Fast Log Analysis Made Easy by Automatically Parsing Heterogeneous Logs

Biplob Debnath and Will Dennis

NEC Laboratories America, Inc.

{biplob,wdennis}@nec-labs.com
Log Analysis

- Log ubiquitously exists in many complex systems.
- Most of the logs include massive amount of transaction or status data.
- Tons of logs are generated, but difficult to manually investigate.
Log Analysis: Example

Raw Log Streams

L1
L2
L3
L4
.
.
.
Ln

Log Analyzer

Analyzes logs using various features and reports alerts

New Log Pattern
Alert
Alert

Sample Alert

Log Rate Change
Alert
Alert

Log Relation Violation
Alert
Alert

Rare Log Event
Alert
Alert

Log Rate Change
Alert
Alert

Log Analyzer

Raw Log Streams

L1
L2
L3
L4
.
.
.
Ln

Log Analyzer

Analyzes logs using various features and reports alerts

New Log Pattern
Alert
Alert

Sample Alert

Log Rate Change
Alert
Alert

Log Relation Violation
Alert
Alert

Rare Log Event
Alert
Alert
Log Parsing is the Core Step to any type of Log Analysis
Log Parsing: Example

Mar 3 16:30:04 envctl APC_PDU_LEGAMPS: [INFO] PDU=pdu2z04-am-rack4f Leg=3 Amps=2.5

Parsing Pattern ???

DATETIME envctl APC_PDU_LEGAMPS: [INFO] PDU=NOTSPACE Leg=NUMBER Amps=NUMBER

```json
{
  "message": "Mar 3 16:30:04 envctl APC_PDU_LEGAMPS: [INFO] PDU=pdu-2z04-am-rack4f Leg=3 Amps=2.5",
  "timestamp": "2017-03-03T21:30:04.000Z",
  "PDU": "pdu-2z04-am-rack4f",
  "Leg": "3",
  "Amps": "2.5"
}
```
Heterogeneous Log Formats

Oct 23 13:53:39 am12-09 kernel: [448260.543317] sd 6:0:0:0: [sdc] tag#0 FAILED Result: hostbyte=DID_ERROR driverbyte=DRIVERSENSE

Oct 23 13:53:39 am12-09 kernel: [448260.543324] sd 6:0:0:0: [sdc] tag#0 Sense Key : Hardware Error [current] [descriptor]

Oct 23 13:53:39 am12-09 kernel: [448260.543335] sd 6:0:0:0: [sdc] tag#0 Add. Sense: No additional sense information

2017-10-04T12:39:44.269-0400 I NETWORK [HostnameCanonicalizationWorker] Starting hostname canonicalization worker
2017-10-04T12:39:44.270-0400 I NETWORK [initandlisten] waiting for connections on port 27017

2017-10-04T14:43:33.061 13625 INFO nova.virt.libvirt.driver [-] [instance: 9d9ff3bf-8b27-4330-8d8a-4bfff5a782477] Instance destroyed successfully. 172.16.4.121 /var/log/nova/nova-compute.log
2017-10-04T14:44:02.446 13625 AUDIT nova.compute.resource_tracker [-] Auditing locally available compute resources 172.16.4.121 /var/log/nova/nova-compute.log

2014-11-12 16:44:02.446 13625 AUDIT nova.compute.resource_tracker [-] Auditing locally available compute resources 172.16.4.121 /var/log/nova/nova-compute.log

2014-11-12 16:44:02.446 13625 AUDIT nova.compute.resource_tracker [-] Auditing locally available compute resources 172.16.4.121 /var/log/nova/nova-compute.log


2017-10-04T12:39:44.269-0400 I NETWORK [HostnameCanonicalizationWorker] Starting hostname canonicalization worker
2017-10-04T12:39:44.270-0400 I NETWORK [initandlisten] waiting for connections on port 27017

Pattern Generation: Challenges

• Various log formats that can change over time

• Extremely huge amount of system logs

• Limited domain knowledge

Goal: Generate patterns with no or minimal human involvement.
Outline

• Pattern Generation Algorithm

• Demo Use-cases
Outline

• Pattern Generation Algorithm

• Demo Use-cases
Pattern Generation: Problem Statement

• **Input**
  - A set of logs
  - Optional
    - Tokenization Delimiter
    - Max Pattern limit

• **Output**
  - A set of patterns to parse all input logs
Sample Input Logs

1. 2017/02/24 09:01:00 login 127.0.0.1 user=bear12
2. 2017/02/24 09:02:00 DB Connect 127.0.0.1 user=bear12
3. 2017/02/24 09:02:00 DB Disconnect 127.0.0.1 user=bear12
4. 2017/02/24 09:04:00 logout 127.0.0.1 user=bear12

5. 2017/02/24 09:05:00 login 127.0.0.1 user=bear34
6. 2017/02/24 09:06:00 DB Connect 127.0.0.1 user=bear34
7. 2017/02/24 09:07:00 DB Disconnect 127.0.0.1 user=bear34
8. 2017/02/24 09:08:00 logout 127.0.0.1 user=bear34

9. 2017/02/24 09:09:00 login 127.0.0.1 user=bear#1
10. 2017/02/24 09:10:00 DB Connect 127.0.0.1 user=bear#1
11. 2017/02/24 09:11:00 DB Disconnect 127.0.0.1 user=bear#1
12. 2017/02/24 09:12:00 logout 127.0.0.1 user=bear#1

Pattern Count?

Is it possible to give an option to the users to cherry-pick pattern-set?
Pattern-Tree: Providing Cherry-Picking Options

Input Logs

1. 2017/02/24 09:01:00 login 127.0.0.1 user=bear12
2. 2017/02/24 09:02:00 DB Connect 127.0.0.1 user=bear12
3. 2017/02/24 09:02:00 DB Disconnect 127.0.0.1 user=bear12
4. 2017/02/24 09:04:00 logout 127.0.0.1 user=bear12

5. 2017/02/24 09:05:00 login 127.0.0.1 user=bear34
6. 2017/02/24 09:06:00 DB Connect 127.0.0.1 user=bear34
7. 2017/02/24 09:07:00 DB Disconnect 127.0.0.1 user=bear34
8. 2017/02/24 09:08:00 logout 127.0.0.1 user=bear34

9. 2017/02/24 09:09:00 login 127.0.0.1 user=bear#1
10. 2017/02/24 09:10:00 DB Connect 127.0.0.1 user=bear#1
11. 2017/02/24 09:11:00 DB Disconnect 127.0.0.1 user=bear#1
12. 2017/02/24 09:12:00 logout 127.0.0.1 user=bear#1

Pattern-Tree

1. DATETIME login IPV4 user = WORD
2. DATETIME DB Connect IPV4 user = WORD
3. DATETIME DB Disconnect IPV4 user = WORD
4. DATETIME logout IPV4 user = WORD
5. DATETIME login IPV4 user = NOTSPACE
6. DATETIME DB Connect user = NOTSPACE
7. DATETIME DB Disconnect IPV4 user = NOTSPACE
8. DATETIME logout IPV4 user = NOTSPACE
9. DATETIME login IPV4 user = NOTSPACE
10. DATETIME DB Connect IPV4 user = NOTSPACE
11. DATETIME DB Disconnect IPV4 user = NOTSPACE
12. DATETIME logout IPV4 user = NOTSPACE
13. DATETIME WORD IPV4 user = NOTSPACE
14. DATETIME DB WORD IPV4 user = NOTSPACE
15. DATETIME * WORD IPV4 user = NOTSPACE
LogMine: An Automatic Pattern Generator

- LogMine: Fast Pattern Recognition for Log Analytics [CIKM 16]

- Logmine generates a pattern-tree from a set of input logs
  - User can cherry-pick any set of patterns from this tree
  - It is a super-Fast algorithm
    - Scans log data only once
    - Scalable
      - Time complexity O(# of logs)
      - Memory complexity O(# of clusters)

- LogMine leverage the following facts
  - Logs are not randomly generated, rather generated by computing devices
  - Logs having same formats are very similar
LogMine: Workflow

Set of N logs

Tokenization Delimiter
Max Pattern Limit
optional

Preprocessor \[\rightarrow\] Clustering \[\rightarrow\] Pattern Recognition

MaxDistance=0.00001 (initial value)

Increase MaxDistance (Relax Clustering Conditions)

Add one level to the Pattern-Tree

Set of Patterns
LogMine: Three Steps

1. **Preprocessing**
   a) Tokenization
   b) Datatype identification

2. **Pattern-Tree Generation (Iterative Process)**
   a) Starts with a small similarity distance threshold
   b) Form clusters based similarity among logs
   c) If any cluster has multiple logs, merge them together to form one log.
   d) Add clusters in the pattern-tree
   e) If more than one clusters formed
      • Increase similarity distance threshold
      • Repeat Step a, b, c, d, e

3. **Output Final Patterns from the Pattern-Tree**
   a) If user inputs a max pattern limit, then outputs the first level closer to Level-0 satisfying user’s limit
   b) Otherwise, outputs the tree level having minimum patterns with no wildcards (i.e., cost = 0)
Pre-Processing: Tokenization

Each Individual word is a token
- Split token by tokenization delimiter if specified by the user.
  - In this example “=” is the delimiter.
- Consecutive spaces are consolidated
### Preprocessing: Data-Type Identification

<table>
<thead>
<tr>
<th></th>
<th>Event</th>
<th>Date</th>
<th>Time</th>
<th>IP Address</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>login</td>
<td>2017/02/24</td>
<td>09:01:00</td>
<td>127.0.0.1</td>
<td>bear12</td>
</tr>
<tr>
<td>2</td>
<td>DB Connect</td>
<td>2017/02/24</td>
<td>09:02:00</td>
<td>127.0.0.1</td>
<td>bear12</td>
</tr>
<tr>
<td>3</td>
<td>DB Disconnect</td>
<td>2017/02/24</td>
<td>09:02:00</td>
<td>127.0.0.1</td>
<td>bear12</td>
</tr>
<tr>
<td>4</td>
<td>logout</td>
<td>2017/02/24</td>
<td>09:04:00</td>
<td>127.0.0.1</td>
<td>bear12</td>
</tr>
<tr>
<td>5</td>
<td>login</td>
<td>2017/02/24</td>
<td>09:05:00</td>
<td>127.0.0.1</td>
<td>bear34</td>
</tr>
<tr>
<td>6</td>
<td>DB Connect</td>
<td>2017/02/24</td>
<td>09:06:00</td>
<td>127.0.0.1</td>
<td>bear34</td>
</tr>
<tr>
<td>7</td>
<td>DB Disconnect</td>
<td>2017/02/24</td>
<td>09:07:00</td>
<td>127.0.0.1</td>
<td>bear34</td>
</tr>
<tr>
<td>8</td>
<td>logout</td>
<td>2017/02/24</td>
<td>09:08:00</td>
<td>127.0.0.1</td>
<td>bear34</td>
</tr>
<tr>
<td>9</td>
<td>login</td>
<td>2017/02/24</td>
<td>09:09:00</td>
<td>127.0.0.1</td>
<td>bear#1</td>
</tr>
<tr>
<td>10</td>
<td>DB Connect</td>
<td>2017/02/24</td>
<td>09:10:00</td>
<td>127.0.0.1</td>
<td>bear#1</td>
</tr>
<tr>
<td>11</td>
<td>DB Disconnect</td>
<td>2017/02/24</td>
<td>09:11:00</td>
<td>127.0.0.1</td>
<td>bear#1</td>
</tr>
<tr>
<td>12</td>
<td>logout</td>
<td>2017/02/24</td>
<td>09:12:00</td>
<td>127.0.0.1</td>
<td>bear#1</td>
</tr>
</tbody>
</table>

- **Identify following datatypes:** DATETIME, IPV4, NUMBER, WORD, NOTSPACE
  - **Exception:** Symbols and token having only alphabets
  - **Intuition:** Developers uses meaningful words to interpret log messages 😊
LogMine: Workflow (Recap)

Set of N logs

Log1
Log2
Log3
....
LogN

Tokenization Delimiter
Max Pattern Limit
optional

Preprocessor → Clustering → Pattern Recognition

MaxDistance=0.00001 (initial value)

Relax Clustering Conditions (increase MaxDistance)

Set of Patterns

Add one level to the Pattern-Tree
LogMine: Pattern-Tree Formation

1. DATETIME login IPV4 user = WORD
2. DATETIME DB Connect IPV4 user = WORD
3. DATETIME DB Disconnect IPV4 user = WORD
4. DATETIME logout IPV4 user = WORD
5. DATETIME login IPV4 user = NOTSPACE
6. DATETIME DB Connect user = NOTSPACE
7. DATETIME DB Disconnect IPV4 user = NOTSPACE
8. DATETIME logout IPV4 user = NOTSPACE

9. DATETIME login IPV4 user = NOTSPACE
10. DATETIME DB Connect IPV4 user = NOTSPACE
11. DATETIME DB Disconnect IPV4 user = NOTSPACE
12. DATETIME logout IPV4 user = NOTSPACE

13. DATETIME WORD IPV4 user = NOTSPACE
14. DATETIME DB WORD IPV4 user = NOTSPACE

15. DATETIME * WORD IPV4 user = NOTSPACE

Pattern Tree

Memory Usage depends on Level 1 Size
Optimization: Fast Level 1 Formation

Level 1 could be formed without clustering

Hint: Identify unique log lines.
Optimization: Fast Level 1 Formation

Preprocessed Logs

Level 1 could be formed without clustering
Hint: Identify unique log lines.
Clustering: Example

MaxDistance = 0.01

<table>
<thead>
<tr>
<th>Sample LOG</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG1</td>
</tr>
<tr>
<td>LOG2</td>
</tr>
<tr>
<td>LOG3</td>
</tr>
<tr>
<td>LOG4</td>
</tr>
<tr>
<td>LOG5</td>
</tr>
<tr>
<td>LOG6</td>
</tr>
<tr>
<td>LOG7</td>
</tr>
<tr>
<td>LOG8</td>
</tr>
<tr>
<td>LOG9</td>
</tr>
</tbody>
</table>
Clustering: Example

MaxDistance = 0.01
Clustering: Example

MaxDistance = 0.01

Dist (LOG1, LOG2) = 0.2
Example: Distance Calculation

Log1: DATETIME login IPV4 WORD NUMBER
Log2: DATETIME login IPV4 WORD

Similarity (Log1, Log2) = 4/5 = 0.80
Distance (Log1, Log2) = 1 − Similarity = 1 − 0.80 = 0.20

\[
Dist(P, Q) = 1 - \sum_{i=1}^{\min(|P|,|Q|)} \frac{Score(P_i, Q_i)}{\max(|P|,|Q|)}
\]

\[
Score(X, Y) = \begin{cases} 
1 & \text{If } X = Y \\
0 & \text{Otherwise}
\end{cases}
\]
Clustering: Example

MaxDistance = 0.01

Dist (LOG3, LOG1) = 0.8
Dist (LOG3, LOG2) = 0.001
Clustering: Example

MaxDistance = 0.01

\[ \text{Dist (LOG4 , LOG1)} = 0 \]
Clustering: Example

MaxDistance = 0.01

Dist (LOG5, LOG1) = 0.2
Dist (LOG5, LOG2) = 0.5

Optimization: Compare only with the first log in a cluster
Clustering: Example

MaxDistance = 0.01

Dist (LOG6, LOG1) = 0.3
Dist (LOG6, LOG2) = 0.001
Clustering: Example

Max Distance = 0.01

Cluster1: LOG1 → LOG4
Cluster2: LOG2 → LOG3 → LOG6 → LOG8
Cluster3: LOG5 → LOG7 → LOG9
Pattern @ Cluster (Smith–Waterman Algorithm)

- Logs in a cluster are merged together to produce one final pattern per cluster.
- Optimization: No need to follow any merging order.

```
LOG1: DATETIME WORD IPV4 user = NOTSPACE
LOG2: DATETIME DB WORD IPV4 user = NOTSPACE
```

Alignment

```
LOG1: DATETIME --- WORD IPV4 user = NOTSPACE
LOG2: DATETIME DB WORD IPV4 user = NOTSPACE
```

Merge the Alignments

```
Result: DATETIME * WORD IPV4 user = NOTSPACE
```
LogMine: Output Pattern-Set

1. DATETIME login IPV4 user = WORD
2. DATETIME DB Connect IPV4 user = WORD
3. DATETIME DB Disconnect IPV4 user = WORD
4. DATETIME logout IPV4 user = WORD
5. DATETIME login IPV4 user = NOTSPACE
6. DATETIME DB Connect user = NOTSPACE
7. DATETIME DB Disconnect IPV4 user = NOTSPACE
8. DATETIME logout IPV4 user = NOTSPACE

Max Pattern Limit = 5, then Level 2 will be the output

Cost is Calculated based on WildCards and covered Logs Count. For example, pattern 15 contains one wildcard and it covers 12 logs – in this case cost is estimated as 1*12 = 12.

If user gives no Max Pattern limit on pattern count, then Level 3 will be the final output as it has the minimum cost with minimum number of patterns.
Outline

• Pattern Generation Algorithm

• Demo Use-cases
LogMine + LogStash: Configuration Generation

WorkFlow

Input Logs → LogMine → Patterns → LogStash Configuration Generator → Config File

Tokenization Delimiter
Max Pattern Limit
optional
Demo 1: Datacenter Power Usage

```
Mar  3 16:25:03 envctl APC_PDU_LEGAMPS: [INFO] PDU=pdu-2g04-apc-rack3a Leg=1
  Amps=4.6
Mar  3 16:25:03 envctl APC_PDU_LEGAMPS: [INFO] PDU=pdu-2g04-rack1a Leg=1
  Amps=4.1
Mar  3 16:30:03 envctl APC_PDU_LEGAMPS: [INFO] PDU=pdu-2g04-apc-rack1a Leg=2
  Amps=1.2
Mar  3 16:30:03 envctl APC_PDU_LEGAMPS: [INFO] PDU=pdu-2g04-apc-rack1a Leg=3
  Amps=0.0
```

Sample Logs

Input Logs: 14976 lines, Tokenization Delimiter : “=”
Patterns: 1,  
Time Taken: 9 seconds (Single core)
Demo 1: LogStash Parsing Configuration

```ruby
filter {
  mutate {
    add_field => { "raw_input" => "${message}" }
  }

  mutate {
    gsub => {
      "message", "="", " \0 ",
      "message", "\s+", " 
    }
  }

  grok {
    match => { "message" => "^\{logTime\}\{MONTH:month\} \{MONTHDAY:day\}\{HOUR:hour\}:\{MINUTE:minute\}:\{SECOND:second\}\ envctl
APC_PDU_LEGAMPS: [INFO] PDU = %{NOTSPACE:PDUName} Leg = %{NUMBER:LegNum:int} Amps = %{NUMBER:Amps:float}$" }
  }

  date {
    match => [ "logTime", "MMM dd HH:mm:ss" ]
    target => "%timestamp"
  }

  mutate {
    remove_field => ["year"]
    remove_field => ["month"]
    remove_field => ["day"]
    remove_field => ["hour"]
    remove_field => ["minute"]
    remove_field => ["second"]
    remove_field => ["logTime"]
  }
}
```
Demo 1: Datacenter Power Usage
Demo 1: Datacenter Power Usage
Demo 2: Custom Application Log

Input Logs: 242232 lines, Tokenization Delimiter: “[=]”
Patterns: 339, Time Taken: 27 seconds (Single core)
Demo 2: Custom Application Log

Sample Logs

Input Logs: 242232 lines, Tokenization Delimiter: "[=]"
Patterns: 339,
Time Taken: 27 seconds (Single core)
### Demo 2: Pattern-Tree Stats

<table>
<thead>
<tr>
<th>Level</th>
<th>Patterns</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2582</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>339</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>256</td>
<td>297396</td>
</tr>
<tr>
<td>4</td>
<td>231</td>
<td>327373</td>
</tr>
<tr>
<td>5</td>
<td>230</td>
<td>327397</td>
</tr>
<tr>
<td>6</td>
<td>219</td>
<td>336073</td>
</tr>
<tr>
<td>7</td>
<td>217</td>
<td>336127</td>
</tr>
<tr>
<td>8</td>
<td>214</td>
<td>336153</td>
</tr>
<tr>
<td>9</td>
<td>213</td>
<td>336175</td>
</tr>
<tr>
<td>10</td>
<td>211</td>
<td>338036</td>
</tr>
<tr>
<td>11</td>
<td>210</td>
<td>338074</td>
</tr>
<tr>
<td>12</td>
<td>195</td>
<td>340653</td>
</tr>
<tr>
<td>13</td>
<td>176</td>
<td>353253</td>
</tr>
<tr>
<td>14</td>
<td>132</td>
<td>456600</td>
</tr>
<tr>
<td>15</td>
<td>121</td>
<td>641338</td>
</tr>
<tr>
<td>16</td>
<td>106</td>
<td>655485</td>
</tr>
<tr>
<td>17</td>
<td>82</td>
<td>669673</td>
</tr>
<tr>
<td>18</td>
<td>57</td>
<td>742742</td>
</tr>
<tr>
<td>19</td>
<td>52</td>
<td>745047</td>
</tr>
<tr>
<td>20</td>
<td>48</td>
<td>749203</td>
</tr>
<tr>
<td>21</td>
<td>39</td>
<td>761506</td>
</tr>
<tr>
<td>22</td>
<td>34</td>
<td>794590</td>
</tr>
<tr>
<td>23</td>
<td>30</td>
<td>795275</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>1171696</td>
</tr>
<tr>
<td>25</td>
<td>20</td>
<td>1333119</td>
</tr>
<tr>
<td>26</td>
<td>9</td>
<td>939438</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>726696</td>
</tr>
</tbody>
</table>

We put no upper limit on pattern count, therefore Level 2 is selected as the final output because it has the minimum cost with minimum number of patterns.
Demo 2: Sample Parsing Pattern

```sql
SELECT TABLE: tblContent WHERE: oPID = 'ad1aa290-01ae-4edd-989c-1cee2ba36707' AND ( ( ( ( iOLD IN SELECT iOLD FROM tblFormData WHERE oFORMINSTID = tblContent.oID AND oFCID = '6a6202aa-9afa-4e2c-95e9-ee900dd4b50' AND (tblFormData.oValue IS NOT NULL AND tblFormData.oValue > '1799-01-01T00:00:00.000') ) AND (iOLD IN SELECT iOLD FROM tblFormData WHERE oFORMINSTID = tblContent.oID AND oFCID = '7e68b547-086f-4a56-a664-2632d2b5804' AND (tblFormData.oValue = '01T00:00:00' OR tblFormData.oValue IS NULL)) ) ) ) AND ( iOLD IN SELECT iOLD FROM tblFormData WHERE oFORMINSTID = tblContent.oID AND oFCID = '2a004b8d-16ef-4973-8ec8-be7db392e436' AND ([tblFormData.oValue=N 'Y' OR tblformData.oValue IS NULL]))) ) ) ) AND (iType=15 OR iOLD IN SELECT iFORMINSTID FROM tblForm WHERE oGrantID = '2995-424-4d61-cb437866a382' OR oGrantID= ANONYMOUS OR oGrantID IN SELECT oParent FROM tblMemb WHERE oChild = 'dadb4506-2995-424-8616-cb437866a382') ) ) )
AND (nSubType=12 AND nSubType=1 AND nSubType=4 AND nSubtype|5) AND (nType=15 OR nVersion=0)
```
Demo 2: Sample Parsed Output

```json
{
  "type": "lisa-demo2",
  "tags": ["pattern199"],
  "P199N1": "(0):_GetObjects():2",
  "@timestamp": "2017-01-25T21:39:21.000Z",
  "P199F1": "15",
  "P199F2": "14",
  "P199N2": "'2017-10-26T03:59:59.000'",
  "P199N3": "N'dadf4506-2995-42c4-8616-cb43786fa382'",
  "P199F3": "15",
  "P199F4": "1",
  "P199F5": "1",
  "P199F6": "15",
  "P199N4": "'dadf4506-2995-42c4-8616-cd43786fa382'",
  "P199N5": "'dadf4506-2995-42c4-8616-cb43786fa382'",
  "P199F7": "2",
  "P199F8": "1",
  "P199F9": "4",
  "P199F10": "15"
}
```
Summary

1. LogMine is fast and scalable 😊
2. It can work with no (or minimal) human Involvement 😊
3. It is flexible because user can cherry-pick any desired pattern-set 😊
4. It is the basic component of a real product 😊

All these benefits come from the fact that LogMine is customized for logs.
Thank you