Error Handling

When errors/exceptions occur in the system, the API Manager throws JSON-based error responses to the client by default.

To change the format of these error responses, you change the relevant XML file in the <API-M_HOME>/repository/deployment/server/synapse-configs/default/sequences directory. The directory includes multiple XML files, named after the type of errors that occur. You must select the correct file.

WSO2 API Manager has the following default fault sequences located in <API-M_HOME> /repository/deployment/server/synapse-configs/default/sequences directory.

Fault Sequence	Description
fault.xml	This is the primary fault sequence that gets invoked when an error occurs during the execution of an API resources
main.xml	This sequence is called when the endpoint being called does not exist
_auth_failure_handler_.xml	This sequence is called when an API authentication error is encountered
_production_key_error_.xml	This sequence is called when a Production key is used to invoke an API that does not have a Production endpoint defined
_sandbox_key_error_.xml	This sequence is called when a Sandbox key is used to invoke an API that does not have a Sandbox endpoint defined
_throttle_out_handler_.xml	This sequence is called when a given request to an API gets throttled out
_token_fault_.xml	This sequence is called when there is an error in invoking the token API
_resource_mismatch_handler_.xml	This sequence is called when a matching resource cannot be found by the gateway to the corresponding resource being invoked
_cors_request_handler_.xml	This sequence enables sending CORS specific headers when the CORS specific configuration (CORSConfiguration) is enabled in WSO2 API Manager in the <API-M_HOME>/repository/conf/deployment.toml file.
_threat_fault_.xml	This sequence is called to send error messages with regard to threat detection.
dispatchSeq.xml	This sequence is defined as a default handler for any inbound WebSocket calls.
outDispatchSeq.xml	This sequence is defined to handle any outbound WebSocket calls.

Error Codes

Given below are some error codes and their meanings.

API handlers error codes

Error code	Error Message	Description	Example
700700	API blocked	This API has been blocked temporarily. Please try again later or contact the system administrators.	Invoke an API which is in the **BLOCKED** lifecycle state
900422	Invalid GraphQL query	Syntax of the provided GraphQL query is invalid	Invoking a GraphQL API which has a invalid query
900424	Temporary Server Error	Server has encountered an error. Please try again.	Invoking a protected backend with an invalid Oauth 2.0 backend token.
900800	Message throttled out	The maximum number of requests that can be made to the API within a designated time period is reached and the API is throttled for the user.	Invoke an API exceeding the tier limit
900801	Hard limit exceeded	Hard throttle limit has been reached	Invoke an API exceeding the hard throttle limit
900802	Resource level throttle out	Message is throttled out because resource level has exceeded	Sending/Receiving messages beyond authorized resource level
900803	Application level throttle out	Message is throttled out because application level is exceeded	Sending/Receiving messages beyond authorized application level
900804	Subscription level throttled out	Message throttled out due to subscription level throttling limit reached.	Sending/Receiving messages beyond configured throttling limit of subscription level policy.
900805	Message blocked	Accessing an API which is blocked on user, IP, application, or API Context.	An admin user can block API invocations in real time by user, IP, application, or API context. The API invocation meets the blocked condition.
900806	Custom policy throttled out	Message throttled out due to exceeding the limit configured through the custom throttling policy rules.	The API invocations meet custom throttle policy rules, exceeding the limits of the configured custom policy.
900807	Message throttled out	Messaged throttled out because of exceeding the burst control/rate limit (requests per second) in the subscription level policy.	Sending/Receiving messages exceeding the configured burst control/rate limit within second.

Error code	Error Message	Description	Example
900900	Unclassified authentication failure	An unspecified error has occurred	Backend service for key validation is not accessible when trying to invoke an API
900901	Invalid credentials	Invalid authentication information provided. **Note** The error code 900904 (Access token inactive) is deprecated from API Manager 1.9.0 onwards. Alternatively, error code 900901 will be sent when the token is inactive.	When the access token is invalid, inactive or expired.
900902	Missing credentials	No authentication information provided	Accessing an API without **Authorization: Bearer** header
900905	Incorrect access token type is provided	The access token type used is not supported when invoking the API. The supported access token types are application and user accesses tokens. See Access Tokens.	Invoke an API with application token, where the resource only allows application user tokens
900906	No matching resource found in the API for the given request	A resource with the name in the request can not be found in the API.	Invoke an API resource that is not available
900907	The requested API is temporarily blocked	Happens when the API user is blocked.	Invoke API resource with a subscription that has been blocked by the API publisher
900908	Resource forbidden	The user invoking the API has not been granted access to the required resource.	Invoke an unsubscribed API
900909	The subscription to the API is inactive	The status of the API has changed to an inaccessible/unavailable state.	Invoke an API resource with a subscription that has not yet been approved by the administrator.
900910	The access token does not allow you to access the requested resource	Can not access the required resource with the provided access token. Check the valid resources that can be accessed with this token.	Invoke API resource with an access token that is not generated to be used with the resource's scope.
102511	Incomplete payload	The payload sent with the request is too large and the client is unable	Sending a large PDF file with the POST request

Error code	Error Message	Description	Example
		to keep the connection alive until the payload is completely transferred to the API Gateway	

Sequences error codes

Error code	Description
900901	Production/sandbox key offered to the API with no production/sandbox endpoint
400	Server cannot process the request due to an error in the request sent by the client
403	No matching resource found in the API for the given request

In addition to the above error codes, we have engaged Synapse-level error codes to the default fault sequence and custom fault sequences (e.g., _token_fault_.xml) of the API Manager. For information, see Error Handling in WSO2 Enterprise Integrator (WSO2 EI) documentation.

Info

The HTTP Status Codes and the corresponding error codes from the error responses are given below.

HTTP Status Code	Error Code
400	102511
401	900901, 900902, 900905, 900907, 900909, 900911
403	900906, 900908, 900910
422	900422
429	900800, 900802, 900803, 900804, 900805, 900806, 900807
500	900900
503	700700, 900801

Transport error codes

Error Code	**Detail**
101000	Receiver input/output error sending
101001	Receiver input/output error receiving
101500	Sender input/output error sending
101501	Sender input/output error receiving
101503	Connection failed

Error Code	**Detail**
101504	Connection timed out (no input was detected on this connection over the maximum period of inactivity)
101505	Connection closed
101506	NHTTP protocol violation
101507	Connection canceled
101508	Request to establish new connection timed out
101509	Send abort
101510	Response processing failed

If the HTTP PassThrough transport is used, and a connection-level error occurs, the error code is calculated using the following equation:

Error code = Base error code + Protocol State

There is a state machine in the transport sender side, where the protocol state changes according to the phase of the message.

Following are the possible protocol states and the description for each:

Protocol State	Description
REQUEST_READY (0)	Connection is at the initial stage ready to send a request
REQUEST_HEAD(1)	Sending the request headers through the connection
REQUEST_BODY(2)	Sending the request body
REQUEST_DONE(3)	Request is completely sent
RESPONSE_HEAD(4)	The connection is reading the response headers
RESPONSE_BODY(5)	The connection is reading the response body
RESPONSE_DONE(6)	The response is completed
CLOSING(7)	The connection is closing
CLOSED(8)	The connection is closed

Since there are several possible protocol states in which a request can time out, you can calculate the error code accordingly using the values in the table above. For example, in a scenario where you send a request and the request is completely sent to the backend, but a timeout happens before the response headers are received, the error code is calculated as follows:

In this scenario, the base error code is CONNECTION_TIMEOUT(101504) and the protocol state is REQUEST_DONE(3).

Therefore,

Error code = 101504 + 3 = 101507

These Transport error codes are used in [Advanced Configurations of Endpoints](#) .

Custom error messages

To send a custom message with a custom HTTP status code, you execute an additional sequence that can generate a new error message. You then override the message body, HTTP status code and other values.

The following steps demonstrate how to override a throttled-out message's HTTP status code as a custom error message:

1. Start the WSO2 API Manager.
2. Go to <API-M_HOME>/repository/deployment/server/synapse-configs/default/sequences directory and create the file convert.xml as follows.

```
<sequence xmlns="http://ws.apache.org/ns/synapse" name="convert">
   <payloadFactory media-type="xml">
      <format>
         <am:fault xmlns:am="http://wso2.org/apimanager">
            <am:code>$1</am:code>
            <am:type>Status report</am:type>
            <am:message>Runtime Error</am:message>
            <am:description>$2</am:description>
         </am:fault>
      </format>
      <args>
         <arg evaluator="xml" expression="$ctx:ERROR_CODE"/>
         <arg evaluator="xml" expression="$ctx:ERROR_MESSAGE"/>
      </args>
   </payloadFactory>
   <property name="RESPONSE" value="true"/>
   <header name="To" action="remove"/>
   <property name="HTTP_SC" value="555" scope="axis2"/>
  <property name="NO_ENTITY_BODY" scope="axis2" action="remove"/>
  <property name="ContentType" scope="axis2" action="remove"/>
  <property name="Authorization" scope="transport" action="remove"/>
  <property name="Access-Control-Allow-Origin" value="*" scope="transport"/>
  <property name="Host" scope="transport" action="remove"/>
  <property name="Accept" scope="transport" action="remove"/>
  <property name="X-JWT-Assertion" scope="transport" action="remove"/>
  <property name="messageType" value="application/json" scope="axis2"/>
  <send/>
</sequence>
```

   ### Tip

   Alternatively, you can use the **Source View** of the API-M Management Console as follows to edit the synapse configuration:

   a. Sign in to the Management Console.(https://<Server Host>:9443/carbon).
   b. Go to **Manager** and then **Source View**.
   c. Copy the content of the sequence in convert.xml, paste it as a new sequence in the source view and update it.

3. Check the terminal logs to see whether there are issues in the deployment.
   If the deployment is successful, you see a message similar to the following in the system logs:

```
 INFO - DependencyTracker Sequence : convert was added to the Synapse configuration successfully
 INFO - SequenceDeployer Sequence named 'convert' has been deployed from file : <API-M_HOME>/repository/deployment/server/synapse-configs/default/sequences/convert.xml
```

4. Include the sequence that you just deployed in a sequence of your choice. For this example, let's add this custom sequence in the _auth_failure_handler_ sequence.

```
<sequence name="_auth_failure_handler_" xmlns="http://ws.apache.org/ns/synapse">
    ...
    <sequence key="convert"/>
    <drop/>
</sequence>
```

5. Check the terminal and see whether there are any errors with the _auth_failure_handler_ sequence deployment.
   If the deployment is successful, you see a message similar to the following in the system logs:

```
  INFO - DependencyTracker Sequence : _auth_failure_handler_ was added to the Synapse configuration successfully
  INFO - SequenceDeployer Sequence: _auth_failure_handler_ has been updated from the file: <API-M_HOME>/repository/deployment/server/synapse-configs/default/sequences/_auth_failure_handler_.xml
```

6. Invoke the API with the respective criteria in order to trigger the sequence.
   In this example, let's view the menu on the PizzaShack API and invoke the API with an incorrect token.

   Format

```
curl -v -H "Authorization: Bearer <Access_Token>" http://localhost:8280/<API_name>/<version>/<context>
```

   Example

```
curl -k -v -X GET "https://localhost:8243/pizzashack/1.0.0/menu" -H "accept: application/json" -H "Authorization: Bearer fb119e84-9542-3194-93dc-1ddddaaa1111"
```

   Sample Response

```
> GET /pizzashack/1.0.0/menu HTTP/1.1

> Host: localhost:8243

> User-Agent: curl/7.54.0

> accept: application/json

> Authorization: Bearer fb119e84-9542-3194-93dc-1ddddaaa1111

>

< HTTP/1.1 555

< Access-Control-Allow-Origin: *

< Access-Control-Allow-Methods: GET

< Access-Control-Allow-Headers: authorization,Access-Control-Allow-Origin,Content-Type,SOAPAction

< Content-Type: application/json; charset=UTF-8

< Date: Fri, 04 Jan 2019 09:53:56 GMT

< Transfer-Encoding: chunked

<

{"fault":{"code":900901,"type":"Status report","message":"Runtime Error","description":"Invalid Credentials"}}
```

Capturing System Data in Error Situations

Carbon Dump is a tool for collecting all the necessary data(i.e., heap and thread dumps) from a running Carbon instance at the time of an error for a head dump and thread stack analysis. The Carbon Dump generates a ZIP archive with the collected data, which helps the WSO2 support team to analyze your system and determine the problem which caused the error. Therefore, it is recommended that you run this tool as soon as an error occurs in the Carbon instance.

As with any other java product, if your WSO2 product cluster fails due to a resource exhaustion, the heap and thread dumps will always point you towards the cause of the leak. Therefore, it is important to be able to retrieve heap and thread dumps from an environment at the point when an error occurs. This will avoid the necessity of reproducing the exact issue again (specially, in the case of production issues). A resource exhaustion can happen for two reasons:

- Due to a bug in the system.
- An actual limitation of resources based on low configuration values.

You can easily create a heap dump and thread dump using the CarbonDump tool that is shipped with your product. These will also provide information about the product version and any patch inconsistencies.

Note

If you are using an Ubuntu version 10.10 or above and if you get an error on being unable to attach the process, execute the following command to rectify it: $ echo 0 | sudo tee /proc/sys/kernel/ yama /ptrace_scope

This changes the yama /ptrace_scope variable of the kernel temporarily (i.e., until the next reboot). For more information, go to Oracle documentation .

When using the tool, you have to provide the process ID (pid) of the Carbon instance and the <PRODUCT_HOME> which is where your unzipped Carbon distribution files reside. The command takes the following format:

```
sh carbondump.sh [-carbonHome path] [-pid of the carbon instance]
```

For example,

```
In Linux: sh carbondump.sh -carbonHome /home/user/wso2carbon-4.0.0/ -pid 5151
In Windows: carbondump.bat -carbonHome c:\wso2carbon-4.0.0\ -pid 5151
```

The tool captures the following information about the system:

- Operating system information ** OS (kernel) version
  - Installed modules lists and their information
  - List of running tasks in the system
- Memory information of the Java process ** Java heap memory dump

    - Histogram of the heap
    - Objects waiting for finalization
    - Java heap summary. GC algorithm used, etc.
    - Statistics on permgen space of Java heap
- Information about the running Carbon instance ** Product name and version
    - Carbon framework version (This includes the patched version)
    - <PRODUCT_HOME>, <JAVA_HOME>
    - configuration files
    - log files
    - H2 database files
- Thread dump
- Checksum values of all the files found in the $CARBON_HOME

Troubleshooting in Production Environments

Analyzing a stack trace

When your Java process starts to spin your CPU, you must immediately analyze the issue using the following two commands and obtain the invaluable information required to tackle the issue. This is done based on the process ID (pid).

1.`jstack <pid> > thread-dump.txt`

2.`ps -C java -L -o pcpu,cpu,nice,state,cputime,pid,tid > thread-usage.txt`

Tip

OS X users can alternatively use the command `ps M <PID>` instead.

These commands provide you with the **thread-dump.txt** file and the **thread-usage.txt** file. After obtaining these two files, do the following.

1. Find the thread ID (the one that belongs to the corresponding PID) that takes up the highest CPU usage by examining the **thread-usage.txt** file.

   ```
   %CPU CPU  NI S    TIME   PID   TID
   ..........
    0.0   -   0 S 00:00:00  1519  1602
    0.0   -   0 S 00:00:00  1519  1603
   24.8   -   0 R 00:06:19  1519  1604
    2.4   -   0 S 00:00:37  1519  1605
    0.0   -   0 S 00:00:00  1519  1606
   ..........
   ```

   In this example, the thread ID that takes up the highest CPU usage is 1604.

2. Convert the decimal value (in this case 1604) to hexadecimal. You can use an online converter to do this. The hexadecimal value for 1604 is 644.
3. Search the **thread-dump.txt** file for the hexadecimal obtained in order to identify the thread that spins. In this case, the hexadecimal value to search for is 644. The **thread-dump.txt** file should have that value as a thread ID of one thread.
4. That thread usually has a stack trace, and that's the lead you need to find the issue. In this example, the stack trace of the thread that spins is as follows.

```
"HTTPS-Sender I/O dispatcher-1" prio=10 tid=0x00007fb54c010000 nid=0x644 runnable [0x00007fb534e20000]
   java.lang.Thread.State: RUNNABLE
        at org.apache.http.impl.nio.reactor.IOSessionImpl.getEventMask(IOSessionImpl.java:139)
        - locked <0x00000006cd91fef8> (a org.apache.http.impl.nio.reactor.IOSessionImpl)
        at org.apache.http.nio.reactor.ssl.SSLIOSession.updateEventMask(SSLIOSession.java:300)
        at org.apache.http.nio.reactor.ssl.SSLIOSession.inboundTransport(SSLIOSession.java:402)
        - locked <0x00000006cd471df8> (a org.apache.http.nio.reactor.ssl.SSLIOSession)
        at org.apache.http.impl.nio.reactor.AbstractIODispatch.inputReady(AbstractIODispatch.java:121)
        at org.apache.http.impl.nio.reactor.BaseIOReactor.readable(BaseIOReactor.java:160)
        at org.apache.http.impl.nio.reactor.AbstractIOReactor.processEvent(AbstractIOReactor.java:342)
        at org.apache.http.impl.nio.reactor.AbstractIOReactor.processEvents(AbstractIOReactor.java:320)
        at org.apache.http.impl.nio.reactor.AbstractIOReactor.execute(AbstractIOReactor.java:280)
        at org.apache.http.impl.nio.reactor.BaseIOReactor.execute(BaseIOReactor.java:106)
        at
org.apache.http.impl.nio.reactor.AbstractMultiworkerIOReactor$Worker.run(AbstractMultiworkerIOReactor.java:604)
        at java.lang.Thread.run(Thread.java:722)
```

Capturing the state of the system

Carbondump is a tool used to collect all the necessary data from a running WSO2 product instance at the time of an error. The carbondump generates a ZIP archive with the collected data that helps to analyze the system and determine the problem that caused the error. Therefore, it is recommended that you run this tool as soon as an error occurs in the WSO2 product instance.

When using the tool, you have to provide the process ID (pid) of the product instance and the <PRODUCT_HOME> location, which is where your unzipped Carbon distribution files reside. The command takes the following format:

```
sh carbondump.sh [-carbonHome path] [-pid of the carbon instance]
```

For example,

```
In Linux: sh carbondump.sh -carbonHome /home/user/wso2carbon-4.0.0/ -pid 5151
In Windows: carbondump.bat -carbonHome c:\wso2carbon-4.0.0\ -pid 5151
```

The tool captures the following information about the system:

- Operating system information ** OS (kernel) version
    - Installed modules lists and their information
    - List of running tasks in the system
- Memory information of the Java process ** Java heap memory dump
    - Histogram of the heap
    - Objects waiting for finalization
    - Java heap summary. GC algorithm used, etc.

    - Statistics on permgen space of Java heap
- Information about the running Carbon instance ** Product name and version
    - Carbon framework version (This includes the patched version)
    - <PRODUCT_HOME>, <JAVA_HOME>
    - configuration files
    - log files
    - H2 database files
- Thread dump
- Checksum values of all the files found in the $CARBON_HOME

Viewing process threads in Solaris

This information is useful to know in situations when the database processes are not fully utilizing the CPU's threading capabilities. It gives you a better understanding on how 11g and 10g takes advantage of threading and how you can validate those queries from the system.

The following information provides insight on whether a Solaris process is parallelized and is taking advantage of the threading within the CPU.

1. Open a command line in Solaris.
2. Run `prstat` and have a look to the last column, labeled `PROCESS/NLWP` . NLWP is a reference to the number of lightweight processes and are the number of threads the process is currently using with Solaris as there is a one-to-one mapping between lightweight processes and user threads. A single thread process will show `1` there while a multi-threaded one will show a larger number. See the following code block for an example.

   ```
      PID USERNAME  SIZE   RSS STATE  PRI NICE     TIME  CPU PROCESS/NLWP
    ...
    12905 root     4472K 3640K cpu0    59    0   0:00:01 0.4% prstat/1
    18403 monitor   474M  245M run     59   17   1:01:28 9.1% java/103
     4102 oracle     12G   12G run     59    0   0:00:12 4.5% oracle/1
   ```

   If you observe the `PROCESS/NLWP` value in the example above, you can identify that `prstat` and `oracle` are single thread processes, while `java` is a multi-threaded process.
3. Alternatively, you can analyze individual thread activity of a multi-threaded process by using the `-L` and `-p` options, like `prstat -L -p pid` . This displays a line for each thread sorted by CPU activity. In that case, the last column is labeled `PROCESS/LWPID` , where LWPID is the thread ID. If more than one thread shows significant activity, your process is actively taking advantage of multi-threading.

Checking the health of a cluster

In Hazelcast, the health of a member in the cluster is determined by the heartbeats the member sends. If the well-known member does not receive a heartbeat within a given amount of time (this can be configured), then the node is assumed dead. By default, the given amount of time is 600 seconds (or 10 minutes), which might be too much for some scenarios.

Failure detectors used in distributed systems can be unreliable. In these sort of scenarios, Hazelcast uses heartbeat monitoring as a fault detection mechanism and the nodes send heartbeats to other nodes.

If a heartbeat message is not received by a given amount of time, Hazelcast assumes the node is dead. This is configured via the hazelcast.max.no.heartbeat.seconds property. The optimum value for this property depends on the system. Although the default is 600 seconds, it might be necessary to reduce the heartbeat to a lower value if nodes are to be declared dead in a shorter time frame. However, you must verify this in your system and adjust as necessary depending on your scenario.

Warning

Reducing the value of this property to a lower value can result in nodes being considered as dead even if they are not. This results in multiple messages indicating that a node is leaving and rejoining the cluster.

Do the following steps to configure the maximum time between heartbeats.

1. Create a property file called hazelcast.properties, and add the following property to it. hazelcast.max.no.heartbeat.seconds=300
2. Place this file in the <PRODUCT_HOME>/repository/conf/ directory in all the nodes in your cluster.
3. Restart the servers.

Utilizing the WSO2 Runtime Diagnostic Tool

WSO2 Runtime Diagnostic Tool is a lightweight and easy-to-use tool for generating diagnostic details. It simplifies the data collection process to minimize user involvement. The tool is capable of preemptive data collection for certain types of issues, such as OOM errors. It also captures significant changes in Passthrough metrics for better insight into specific issues.

The tool can be configured using the Configuration Guide provided below. The output of the tool can be analyzed using the Analysis Guide provided below.

Components

There are four main components in the tool:

- **Memory Watcher**: Monitors the memory usage of the server and executes the configured Action Executors when the memory usage exceeds the threshold.
- **CPU Watcher**: Monitors the CPU usage of the server and executes the configured Action Executors when the CPU usage exceeds the threshold.
- **Traffic Analyzer**: Monitors the traffic of the server and generates logs when the traffic pattern suddenly changes significantly.
- **Log Watcher**: Monitors the error logs of the server and executes the configured Action Executors when the log pattern matches the configured pattern.

Configuration Guide

The tool is packaged inside the product distribution with default configurations. The configurations can be customized based on user requirements.

Server Configurations

The table given below describes the server configurations.

Configuration	Description
deployment_toml_path	Path to the deployment.toml file in the WSO2 server
logs_directory	Path to the logs directory.
updates_config_path	Path to the updates config file.
diagnostic_log_file_path	Path to write the diagnostic log file.
carbon_log_file_path	Path to the carbon error log (wso2error.log) file.
process_id_path	Path to the process id file (wso2carbon.pid).
server_name	Name of the WSO2 server.
server_version	Version of the WSO2 server.

Given below is a sample configuration for the WSO2 API Manager.

```
[server_configuration]
deployment_toml_path = "../conf/deployment.toml"
logs_directory = "../repository/logs"
updates_config_path = "../updates/config.json"
diagnostic_log_file_path = "logs/diagnostics.log"
carbon_log_file_path = "../repository/logs/wso2error.log"
process_id_path = "../wso2carbon.pid"
server_name = "WSO2 API Manager"
server_version = "#.#.#"
```

Action Executor Configurations

Currently, the tool supports the following action executors.

Action Executor	Description
ThreadDumper	Runs the jstack tool to take thread dump and writes the output to a file.
MemoryDumper	Takes a heap dump
OpenFileFinder	Finds the open files by the server process and writes the output to a file.
Netstat	Dumps the network statistics of the server to a file.
ServerInfo	Dumps the server information such as name, version, etc.
MetricsSnapshot	Takes a current snapshot of the Passthrough transport metrics in synapse

ThreadDumper

Configuration	Description
count	Number of thread dumps to be taken.
delay	Delay between each thread dump in milliseconds.

Given below is a sample configuration for the ThreadDumper action executor.

```
[[action_executor_configuration]]
executor = "ThreadDumper"
count = "5"
delay = "2000"
```

MemoryDumper

Given below is a sample configuration for the MemoryDumper action executor.

```
[[action_executor_configuration]]
executor = "MemoryDumper"
```

OpenFileFinder

Given below is a sample configuration for the OpenFileFinder action executor.

```
[[action_executor_configuration]]
executor = "OpenFileFinder"
```

Netstat

Given below is a sample configuration for the Netstat action executor.

```
[[action_executor_configuration]]
executor = "Netstat"
command = "netstat -lt"
```

ServerInfo

Given below is a sample configuration for the ServerInfo action executor.

```
[[action_executor_configuration]]
executor = "ServerInfo"
```

MetricsSnapshot

Given below is a sample configuration for the MetricsSnapshot action executor.

```
[[action_executor_configuration]]
executor = "MetricsSnapshot"
```

Watcher Configurations

Currently, the tool supports the following watchers.

Watcher	Description
cpu_watcher	Watches the CPU usage of the server.
memory_watcher	Watches the memory usage of the server.
log_watcher	Watches the logs for specific error patterns and triggers actions.
traffic_analyzer	Analyzes the Passthrough server traffic and records in the diagnostic log file.

cpu_watcher

Configuration	Description
Enabled	Whether the watcher is enabled or not.
Threshold	The threshold value for the CPU usage.
Attempts	The number of attempts before triggering the action executors (This resets every hour).
Interval	The interval between each check in seconds.
action_executors	The action executors to be triggered when the threshold is reached. (Comma separated)

Given below is a sample configuration for cpu_watcher.

```
[cpu_watcher]
enabled = "true"
threshold = "20"
attempts = "2"
interval = "5"
action_executors = "ThreadDumper,MetricsSnapshot,ServerInfo"
```

memory_watcher

Configuration	Description
enabled	Whether the watcher is enabled or not.
threshold	The threshold value for the memory usage.
attempts	The number of attempts before triggering the action executors (This resets every hour).
interval	The interval between each check in seconds.
action_executors	The action executors to be triggered when the threshold is reached. (Comma separated)

Given below is a sample configuration for memory_watcher.

```
[memory_watcher]
enabled = "true"
threshold = "30"
attempts = "2"
interval = "5"
action_executors = "ThreadDumper,MetricsSnapshot,ServerInfo"
```

log_watcher

Configuration	Description
Enabled	Whether the watcher is enabled or not.
Interval	The interval between each check in seconds.

Given below is a sample configuration for log_watcher.

```
[log_watcher]
enabled = "true"
interval = "0.1"
```

Log error patterns

Configuration	Description
regex	Directory to store the zip files
executors	The action executors to be triggered when the threshold is reached. (Comma separated)
reload_time	Continuous error logs that match the regex pattern won't be processed repeatedly unless the reload_time has elapsed. An error log which matches a certain regex pattern will only be processed after the reload time interval whereas a similar error log was processed before.

Given below is a sample configuration.

```
[[log_pattern]]
regex = "(.*)org.apache.synapse.transport.passthru(.*)"
executors = "MetricsSnapshot,Netstat,OpenFileFinder,ThreadDumper,ServerInfo"
reload_time = "30"
```

Traffic Analyzer Configurations

Configuration	Description
last_second_requests_enabled	Whether the last second requests watcher is enabled or not.
last_second_requests_windows_size	The window size for the last second requests watcher.
last_second_requests_delay	The delay for the last second requests watcher.
last_second_requests_interval	The interval for the last second requests watcher.
last_fifteen_seconds_requests_enabled	Whether the last fifteen seconds requests watcher is enabled or not.
last_fifteen_seconds_requests_window_size	The window size for the last fifteen seconds requests watcher.
last_fifteen_seconds_requests_delay	The delay for the last fifteen seconds requests watcher.
last_fifteen_seconds_requests_interval	The interval for the last fifteen seconds requests watcher.
last_minutes_requests_enabled	Whether the last minutes requests watcher is enabled or not.
last_minutes_requests_window_size	The window size for the last minutes requests watcher.
last_minutes_requests_delay	The delay for the last minutes requests watcher.
last_minutes_requests_interval	The interval for the last minutes requests watcher.
notify_interval	The interval for the traffic analyzer to notify the user.

Given below is a sample configuration for the traffic analyzer.

```
[traffic_analyzer]
last_second_requests_enabled = "false"
last_second_requests_windows_size = "300"
last_second_requests_delay = "60"
last_second_requests_interval = "1"
last_fifteen_seconds_requests_enabled = "true"
last_fifteen_seconds_requests_window_size = "100"
last_fifteen_seconds_requests_delay = "4"
last_fifteen_seconds_requests_interval = "15"
last_minutes_requests_enabled = "true"
last_minutes_requests_window_size = "100"
last_minutes_requests_delay = "1"
last_minutes_requests_interval = "60"
```

```
notify_interval = "300"
```

Post Action Executors

Zip File Configurations

Configuration	Description
output_directory	Directory to store the zip files
max_count	Maximum number of zip files to maintain. When the count exceeds, the older files will be deleted.

Given below is a sample configuration.

```
[zip_file_configuration]
output_directory = "data"
max_count = "50"
```

FTP Configurations

Configuration	Description
enabled	Whether the FTP is enabled or not.
host	The FTP host.
port	The FTP port.
username	The FTP username.
password	The FTP password.
directory	The FTP directory.

Given below is a sample configuration.

```
[ftp_configuration]
enabled = "true"
host = "ftp.example.com"
port = "21"
username = "user"
password = "password"
directory = "diagnostics"
```

SFTP Configurations

Configuration	Description
enabled	Whether the SFTP is enabled or not.
host	The SFTP host.
port	The SFTP port.

Configuration	Description
username	The SFTP username.
password	The SFTP password.
directory	The SFTP directory.

Given below is a sample configuration.

```
[sftp_configuration]
enabled = "true"
host = "sftp.example.com"
port = "22"
username = "user"
password = "password"
directory = "diagnostics"
```

Log4j2 Configurations

The log4j2.properties file can be used to configure the logging level of the tool. The default log level is set to INFO. The log4j2.properties file can be found in the conf directory.

Analysing Runtime Issues with the WSO2 Runtime Diagnostic Tool

Data

The zip files are generated in the <WSO2_HOME>/diagnostics-tool/data directory. The zip file is named as <processId>-<timestamp>.zip. The zip file contains the following files:

- **deployment.toml**: The deployment.toml file of the server which contains the configurations.
- **diagnostics.log**: The log file of the diagnostics tool which contains logs related to traffic pattern to the server. The logs are explained in the Diagnostics Log section.
- **log.txt**: The log line that triggered the Action Executors.
- **logs.zip**: The log directory of the server in zip format. This may contain the heap dump file if it is generated.
- **lsof-output.txt** [Optional]: The output of the lsof command which contains the open files by the server process during the error time.
- **netstat-output.txt** [Optional]: The output of the netstat command which contains the network statistics of the server during the error time.
- **server-info.txt**: The server information such as name, version etc.
- **thread-dump--.txt**: The thread dump of the server taken during different time intervals during the error time.
- **metrics-snapshot.txt**: The current snapshot of the Passthrough transport metrics in synapse runtime.

Diagnostics Log

The diagnostics log contains the following logs:

Memory Watcher

The memory watcher logs are prefixed with [MemoryWatcher].

For example: MemoryWatcher Heap usage is above threshold. Heap usage: 87, Retry count: " + count.

Here, the log indicates that the heap usage is above the threshold and the heap usage is 87%. The MemoryWatcher retries a couple of times before executing the Action Executors.

CPU Watcher

The CPU watcher logs are prefixed with [CPUWatcher].

For example: CPUWatcher CPU usage is above threshold. CPU usage: 91, Retry count: " + count.

Here, the log indicates that the CPU usage is above the threshold and the CPU usage is 91%. The CPUWatcher retries a couple of times before executing the Action Executors.

Traffic Analyzer

The traffic analyzer logs are prefixed with [TrafficAnalyzer].

For example: TrafficAnalyzer Attribute Last15SecondRequests of type http-listener increased more than the threshold, old value: 2, new value: 227, threshold: 115.22752880979914.

Here, the log indicates that the Last15SecondRequests attribute of type http-listener increased more than the threshold. The old value is 2, the new value is 227 and the threshold is 115.22752880979914. The threshold is calculated based on the standard deviation of a Simple Moving Average window.

Log Watcher

Following are examples of log watcher logs:

```
[Interpreter] Executing the action executors for the log line matching the regex pattern (.*)org.apache.synapse.transport.passthru(.*)
ServerInfo [INFO] ServerInfo executed successfully.
OpenFileFinder [INFO] OpenFileFinder executed successfully.
Netstat [INFO] Netstat executed successfully
ZipFileExecutor [INFO] Zipping the folder at /Users/user/wso2am-4.3.0/diagnostics-tool/temp/2024-03-01_14:21:06.743
ZipFileExecutor [INFO] Diagnosis Dumped in :/Users/user/wso2am-4.3.0/diagnostics-tool/data/96970_2024-03-
01_14:21:06.743.zip
ThreadDumper [INFO] Thread dump execution is completed for 96970, thread dump count: 5, delay: 2000ms
MetricsSnapshot [INFO] MetricsSnapshot executed successfully.
ZipFileExecutor [INFO] Zipping the folder at /Users/user/wso2am-4.3.0/diagnostics-tool/temp/2024-03-01_14:21:06.838
ZipFileExecutor [INFO] Diagnosis Dumped in :/Users/user/wso2am-4.3.0/diagnostics-tool/data/96970_2024-03-
01_14:21:06.838.zip
```

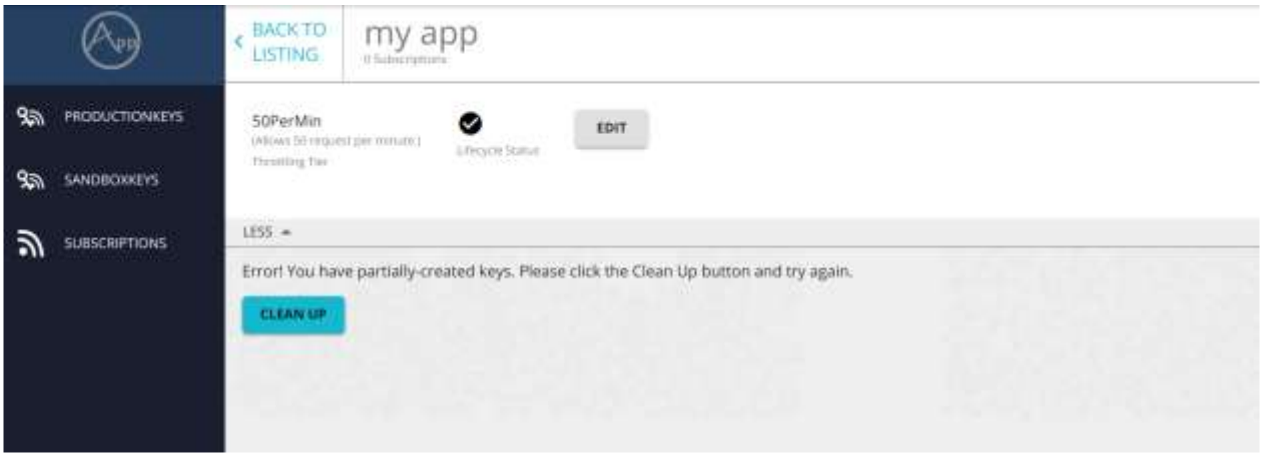
Here, the log indicates that the log line matching the regex pattern (.*)org.apache.synapse.transport.passthru(.*) is found. The ServerInfo, OpenFileFinder, Netstat, MetricsSnapshot, and ThreadDumper Action Executors are executed. The executors are executed in parallel and the zip file is generated with the output at the end.

Cleaning Up Partially Created Keys

An application created in WSO2 API Manager has a corresponding OAuth application in the Key Manager node. An application can be created or deleted partially, where the OAuth application is successfully created/deleted but there is stale data left in the API Manager node. This can happen due to network failures between the API Manager and the Key Manager nodes, partial deletion of applications, etc.

To delete the remaining application data from API Manager, follow below steps.

1. Navigate to view the application listing by clicking on the **Applications** tab in the top ribbon.
2. Click on the application name. This will show the application details.
3. Click on the **Clean up** button found at the bottom of the page.

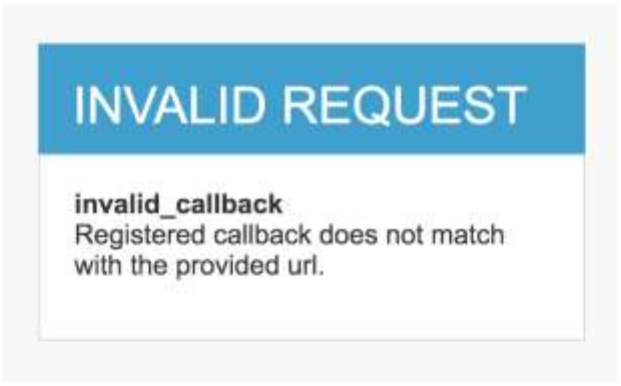


Error! You have partially-created keys. Please click the Clean Up button and try again.

Troubleshooting 'Registered callback does not match with the provided URL' error

The **Registered callback does not match with the provided URL** error can be encountered during the API Publisher(https://<hostname>:9443/publisher) and API Developer Portal (https://<hostname>:9443/devportal) login attempts, in a case where the hostname of the API Manager has been changed after accessing the Developer Portal or Publisher apps via different hostnames.

For example, let's assume that you have started a fresh APIM server and accessed the API Publisher and Developer Portal apps via localhost. If you have changed the hostname of the server from localhost to apim.wso2.com, the next login attempt to API Publisher or Developer Portal will be failed giving this error.

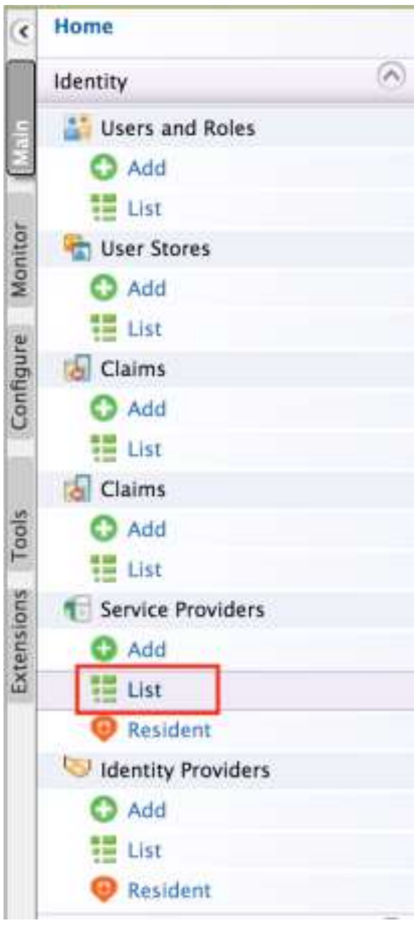

INVALID REQUEST

invalid_callback
Registered callback does not match with the provided url.

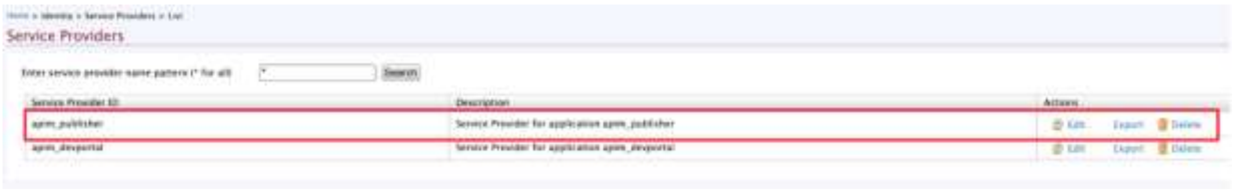
This error has been occurred due to the mismatch of the API Publisher or API Developer Portal access URLs((https://<hostname>:9443/publisher and https://<hostname>:9443/devportal) and callback URLs which are configured in API Publisher and API Developer Portal Service Providers.

Please follow below steps to fix the login failure due to callback URL mismatch.

1. Sign in to the Management Console (https://<hostname>:9443/carbon).
2. Navigate to service providers list.

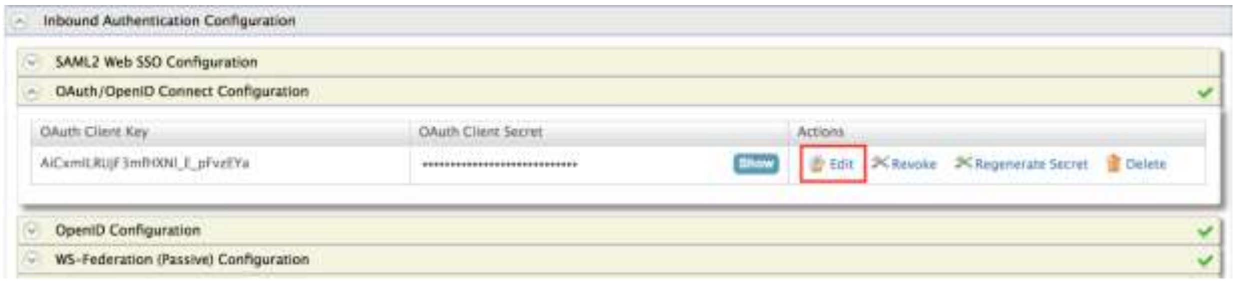


3. Click on the **Edit** button of API Publisher service provider


Home > Identity > Service Providers > List
Service Providers
Enter service provider name pattern (* for all)
Search

Service Provider ID	Description	Actions
apim_publisher	Service Provider for application apim_publisher	Edit Export Delete
apim_devportal	Service Provider for application apim_devportal	Edit Export Delete

4. Navigate to **Inbound Authentication Configuration > OAuth/OpenID Connect Configuration** and click on OAuth application edit button.


Inbound Authentication Configuration
SAML2 Web SSO Configuration
OAuth/OpenID Connect Configuration
OpenID Configuration
WS-Federation (Passive) Configuration
Show
Edit
Revoke
Regenerate Secret
Delete

OAuth Client Key	OAuth Client Secret	Actions
AiCxmlLRUjF3mfHXNI_E_pFvzEYa	******************************	Edit Revoke Regenerate Secret Delete

5. See the **Callback Url** regex value configured under Application Settings. You will observe that the callback URL value is having a different hostname(localhost or previous hostname which was configured before the hostname change).
6. regexp=(https://localhost:9443/publisher/services/auth/callback/login|https://localhost:9443/publisher/services/auth/callback/logout)

   Then replace the callback URL hostname with the current hostname of the server. For example, if the current hostname of the server is apim.wso2.com, the callback URL regex has to be changes as follows.

   regexp=(https://apim.wso2.com:9443/publisher/services/auth/callback/login|https://apim.wso2.com:9443/publisher/services/auth/callback/logout)
7. Click on **Update** button in Applications Settings page, then the **Update** button of service provider information page to save the callback URL change.
8. Select the service provider for API Developer Portal (admin_admin_store) and repeat the step 4 - step 6 to apply the same changes.

Troubleshooting JMS scenarios

The following sections will help you to resolve common problems encountered in JMS integration scenarios with WSO2 Micro Integrator.

Exceptions related to client libraries

You may encounter ClassNotFoundExceptions and NoClassDefFoundExceptions if you have not deployed all the required client libraries. The missing class should be available in one of the jar files deployed in MI_HOME/lib directory. WSO2 Micro Integrator comes with geronimo-jms library, which contains the javax.jms packages. Therefore, you do not have to deploy them again.

HTTP header conversion

When forwarding HTTP traffic to a JMS queue using WSO2 Micro Integrator, you might get an error similar to the one given below.

```
ERROR JMSSender Error creating a JMS message from the axis message context
javax.jms.MessageFormatException: MQJMS1058: Invalid message property name: Content-Type
```

This exception is specific to the JMS broker used, and is thrown by the JMS client libraries used to connect with the JMS broker.

The incoming HTTP message contains a bunch of HTTP headers that have the ‘-‘ character. Some noticeable examples are **Content-length** and **Transfer-encoding** headers. When the Micro Integrator forwards a message over JMS, it sets the headers of the incoming message to the outgoing JMS message as JMS properties. But, according to the JMS specification, the ‘-‘ character is prohibited in JMS property names. Some JMS brokers like ActiveMQ do not check this specifically, in which case there will not be any issues. But some brokers do and they throw exceptions.

The solution is to simply remove the problematic HTTP headers from the message before delivering it over JMS. You can use the property mediator as follows to achieve this:

```
<property action="remove" name="Content-Length" scope="transport">
<property action="remove" name="Accept-Encoding" scope="transport">
<property action="remove" name="User-Agent" scope="transport">
<property action="remove" name="Content-Type" scope="transport">
```

Alternatively, you can use the `transport.jms.MessagePropertyHyphens` parameter to handle hyphenated properties, instead of handling them as described above.

JMS property data type mismatch

When WSO2 Micro Integrator attempts to forward a message over JMS, there are instances that the client libraries throw an exception saying the data type of a particular message property is invalid.

This problem occurs when the developer uses the property mediator to manipulate property values set on the message. Certain implementations of JMS have data type restrictions on properties. But the property mediator always sets property values as strings.

The solution is to revise the mediation sequences and avoid manipulating property values containing non-string values. If you want to set a non-string property value, write a simple custom mediator. Instructions are given in section Creating Custom Mediators. For an example, to set a property named foo with integer value 12345, use the property mediator as follows and set the type attribute to INTEGER. If the type attribute of the property is not specifically set, it will be assigned to String by default.

```
<property name="foo" value="12345" type="INTEGER" scope="transport/">
```

Too-many-threads and out-of-memory issues

With some JMS brokers, WSO2 Micro Integrator tends to spawn new worker threads indefinitely until it runs out of memory and crashes. This problem is caused by a bug in the underlying Axis2 engine. A simple workaround to this problem is to engage the property mediator of the mediation sequence as follows:

```
<property action="remove" name="transportNonBlocking" scope="axis2">
```

This prevents the Micro Integrator from creating new worker threads indefinitely. You can use a jconsole like JMX client to monitor the active threads and memory consumption of WSO2 Micro Integrator.

JMSUtils cannot locate destination

If your topic or queue name contains the termination characters ":" or "=", JMSUtils will not be able to find the topic/queue and will give you the warning "JMSUtils cannot locate destination". For more information, see http://docs.oracle.com/javase/7/docs/api/java/util/Properties.html#load%28java.io.Reader%29. For example, if the topic name is `my::topic`, the following configuration will not work, because the topic name will be parsed as `my` instead of `my::topic`:

```
<address
uri="jms:/my::topic?transport.jms.ConnectionFactoryJNDIName=QueueConnectionFactory&amp;java.naming.factory.initial=org.
wso2.andes.jndi.PropertiesFileInitialContextFactory&amp;java.naming.provider.url=repository/conf/jndi.properties&amp;transport.
jms.DestinationType=topic"/>
```

To avoid this issue, you can create a key-value pair in the `deployment.toml` file that maps the topic/queue name to a key that either escapes these characters with a backslash (\) or does not contain ":" or "=". For example:

```
[transport.jndi.topic]
'my\:\:topic' = "my::topic"
```

or

```
[transport.jndi.topic]
myTopic = "my::topic"
```

You can then use this key in the proxy service as follows:

```
<address
uri="jms:/my\:\:topic?transport.jms.ConnectionFactoryJNDIName=TopicConnectionFactory&amp;java.naming.factory.initial=org.
wso2.andes.jndi.PropertiesFileInitialContextFactory&amp;java.naming.provider.url=repository/conf/jndi.properties&amp;transport.
jms.DestinationType=topic"/>
```

If you do not want to use the JNDI properties file, you can define the key-value pair right in the proxy configuration:

```
<address
uri="jms:/my\:\:topic?transport.jms.ConnectionFactoryJNDIName=TopicConnectionFactory&amp;java.naming.factory.initial=org.
```

```
wso2.andes.jndi.PropertiesFileInitialContextFactory&amp;topic.my\:\:topic=my::topic&amp;java.naming.provider.url=repository/c
onf/jndi.properties&amp;transport.jms.DestinationType=topic"/>
```

All the threads get blocked if one JMS backend is not available and do not recover when the backend is available again

When one backend fails, the following state appears for all the threads as they try to connect to the unavailable backend.

```
- state:BLOCKED
```

Once the backend is available again, the threads do not become active again

This is because in the JMS transport, all the threads that use the same JMS session for communication are synchronized for thread safety. Therefore, if one thread obtains the shared JMS session object and waits to obtain another resource (i.e., a reconnection to IBM MQ), this results in a set of threads waiting on this monitor. This results in all the synapse threads being blocked. When all the threads are blocked in the connection pool, WSO2 MI stops responding to requests.

In order to make sure that WSO2 MI recovers after the backend is fixed, specify a connection timeout by following the steps below.

1. Open the `<MI_HOME>/bin/micro-integrator.sh` file.
2. In the `JAVA_OPTS` section, set the following property.

   ```
   Dcom.ibm.mq.cfg.TCP.Connect_Timeout=5
   ```

   e.g., The following is an extract of the `JAVA_OPTS` section with this property.

   ```
   JAVA_OPTS="-Xdebug -Xnoagent -Djava.compiler=NONE -
   Dcom.ibm.mq.cfg.TCP.Connect_Timeout=5 -
   Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=$PORT"
   ```

   Info

   By default, the value of this parameter is `0` which results in the thread that tries to reconnect to IBM MQ keeps retrying, and remains blocked together with all the other threads that share the same JMS session. Here, you are specifying a timeout period of five seconds to reconnect to IBM MQ. So that all the threads in the thread pool are prevented from being blocked until the connection is successfully established. For more information about this parameter, see IBM Knowledge Centre - TCP stanza of the client configuration file.

3. Restart the WSO2 MI server.

Troubleshooting WebSocket APIs

Troubleshooting WebSocket APIs

If you require more detailed logs in the WebSocket API flow in order to troubleshoot and debug an error in your scenario, you can enable debug logs as indicated below.

Add the following entries in the `<API-M_HOME>/repository/conf/log4j2.properties` file.

```
logger.InboundWebsocketSourceHandler.name =
org.wso2.carbon.inbound.endpoint.protocol.websocket.InboundWebsocketSourceHandler
logger.InboundWebsocketSourceHandler.level = DEBUG

logger.InboundWebsocketResponseSender.name =
org.wso2.carbon.inbound.endpoint.protocol.websocket.InboundWebsocketResponseSender
logger.InboundWebsocketResponseSender.level = DEBUG

logger.WebSocketClientHandler.name = org.wso2.carbon.websocket.transport.WebSocketClientHandler
logger.WebSocketClientHandler.level = DEBUG

logger.WebsocketTransportSender.name = org.wso2.carbon.websocket.transport.WebsocketTransportSender
logger.WebsocketTransportSender.level = DEBUG
```

Then, add the loggers in a comma-separated list as shown here:

```
loggers = InboundWebsocketSourceHandler, InboundWebsocketResponseSender, WebSocketClientHandler,
WebsocketTransportSender
```