Logagent Plugins

Logagent features a modular architecture. Each input or output module is implemented as a plugin for the Logagent framework. Plugins are loaded on demand as declared in the configuration file.

Plugin	Type	Description
stdin (default)	input	Reads from standard input
files	input	Watching and tailing files
logagent-input-windows-events	input	Collect Windows Events. Available as separate npm package
logagent-input-elasticsearch-stats	input	Monitoring of Elasticsearch metrics. Available as separate npm package
syslog	input	Receive Syslog messages via UDP
elasticsearch-query	input	Receive results from Elasticsearch queries, which could run once or periodically
input-elasticsearch-http	input	Receive documents via Elasticsearch HTTP indexing API (bulk and post)
input-tcp	input	Receive data via TCP
input-mqtt-client	input	Receive data via MQTT client (subscriber for N topics)
input-mqtt-broker	input	Starts an MQTT broker and emits all received events from all topics to Logagent
input-gelf	input	Receive data via GELF protocol
heroku	input	Receive logs from Heroku log drains (HTTP)
cloudfoundry	input	Receive logs from Cloud Foundry log drains (HTTP)
command	input	Receive logs from the output of a command, which could run once or periodically

Plugin	Type	Description
mysql-query	input	Receive results from SQL queries, which could run once or periodically
mssql-query	input	Receive results from SQL queries, which could run once or periodically
postgresql-query	input	Receive results from SQL queries, which could run once or periodically
logagent-input-kafka	input	Receives messages from Apache Kafka topics. 3rd party module.
input-influxdb-http	input	Receives metrics from InfluxDB compatible monitoring agents like Telegraf.
logagent-apple-location	input	Tracking of GPS positions from Apple devices via “find-my-iphone” API
logagent-novasds	input	Read PM10 and PM2.5 values from Nova SDS011 dust sensor (USB to serial interface)
grep	Processor / input filter	Filters text with regular expressions before parsing
sql	Processor / output filter	Transforms and aggregates parsed messages with SQL statements
access-watch	Processor / output filter	Enriches web server logs with robot detection and traffic intelligence
stdout (default)	output	Prints parsed messages to standard output. Supported formats: YAML, JSON, Line delimited JSON (default).
output-gelf	output	Sends data via GELF protocol
output-mqtt	output	Sends messages via MQTT protocol
elasticsearch	output	Stores parsed messages in Elasticsearch
output-aws-elasticsearch	output	Stores parsed messages in Amazon Elasticsearch

Plugin	Type	Description
output-files	output	Stores parsed messages files. Log rotation and dynamic file name generation are supported.
rtail	output	Sends parsed messages to rtail servers for real-time view of logs in a web browser
logagent-output-kafka	output	Sends parsed messages to Apache Kafka topics. 3rd party module. 3rd party module.
slack-webhook	output	Sends parsed messages to Slack chat. Should be combined with SQL filter plugin or filter function to define alert criterias.
[@sematext/logagent-nodejs-monitor](https://www.npmjs.com/package/@sematext/logagent-nodejs-monitor)	other	Monitors server and nodejs metrics of the Logagent process using spm-agent-nodejs

For Developers: How Logagent plugins work

- Logagent checks the configuration file for properties with a "module" key for the nodejs module name. External plugins need to be installed via npm.
- Plugins are initialized with the Logagent configuration (from command line arguments + configuration file) and the event emitter for Logagent. Plugins should provide a start and stop method.
- Input plugins read data from a data source and emit events to the Logagent event emitter. These events have the identifier "data.raw" and 2 parameters:
- data - data read from a data source
- context - an object with meta data e.g. {sourceName: '/var/log/httpd/access.log'} The "context" helps other plugins to process the data correctly, e.g. to handle multiple open files.
- Output plugins listen to "data.parsed" events and store or forward the data to the target.

Examples

Example Input Plugin (TCP Input)

This example implements a plugin to receive data via TCP socket with a configurable rate limit.

The plugin config file:

```
# Global options
input:
  tcp:
    module: input-tcp
    port: 45900
    bindAddress: 0.0.0.0
    sourceName: tcpTest
output:
  # print parsed logs in YAML format to stdout
  stdout: yaml
```

Node.js source code:

```
'use strict'
var split = require('split2')
var net = require('net')
var safeStringify = require('fast-safe-stringify')

/**
 * Constructor called by logagent, when the config file contains this entry:
 * input
 *  tcp:
 *    module: megastef/logagent-input-tcp
 *    port: 4545
 *    bindAddress: 0.0.0.0
 *
 * @config cli arguments and config.configFile entries
 * @eventEmitter logent eventEmitter object
 */
function InputTCP (config, eventEmitter) {
  this.config = config.configFile.input.tcp
 this.config.maxInputRate = config.configFile.input.tcp.maxInputRate || config.maxInputRate
  this.eventEmitter = eventEmitter
}
module.exports = InputTCP
/**
 * Plugin start function, called after constructor
 *
 */
InputTCP.prototype.start = function () {
  if (!this.started) {
    this.createServer()
    this.started = true
  }
}

/**
 * Plugin stop function, called when logagent terminates
 * we close the server socket here.
 */
InputTCP.prototype.stop = function (cb) {
  this.server.close(cb)
}

InputTCP.prototype.createServer = function () {
  var self = this
  this.server = net.createServer(function (socket) {
    // Context object, the source name is used to identify patterns
   var context = { name: 'input.tcp', sourceName: self.config.sourceName || socket.remoteAddres
   socket.pipe(Throttle(self.config.maxInputRate)).pipe(split()).on('data', function emitLine
      // emit a 'data.raw' event for each line we receive
      self.eventEmitter.emit('data.raw', data, context)
      if (self.config.debug) {
        console.log(data, context)
      }
    }).on('error', console.error)
  /*
  // We could return parsed objects to the client
  // Logagent will emit "data.parsed" events
  self.eventEmitter.on('data.parsed', function (data, aContext) {
    socket.write(safeStringify(data) + '\n')
  })
  */
  })
  var port = this.config.port || 4545
  var address = this.config.bindAddress || '0. 0.0.0'
  this.server.listen(port, address)
  console.log('listening to ' + address + ':' + port)
}

// helper  to throttle bandwidth
var StreamThrottle = require('stream-throttle').Throttle
function Throttle (maxRate) {
  var inputRate = maxRate || 1024 * 1024 * 100
  var chunkSize = inputRate / 10
  if (chunkSize < 1) {
    chunkSize = 1
  }
  return new StreamThrottle({
    chunksize: chunkSize,
    rate: inputRate || 1024 * 1024 * 100
  })
}
```

Example Output Plugin (stdout)

```
'use strict'
var prettyjson = require('prettyjson')
var safeStringify = require('fast-safe-stringify')
function OutputStdout (config, eventEmitter) {
  this.config = config
  this.eventEmitter = eventEmitter
}

OutputStdout.prototype.eventHandler = function (data, context) {
  if (this.config.suppress) {
    return
  }
  if (this.config.pretty) {
    console.log(JSON.stringify(data, null, '\t'))
  } else if (this.config.yaml) {
    console.log(prettyjson.render(data, {noColor: false}) + '\n')
  } else {
    console.log(safeStringify(data))
  }
}

OutputStdout.prototype.start = function () {
 this.eventEmitter.on('data.parsed', this.eventHandler.bind(this))
}

OutputStdout.prototype.stop = function (cb) {
 this.eventEmitter.removeListener('data.parsed', this.eventHandler)
  cb()
}

module.exports = OutputStdout
```