title: Logagent plugins description: Logagent features modular logging architecture framework where each input or output module is implemented as a plugin, and loaded on demand as declared in the configuration file. It is used with elasticsearch, syslog, gelf, cassandra, mysql, postgres, mqtt, log anonymization, apache kafka, and more

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
docker-logs	input	Collection of Docker container logs
input-kubernetes-events	input	Collection of Kubernetes events
input-kubernetes-audit	input	Receive Kubernetes audit logs via http / webhook
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
input-journald-upload	input	Receive data via HTTP from the systemd-journal-upload.service
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

Plugin	Type	Description
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
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)
input-azure-eventhub	input	Receives events from Azure Event Hubs
grep	Processor / input filter	Filters text with regular expressions before parsing
input-filter-k8s-containerd	Processor / input filter	Parsing cri-o log format and add Kubernetes context to container logs
sql	Processor / output filter	Transforms and aggregates parsed messages with SQL statements
aes-encrypt-fields	Processor / output filter	Encrypt field values with AES before any output happens
hash-fields	Processor / output filter	Hashing of field values before any output happens
ip-truncate-fields	Processor / output filter	Replaces the last block of IPv4 and IPv6 address fields with "0" to anonymize IP addresses
remove-fields	Processor / output filter	Removes fields before any output happens

Plugin	Type	Description
drop-events	Processor / output filter	Drop events via value filters for fields
docker-enrichment	Processor / output filter	Metadata enrichment for docker logs, including log routing options
kubernetes-enrichment	Processor / output filter	Metadata enrichment for pod logs, including log routing options
geoip	Processor / output filter	Add Geo-IP information to logs
stdout (default)	output	Prints parsed messages to standard output. Supported formats: YAML, JSON, Line delimited JSON (default).
elasticsearch	output	Stores parsed messages in Elasticsearch
output-gelf	output	Sends data via GELF protocol
output-mqtt	output	Sends messages via MQTT protocol
output-influxdb	output	Stores parsed messages in InfluxDb
output-aws-elasticsearch	output	Stores parsed messages in Amazon Elasticsearch
output-files	output	Stores parsed messages files. Log rotation and dynamic file name generation are supported.
output-clickhouse	output	Sends parsed messages to Yandex ClickHouse DB
logagent-output-kafka	output	Sends parsed messages to Apache Kafka topics. 3rd party module. 3rd party module.
output-http	output	Sends parsed messages via HTTP or HTTPS
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

Find plugins on npm

Developers of 3rd party plugins publish logagent plugins in the npm registry. Simply search for logagent to discover more plugins.

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 - a string containing a text line, 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. In some cases, input plugins create strcutured data, and it makes no sense to process the data with text bases input-filters and Logagent parser. Input plugins can emit a `data.object` event, and only output-filters and output plugins will process such events with the following parameters:
  - data - a JavaScript object e.g. `{message: 'hello', severity: 'info'}`
  - context - an object with meta data e.g. {sourceName: ‘/var/log/httpd/access.log’}
- 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.remoteAc
    socket.pipe(Throttle(self.config.maxInputRate)).pipe(split()).on('data', function emitLi
      // 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
```