## How does the parser work?

The parser detects log formats based on a pattern library (YAML file) and converts it to a JSON Object:

- JSON lines are detected, parsed, and scanned for "@timestamp" and "time" fields (Logstash and Bunyan format)
- find matching regex in the pattern library
- tag it with the recognized type
- extract fields using regex
- if 'autohash' is enabled, sensitive data is replaced with its sha256 hash code (or alternative sha512 hash code by configuration)
- parse dates and detect date format (use 'ts' field for date and time combined)
- create ISO timestamp in '@timestamp' field
- call patterns "transform" function to manipulate parsed objects
- unmatched lines end up with a timestamp and original line in the message field
- Logagent includes default patterns for many applications (see below)

The default pattern definition file comes with patterns for:

- MongoDB
- MySQL
- Nginx
- Redis
- Elasticsearch
- Webserver (Nginx, Apache Httpd)
- Zookeeper
- Cassandra
- Kafka
- HBase HDFS Data Node
- HBase Region Server
- Hadoop YARN Node Manager
- Apache Solr
- various Linux/Mac OS X system log files

The file format for pattern definitions is based on JS-YAML, in short:

```
- - indicates an  array
- !js/regexp - indicates a JS regular expression
- !!js/function > - indicates a JS function
```

Properties:

- patterns: list of patterns, each pattern starts with "-"
- match: group of patterns for a specific log source
- blockStart: regular expression indicating a new message block for multi-line logs

- sourceName: regular expression matching the name of the log source (e.g. file or container image name)
- regex: JS regular expression
- fields: field list of extracted match groups from the regex
- type: type used in Logsene (Elasticsearch Mapping)
- dateFormat: format of the special fields ‘ts’. If the date format matches, a new field @timestamp is generated.
- transform: JS function to manipulate the result of regex and date parsing

## Example

```
# Sensitive data can be replaced with a hashcode (sha1)
# it applies to fields matching the field names by a regular expression
# Note: this function is not optimized (yet) and might take 10-15% of performance
autohash: !!js/regexp /user|password|email|credit_card_number|payment_info/i

# set this to false when autohash fields
# the original line might include sensitive data!
originalLine: false

# activate GeoIP lookup
geoIP: true

# Logagent updates GeoIP DB files automatically
# pls. note write access to this directory is required
maxmindDbDir: /tmp/

patterns:
 - # APACHE  Web Logs
  sourceName: httpd
  match:
    # Common Log Format
    - regex:        !!js/regexp /([0-9a-f.:]+)\s+(-|.+?)\s+(-|.+?)\s+\[([0-9]{2}\/[a-z]{3}\/
      type: apache_access_common
      fields:       [client_ip,remote_id,user,ts,method,path,http_version,status_code,size]
      dateFormat: DD/MMM/YYYY:HH:mm:ss ZZ
      # lookup geoip info for the field client_ip
      geoIP: client_ip
      # parse only messages that match this regex
      inputFilter: !!js/regexp /api|home|user/
      # ignore messages matching inputDrop
      inputDrop: !!js/regexp /127.0.0.1|\.css|\.js|\.png|\.jpg|\.jpeg/
      # modify parsed object
      transform: !!js/function >
        function (p) {
```

```
        p.message = p.method + ' ' + p.path
      }
    customPropertyMinStatusCode: 399
    filter: !!js/function >
      function (p, pattern) {
        // log only requests with status code > 399
        return p.status_code > pattern.customPropertyMinStatusCode // 399
      }
```

The handling of JSON is different - regular expressions are not matched against JSON data. Logagent parses JSON and provides post processing functions in the pattern definition. The following example masks fields in JSON and removes fields from the parsed event.

```
hashFunction: sha512
# post process journald JSON format
# logagent feature to hash fields
# and a custom property 'removeFields', used in the transform function
json:
  autohashFields:
    - _HOSTNAME: true
  removeFields:
    - _SOURCE_REALTIME_TIMESTAMP
    - __MONOTONIC_TIMESTAMP
  transform: !!js/function >
   function (source, parsedObject, config) {
     for (var i=0; i<config.removeFields.length; i++) {
       // console.log('delete ' +config.removeFields[i])
       delete parsedObject[config.removeFields[i]]
     }
   }
```

The default patterns are available here - contributions are welcome!

## Node.js API for the parser

Install Logagent as a local module and save the dependency to your package.json

```
npm i logagent-js --save
```

Use the Logparser module in your source code

```
var Logparser = require('logagent-js')
var lp = new Logparser('./patterns.yml')
lp.parseLine('log message', 'source name', function (err, data) {
    if(err) {
      console.log('line did not match any pattern')
    }
    console.log(JSON.stringify(data))
})
```

Use the command line tool 'logagent' to test patterns or convert logs from text to JSON. It reads from stdin and outputs line delimited JSON (or pretty JSON or YAML) to the console. In addition, it can forward the parsed objects directly to Logsene or Elasticsearch.

Test your patterns:

```
cat myapp.log | bin/logagent -y -n myapp -f mypatterns.yml
```