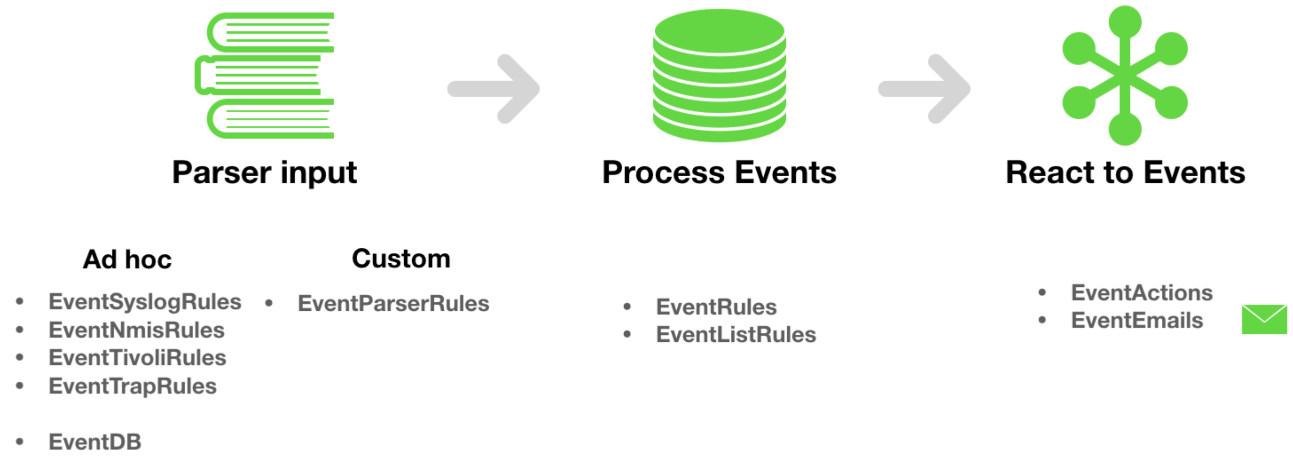
Configuring opEvents



Parser input

Ad hoc

- EventSyslogRules
- EventNmisRules
- EventTivoliRules
- EventTrapRules

- EventDB

Custom

- EventParserRules

Process Events

- EventRules
- EventListRules

React to Events

- EventActions
- EventEmails

opEvents is flexible and configurable. In the above schema you can see all the involved files to configure opEvents.

Parser Input

opEvents can parse a variety of inputs in a wide variety of formats, and the way opEvents has to understand them is with parsers. opEvents has different parsers for natively-supported log formats, to filter and normalise the data. It uses some files for each format file supported:

- **EventSyslogRules**
- **EventNmisRules**
- **EventTivoliRules**
- **EventTrapRules**

It also can process events from a database. All this configuration is in the file **EventDB.nmis**.

If some other syslog formats are required, we can write our own parser in **EventParserRules.nmis** file. It defines regular expressions to extract data and the variables to assign them to. If a parser is not enough, we can also use a parser plugin and generate the event using perl code.

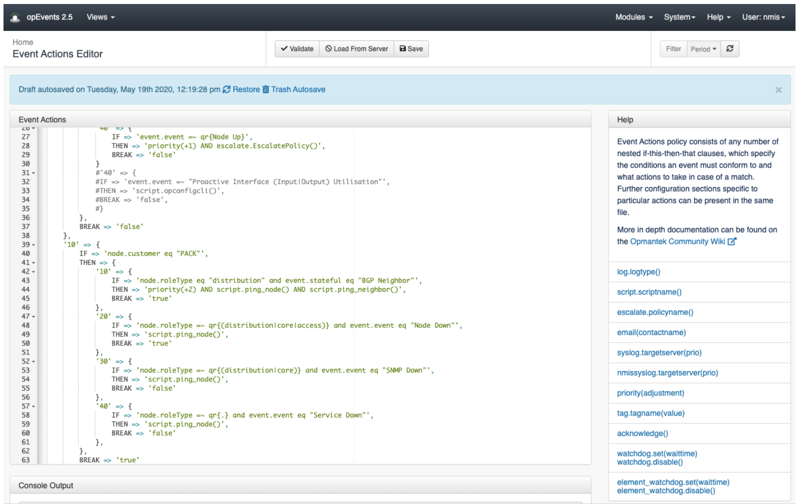
Process Events

It is possible to specify rules to reduce voluminous inputs down to the relevant details, suppress duplicates or correlate events. This is the purpose of **Event Rules.nmis** file. **EventListRules.nmis** also helps in controller the event in the system by blacklisting of whitelisting the events.

React to Events

Once the event is created, **EventActions.nmis** file specifies how to react to events. The file consist in a set of rules with flexibility to being adapted to a wide variety of use cases.

This file can be edited used the opEvents editor, http://host/en/omk/opEvents/config/actions:



Draft autosaved on Tuesday, May 19th 2020, 12:19:28 pm Restore Trash Autosave

```
                    IF => 'event.event =~ qr{Node Up}',
                    THEN => 'priority(+1) AND escalate.EscalatePolicy()',
                    BREAK => 'false'
                }
                #'40' => {
                #IF => 'event.event =~ "Proactive Interface (Input|Output) Utilisation"',
                #THEN => 'script.opconfigcli()',
                #BREAK => 'false',
                #}
            },
            BREAK => 'false'
        },
        '10' => {
            IF => 'node.customer eq "PACK"',
            THEN => {
                '10' => {
                    IF => 'node.roleType eq "distribution" and event.stateful eq "BGP Neighbor"',
                    THEN => 'priority(+2) AND script.ping_node() AND script.ping_neighbor()',
                    BREAK => 'true'
                },
                '20' => {
                    IF => 'node.roleType =~ qr{(distribution|core|access)} and event.event eq "Node Down"',
                    THEN => 'script.ping_node()',
                    BREAK => 'true'
                },
                '30' => {
                    IF => 'node.roleType =~ qr{(distribution|core)} and event.event eq "SNMP Down"',
                    THEN => 'script.ping_node()',
                    BREAK => 'false'
                },
                '40' => {
                    IF => 'node.roleType =~ qr{.} and event.event eq "Service Down"',
                    THEN => 'script.ping_node()',
                    BREAK => 'false'
                },
            },
            BREAK => 'true'
```

Event Actions policy consists of any number of nested if-this-then-that clauses, which specify the conditions an event must conform to and what actions to take in case of a match. Further configuration sections specific to particular actions can be present in the same file.

More in depth documentation can be found on the Opmantek Community Wiki

- log.logtype()
- script.scriptname()
- escalate.policyname()
- email(contactname)
- syslog.targetserver(prio)
- nmissyslog.targetserver(prio)
- priority(adjustment)
- tag.tagname(value)
- acknowledge()
- watchdog.set(waittime)
  watchdog.disable()
- element_watchdog.set(waittime)
  element_watchdog.disable()

One of the available actions is to notify using email, and different templates can be defined in **EventEmails.nmis** file.