Parser Troubleshooting Guide

Exabeam Security Management Platform - Version SMP 2020.3

Publication date November 2, 2020

Exabeam
1051 E. Hillsdale Blvd., 4th Floor
Foster City, CA 94404

650.209.8599

Have feedback on this guide? We'd love to hear from you!
Email us at docs@exabeam.com

Table of Contents

1. Parser Troubleshooting

Exabeam provides out-of-the-box parsers to address the wide variety of log sources. In addition, environments can also have custom parsers to suit their specific needs.

Under certain scenarios, parsers could be inefficient or take too long to parse specific log events or sources. This could in turn impact the overall system performance. In order to keep the system stable and prevent any system wide issues, Exabeam has implemented automatic parser disabling. This can prevent performance degradation by taking the slow parsers out of service.

Slow parsers can impact the volume of log ingestion thereby causing a deteriorating ripple effect within the rest of the system. This can result in the slowing down of the entire data pipeline from ingestion to analytics. Therefore, disabling such parsers is necessary in order to maintain reliable data ingestion.

Exabeam Advanced Analytics and Data Lake both utilize parsers but manage them differently. The following troubleshooting guides will help to detect and fix slow parsers. Please refer to the product you are troubleshooting for more information:

- Data Lake Parser Troubleshooting
- Advanced Analytics Parser Troubleshooting

2. Troubleshoot Data Lake Parsers

Under certain scenarios, parsers could be inefficient or take too long to parse specific log events or sources. Slow parsers can impact the volume of log ingestion thereby causing a deteriorating ripple effect within the rest of the system. This can result in the slowing down of the entire data pipeline from ingestion to analytics. This could in turn impact the overall system performance. In order to keep the system stable and prevent any system wide issues, Exabeam has implemented automatic parser pausing. This can prevent performance degradation by taking the slow parsers out of service.

A slow parser in Data Lake is determined by its execution performance. Each parser is managed individually based on the following:

- Parsing must be completed in 12 ms or less for every log/event.
- All parsers are polled for completion times every five minutes.
- Individual parsers that fail to complete parsing at or under the threshold in three consecutive tries are then automatically paused.

2.1. Parser Configuration

The pausing of parsers is based on two configurable variables found in mojito_default.conf. The custom config file, custom_mojito.conf, that you should edit can be found at the following locations:

- Before Data Lake i24:

  ```
  /opt/exabeam/config/lms/elasticsearch
  ```

- For Data Lake i24 to i31:

  ```
  /opt/exabeam/config/lms/elasticsearch/ingest_mojito
  ```

- For Data Lake i32 and later:

  ```
  /opt/exabeam/config/lms/mojito-kafka-connect/ingest-mojito
  ```

Thresholds are found in the `SlowParsingConfig` block:

```
SlowParsingConfig {
...
AutoDisableEnabled = true
DisableThresholdTime = 12 milliseconds
DisableThresholdPeriods = 3
... }
```

- **`DisableThresholdTime`** — Performance threshold, in milliseconds, for a parser to complete an execution

⚠ **CAUTION**
Exercise care when increasing this threshold from the default 12 ms. The default value is the supported threshold to maintain processing at 4,500 EPS for appliance and 3,000 EPS for cloud on a per host basis. Maximum EPS that a host can handle is expected to decrease as the threshold increases.

- **`DisableThresholdPeriod`** — Number of consecutive performance polls

2.2. Apply Parser Performance Threshold

Slow parsers are paused when a threshold is triggered. Only Exabeam parsers are subject to performance parameters `DisableThresholdTime` and `DisableThresholdPeriods`. Custom parsers are paused based on resource allocation you choose.

For Exabeam parsers, you must restart the parsing service to apply any changes to the configuration, such as `DisableThresholdTime`. You can restart using the following commands.

- For Data Lake i20 to i31.1:

```
/opt/exabeam/bin/lms/apply-parsers-and-restart-ingest-nodes.py
```

- For Data Lake i32 and later:

```
sos
dl-mojito-restart -enable-parsers
```

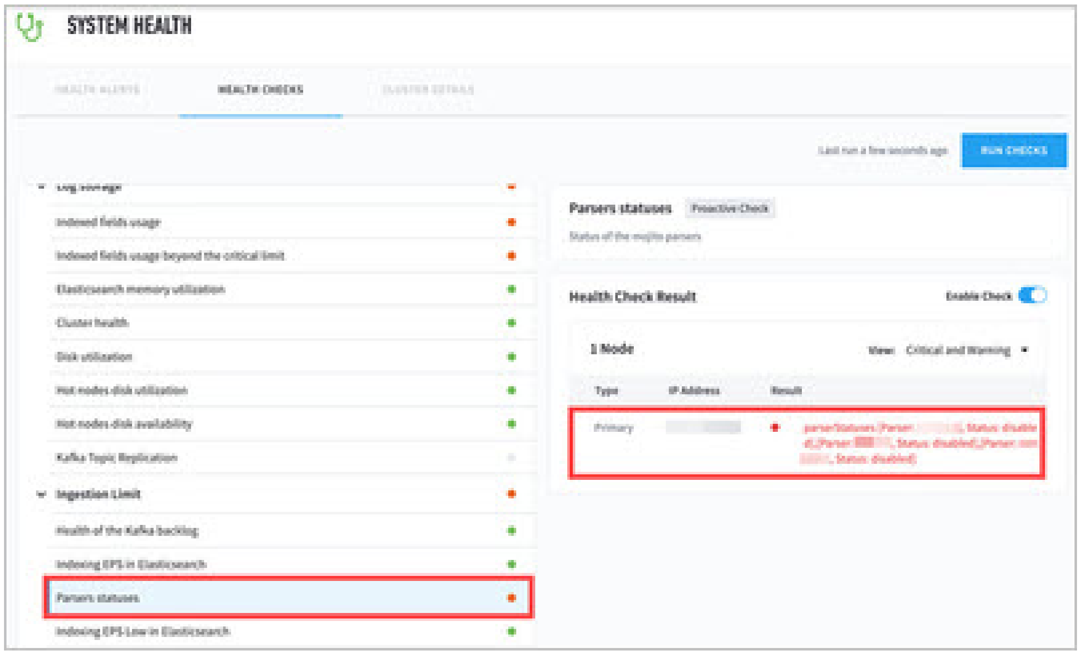
2.3. Find the Paused Parser

Parsers are sometimes paused in Data Lake when they are determined to be parsing too slowly. Therefore, you may notice that outputs have suddenly stopped from a particular parser.

There are two ways you can verify which parsers have been paused:

Method 1 - Health Check UI

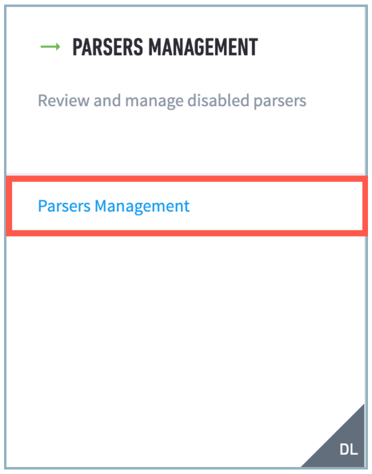
Use the Data Lake UI to navigate to **System Health** > **Health Checks** > **Ingestion Limit** > **Parser statuses**.


SYSTEM HEALTH
HEALTH ALERTS
HEALTH CHECKS
CLUSTER DETAILS
Last run a few seconds ago
RUN CHECKS
Log Storage
Indexed fields usage
Indexed fields usage beyond the critical limit
Elasticsearch memory utilization
Cluster health
Disk utilization
Hot nodes disk utilization
Hot nodes disk availability
Kafka Topic Replication
Ingestion Limit
Health of the Kafka backlog
Indexing EPS in Elasticsearch
Parsers statuses
Indexing EPS Low in Elasticsearch
Parsers statuses
Proactive Check
Status of the mojito parsers
Health Check Result
Enable Check
1 Node
View: Critical and Warning

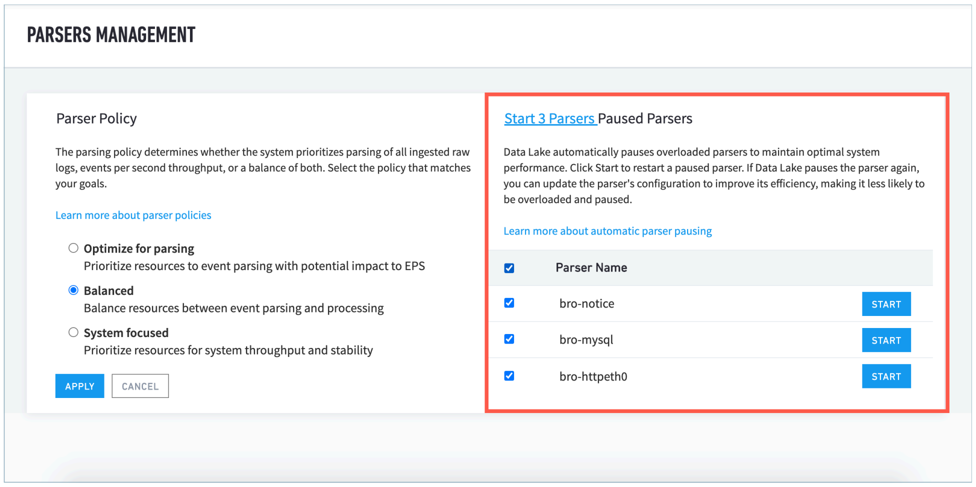
Type	IP Address	Result
Primary	[illegible]	parserStatuses [Parser: [illegible], Status: disabled],[Parser [illegible], Status: disabled],[Parser: [illegible], Status: disabled]

Method 2 - Custom Parsers Management

1. Navigate to **Settings** > **Parsers Management** > **Parsers Management**.



2. Review the list of **Paused Parsers**. If you have resolved the cause of the slow parser, you may re-enable the paused parser by clicking **START** or the hyperlink (shown as **Start 3 Parsers** in this example) for a batch start.


PARSERS MANAGEMENT

Parser Policy

The parsing policy determines whether the system prioritizes parsing of all ingested raw logs, events per second throughput, or a balance of both. Select the policy that matches your goals.

Learn more about parser policies

( ) Optimize for parsing
Prioritize resources to event parsing with potential impact to EPS

[✓ Balanced]
Balance resources between event parsing and processing

( ) System focused
Prioritize resources for system throughput and stability

APPLY CANCEL

Start 3 Parsers Paused Parsers

Data Lake automatically pauses overloaded parsers to maintain optimal system performance. Click Start to restart a paused parser. If Data Lake pauses the parser again, you can update the parser's configuration to improve its efficiency, making it less likely to be overloaded and paused.

Learn more about automatic parser pausing

✓	Parser Name	
✓	bro-notice	START
✓	bro-mysql	START
✓	bro-httpeth0	START

2.4. Investigate the Paused Parser

For more clues, run a search for logs that went through the parser before it was paused, download the logs, and create a case for the Exabeam Content Team with the files attached.

Here is an example of how to investigate the root cause of the paused parser:

1. The customer receives a health alert posted to Search UI. This routes the customer to System Health Page where paused parsers are listed. Note which parser was labeled as `disabled`.
2. A ticket should be started with Exabeam Customer Success to initiate the investigation. Confirm the status of the parser and review logs for activities prior to the parser getting paused.

NOTE
While you continue to work with Exabeam, you may also choose to restart the slow parser on your own. This may help in situations where the log source sent an invalid or long log/event not conforming to the optimized parser.

To find and review the logs:

1. Find the last time the parser was running by reviewing the output from the following commands:

```
mongo
use dl_metrics_db
db.parser_status_collection.find().pretty()
```

2. Look for the `timestamp` in the table output for the last entry in the record for the paused parser. The example below shows the record for parser `raw-4658` at the time it became `disabled`.

```
{
    "_id" : ObjectId("5ce43963130b38000c98556c"),
    "timestamp" : "2019/05/21 05:46:11",
    "parsers" : [
        {
```

```
        "name" : "raw-4658",
        "status" : "disabled",
        "isEnable" : false
        }
    ]
}
```

Alternatively, you can also run the following query, which returns the first time the parser was paused:

```
db.parser_status_collection.find({parsers: {$all: [{ "name" : "PARSER_NAME",
"status" : "disabled", "isEnable" : false }]}}).limit(1)
// example output, remember the timestamp of the first record
{ "_id" : ObjectId("5ce3fa16130b38000c79ecff"), "timestamp" : "2019/05/21
01:16:06", "parsers" : [ { "name" : "raw-4658", "status" : "disabled",
"isEnable" : false } ] }
```

3. Note the parser’s execution run time that triggered the disabling. Review the output from:

```
sos; mongo --quiet dl_metrics_db --eval
'db.parser_metrics_collection.find().forEach(printjson)'
```

For example:

```
db.parser_metrics_collection.find({$and:[{"metricTag" : "PARSER_NAME"},
{"timestamp" : "2019/05/21 01:16:06"}]})
```

```
// example output:
{ "_id" : ObjectId("5ce3fa16130b38000c79ecf7"), "metricGroup" : "parser",
"metricTag" : "raw-4658", "metricType" : "timeTakenInNanos", "timestamp" :
"2019/05/21 01:16:06", "values" : [ { "metricAggregatedType" : "minValue",
"value" : NumberLong(12300042) }, { "metricAggregatedType" : "averageValue",
"value" : NumberLong(23563191) }, { "metricAggregatedType" : "maxValue",
"value" : NumberLong(116349011) } ] }
```

4. Depending on the parer's execution run time, complete one of the following steps:
   - If the average execution time is higher than twice the configured threshold, proceed to step 4.
   - If the average time is less than half of the configured threshold, open a ticket with the Data Lake team.
   - If the average time is between half the threshold and twice the threshold, troubleshoot with threshold tuning. Apply the tuning steps prescribed in *Scenario 3 - Performance Thresholds Set Too Low for Disabling Parsers.*

5. Run a search of the logs for the paused parser for the time before its disabling (timestamp), say 15 minutes prior to the event. This may give clues to which action initiated slow parsing:
   `* AND exa_parser_name: "PARSER_NAME"`

2.5. Identify and Fix a Slow Parser

Find hints within the parser code to help direct your efforts. Go to the following to review the parser configuration:

- Before Data Lake i24: `/opt/exabeam/config/lms/elasticsearch/custom_mojito.conf`
- For Data Lake i24 to i31: `/opt/exabeam/config/lms/elasticsearch/ingest_mojito/custom_mojito.conf`
- For Data Lake i32 and later: `/opt/exabeam/config/lms/mojito-kafka-connect/ingest-mojito/custom_mojito.conf`

NOTE
You can find the default config file at `/opt/exabeam/config/lms/mojito-kafka-connect/ingest-mojito/mojito_default.conf`. The custom config file that you should edit can be found at `/opt/exabeam/config/lms/mojito-kafka-connect/ingest-mojito/custom_mojito.conf`.

Then run the following to extract Elasticsearch log entries, and then view the output.

Tail:

```
sudo journalctl -u elasticsearch-a -f
```

Viewing:

```
sudo journalctl -u elasticsearch-a -e
```

From reviewing the parser logs, determine which of the following common issues match your scenario:

- **Scenario 1** - Unintentional performance lag due to malformed date format. (See example below.)
- **Scenario 2** - Unintentional performance lag due to an error in the parser code.
  One common issue for parsers that manifest as slow performance is error in the parser code. Another possible cause is that the parser is inefficient (for example, it parses four fields rather than three fields but produces the same result).
- **Scenario 3** - Intentional performance lag due to the complexity of the parser.
  You may expect a parser to take a long time to run due to the nature of the logs or the complexity of the parser. In such cases, adjust the performance thresholds to prevent false-positive parser disabling.

NOTE
For the short-term, you can re-enable all parsers by the parser engine. This can be done manually or by running sos and then apply parsers script under `/opt/exabeam/bin/lms`. The parser will need to be restarted to implement changes.

- For Data Lake i31 and earlier:

```
/opt/exabeam/bin/lms/apply-parsers-and-restart-ingest-nodes.py
```

- For Data Lake i32 and later:

```
sos
dl-mojito-restart -enable-parsers
```

2.5.1. SCENARIO 1 - MALFORMED DATE FORMAT

The timestamp in the log is not being parsed correctly.

> **NOTE**
> You can find the error in the mojito log file:

- For Data Lake i31 and earlier, run

```
journalctl -eu elasticsearch-b -f
```

- For Data Lake i32 and later, run

```
journalctl -eu exabeam-dl-mojito-kafka-connect -f
```

Below is an example of the error that notes which parser and what part of the time is malformed:

```
Oct 11 16:20:23 exabeam02 docker[19776]: Privileges: SeBackupPrivilege</
Message><Level>Information</Level><Task>Sensitive Privilege Use</
Task><Opcode>Info</Opcode><Channel>Security</Channel><Provider>Microsoft
Windows security auditing.</Provider><Keywords><Keyword>Audit Success</
Keyword></Keywords></RenderingInfo></Event>, beat={name=beat_name,
version=5.1.2}, @version=1, forwarder=forwarder_name, data_type=windows-
privileged-access, _id=null, exa_parser_name=l-4674, source_name=Microsoft-
Windows-Security-Auditing}, ingestMetadata={timestamp=Wed Oct 11 16:20:23 UTC
2017}}
Oct 11 16:20:23 exabeam02 docker[19776]: java.lang.IllegalArgumentException:
Invalid format: "2017-10-04T16:49:40.908434400Z" is malformed at "434400Z"
Oct 11 16:20:23 exabeam02 docker[19776]: at
org.joda.time.format.DateTimeFormatter.parseDateTime(DateTimeFormatter.java:945)
Oct 11 16:20:23 exabeam02 docker[19776]: at
com.exabeam.mojito.elasticsearch.LimeIngestProcessor.com$exabeam$mojito
$elasticsearch$LimeIngestProcessor$
$strToDateTimeWithFormat(LimeIngestProcessor.scala:111)
```

Solution: Edit the time field of the parser in question (l-4674 in the above example) to match the given log.

Exabeam supports Unix timestamp formats for parsers, as well as any format that is Unix readable. If the time field is parsed as a 10 digit number, epoch time, then the value for TimeFormat would be 'epoch'.

In the solution below, TimeFormat is set to seconds rather than milliseconds. For the example above, since the milliseconds (SSS) is not in the proper format, you can trim the format to not look for that. Note that the millisecond is not required in TimeFormat anymore. An improper TimeFormat can prevent lime from starting in Exabeam products.

See the changes below that were made to the parser:

Original Parser:

```
TimeFormat = "yyyy-MM-dd'T'HH:mm:ss.SSS Z"
"""<TimeCreated SystemTime='({time}\d\d\d\d-\d\d-\d\dT\d\d:\d\d:\d\d.\d*Z+)'/
>"""
```

Custom Parser Change:

```
TimeFormat = "yyyy-MM-dd'T'HH:mm:ss"
"""<TimeCreated SystemTime='({time}\d\d\d\d-\d\d-\d\dT\d\d:\d\d:\d\d)"""
```

Restart the ingestor on the main node and the changes should take place across each node.

- For Data Lake i31 and earler:

  ```
  /opt/exabeam/bin/lms/apply-parsers-and-restart-ingest-nodes.py
  ```

- For Data Lake i32 and later:

  ```
  sos
  dl-mojito-restart -enable-parsers
  ```

2.5.2. SCENARIO 2 - POOR EPS DUE TO SLOW PARSING

If you suspect that parsing is causing low EPS, start with determining what the performance baseline should be to compare against.

You can find the index EPS on the UI at: **System Health** > **Indexing Metrics** > **Indexer diagram**

Or you can manually determine it by following these steps:

1. Run a search in the UI. Suggestion is to search for * for a 15m period.
2. Check the amount of results in the UI. This should be visible, essentially between the search bar and the actual results.
3. Divide the amount of results by 15m * 60s.
4. Example on 4,000,000 results: `((eps=4000000/(15*60)); echo $eps`
5. Then, check the logs to determine which node is the ingestor. Here is an example query:
   `journalctl -exu elasticsearch-b`
   Note that the ingest node is usually the first node on a cluster, however the second ES node on the master DL appliance will have the ingest role. To see which node(s) have the ingest role, check for the i role in the `node.role` column of:
   `curl -ks "https://localhost:9200/_cat/nodes?h=node.role,name"`

If you are not sure if a particular parser is the cause of the slow down, an item you could check is the performance of each parser:

- For Data Lake i31 and earlier, run:

  ```
  $ journalctl --since yesterday -exu elasticsearch-b | grep "Slow Parsing
  with" | awk '{print $NF}' | sort | uniq -c | sort -rn
  ```

  Example output:

  ```
        7 pan-proxy]
        3 s-checkpoint-proxy]
        3 paloalto-firewall-allow-2]
        3 checkpoint-firewall-drop-2]
        2 xml-4624]
        2 l-4688-v2]
        1 s-xml-4634]
        1 cisco-asa-all]
        1 checkpoint-firewall-accept-2]
  ```

- For Data Lake i32 and later, run:

  ```
  $ journalctl --since yesterday -exu exabeam-dl-mojito-kafka-connect | grep
  "Slow Parsing with" | awk '{print $NF}' | sort | uniq -c | sort -rn
  ```

  Example output:

  ```
        7 pan-proxy]
        3 s-checkpoint-proxy]
        3 paloalto-firewall-allow-2]
        3 checkpoint-firewall-drop-2]
        2 xml-4624]
        2 l-4688-v2]
        1 s-xml-4634]
        1 cisco-asa-all]
        1 checkpoint-firewall-accept-2]
  ```

This should give you a list of one or more slow parsers on your system.

2.5.2.1. Pause a Specific Parser

To pause a specific parser:

1. Comment out the problematic parser record in `custom_mojito.conf`.
   - For Data Lake i24 and earlier, edit: `/opt/exabeam/config/lms/elasticsearch/custom_mojito.conf`
   - For Data Lake i24 to i31, edit: `/opt/exabeam/config/lms/elasticsearch/ingest_mojito/custom_mojito.conf`
   - For Data Lake i32 and after, edit: `/opt/exabeam/config/lms/mojito-kafka-connect/ingest-mojito/custom_mojito.conf`

2. Restart the parsing engine to implement the changes.
   - For Data Lake i31 and earlier, run:

     ```
     /opt/exabeam/bin/lms/apply-parsers-and-restart-ingest-nodes.py
     ```

   - For Data Lake i32 and later, run:

     ```
     sos
     dl-mojito-restart -enable-parsers
     ```

2.5.2.2. Pausing All Parsing

You can pause all parsing if you want to test the EPS rate.

1. Add a `Parsers=[]` line underneath the entire parser configuration in `custom_mojito.conf`.
   - For Data Lake i24 and earlier, edit: `/opt/exabeam/config/lms/elasticsearch/custom_mojito.conf`
   - For Data Lake i24 to i31, edit: `/opt/exabeam/config/lms/elasticsearch/ingest_mojito/custom_mojito.conf`
   - For Data Lake i32 and after, edit: `/opt/exabeam/config/lms/mojito-kafka-connect/ingest-mojito/custom_mojito.conf`
2. Reload the parsers via the following commands:
   - For Data Lake i31 and earler, run:

     ```
     /opt/exabeam/bin/lms/apply-parsers-and-restart-ingest-nodes.py
     ```

   - For Data Lake i32 and later, run:

     ```
     sos
     dl-mojito-restart
     ```

3. Re-calculate the EPS for the comparison. If desired, simply remove the `Parsers=[]` line from the configuration and reload the parsers.

2.5.3. SCENARIO 3 - PERFORMANCE THRESHOLDS THAT PAUSE PARSERS ARE SET TOO LOW

There may be parsers that are expected to run for some time, not because there is an error in the code but that the parsing is very complex and the volume of logs dictates the run time to be so. You can tune the threshold for disabling parsers to prevent breaching the trigger threshold.

1. Edit the configuration file based on the Data Lake version you are running.
   - For pre-Data Lake i24: `/opt/exabeam/config/lms/elasticsearch/custom_mojito.conf`

- For Data Lake i24 to i31:
  `/opt/exabeam/config/lms/elasticsearch/ingest_mojito/custom_mojito.conf`
- For Data Lake i32 and after:
  `/opt/exabeam/config/lms/mojito-kafka-connect/ingest-mojito/custom_mojito.conf`

2. Tune up the following fields to prevent false-positive parser disabling:
   - `DisableThresholdTime`
   - `DisableThresholdPeriods`
3. Restart all parsers when you have finished editing.
   - For Data Lake i31 and earler:

     ```
     /opt/exabeam/bin/lms/apply-parsers-and-restart-ingest-nodes.py
     ```

   - For Data Lake i32 and later:

     ```
     sos
     dl-mojito-restart -enable-parsers
     ```

NOTE
The parser thresholds are global values and changing them may impact detecting other slow parsers. This may also delay finding Scenario 2 described above.

2.6. Re-enable All Parsers in Data Lake

When configuration changes are made to Data Lake parsers, it may be necessary to re-enable them if the reconfiguration process does not.

At the master node, run the following commands to re-enable all parsers:

```
// python scripts in /opt/exabeam/bin/lms
dl-mojito-clean-disabled-parsers // drop dl_metrics_db
sos; mongo --quiet dl_metrics_db --eval 'db.parser_status_collection.drop()'
sos; mongo --quiet dl_metrics_db --eval 'db.parser_metrics_collection.drop()'

// restart mojito
dl-mojito-restart // sync custom_mojito.conf to all nodes
ansible all -i inventory -m shell --args='sudo systemctl restart exabeam-dl-
mojito-kafka-connect'
```

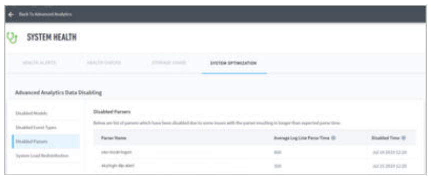
3. Troubleshoot Advanced Analytics Parser

Advanced Analytics automatically identifies poor parser performance and disables such parsers in order to preserve the system health.

NOTE
You are shown an indicator when Advanced Analytics determines that a parser is problematic and disables it.

3.1. Find Disabled Parsers

Navigate to **System Health** > **System Optimization**, and then click on the **Data Disabling** tab.



You will see the list of disabled parsers. The table includes columns with the following categories:

- **Parser Name** – The name of the disabled parser.
- **Average Log Line Parse Time** – Average time taken by the parser to parse each event.
- **Disabled Time** – Date and time when the parser was disabled.

Alternatively, query MongoDB for the contents of `Disable_parser_db.current_collection`, if the Advanced Analytics UI is not available:

- For each monitoring interval, `lime.log` will contain "`Disabled parsers for this period are:`"
- When disabling one parser , `lime.log` will note "`Disabling parser:...`"

NOTE
It will also output average parsing time periodically as configured through `OutputParsingTimePeroidInMinutes`.

To find out the historical disabled parsers, query MongoDB for the contents of `Disable_parser_db.historical_collection`.

3.2. Fix a Disabled Parser

Once a parser is disabled it will show with in the UI as stated in the previous section. You can also find the corresponding log message associated with the disabled parser in `/opt/exabeam/data/logs/exabeam.log`.

Determine which parser has been disabled:

1. Obtain the entry in the exabeam.log for the disabled parser.
2. Create a ticket with the Exabeam Content Team to troubleshoot the parser. Include logs for the parser from the hour before it was disabled.
3. Once the Exabeam Content Team delivers the fixed parser, you can apply it and then restart the parser service:

   ```
   lime-stop; lime-start
   ```

Adjust the threshold:

- If you want to see only the parsing statistics in the logs, for example, what is the average parsing time for each parser, then you can overwrite `LogParser.OutputParsingTime` to true.
- If you want to turn off parser disabling, you can set `LogParser.AllowDisableParser = true`. There is no need to turn on the `OutputParsingTime` flag, because once the `AllowDisableParser` is enabled, it will automatically output the statistics.

Restart the parsing engine after threshold adjustments are made.

```
lime-stop; lime-start
```

3.2.1. THRESHOLD TUNING

You may expect a parser to take a long time to run due to the nature of the logs or the complexity of the parser. In such cases, re-enable the parser and adjust the performance thresholds to prevent false-positive parser disabling.

Tune up the following two parameters to prevent false-positive disabling of the parser:

- `LogParser.ParserDisableThresholdInMills`
- `LogParser.ParserDisableTimePercentage`

To configure the thresholds, navigate to the **`LogParser`** section located in the `lime_default.conf` file at `/opt/exabeam/config/default/`.

All changes should be made to `/opt/exabeam/config/custom/custom_lime_config.conf`:

```
LogParser{
          #Output parsing performance in debug mode, be cautious this might
affect performance in parsing
          OutputParsingTime = true
          OutputParsingTimePeriodInMinutes = 5
          AllowDisableParser = true // If this is enabled, output parsing time
will be enabled by default.
          ParserDisableThresholdInMills = 30 //If average parsing time pass
this threshold, we will disable that parser
          ParserDisableTimePercentage = 0.5
}
```

Acceptable values for `ParserDisableThresholdInMills` includes any integer value. `ParserDisablePercentage` can be a percentage decimal value between 0.1 to 0.9.

NOTE
Setting a higher parsing time percentage identifies less severe parsers.