlacklist()	/admin/ blacklist	GET	
	Add IP to the list of blacklisted IPs	addToBlacklist (@PathParam("ip") String ip)	/admin/blacklist/{ip}	POST	
	Remove IP from blacklisted IPs	removeFromBlacklist(@PathParam("ip") String ip)	/admin /blacklist/{ip}	DELETE	

Transactions	Get a list of all the existing Transaction Ids	getAllTransactionObjs()	/transaction	GET	
	Get details of a particular transaction id	getTransactionObj(@PathParam("transactionId") String transactionId)	/transaction/{transactionId}	GET	