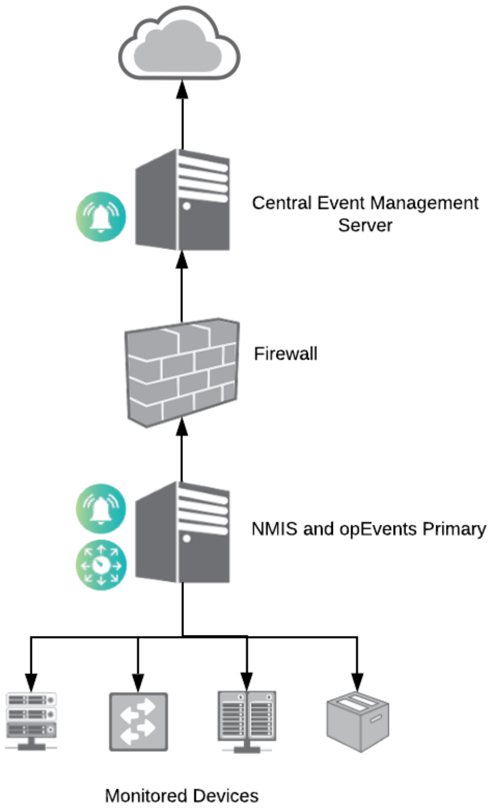
opEvents - Solution Guide - Sending Events to a Central Event Management Server

Feature Overview

opEvents has the ability to forward events based on filters to another server running opEvents (or other service desk systems, like Servicenow and Connectwise).   This is used as part of the Opmantek Multi-server architecture with distributed pollers and centralised primary servers, it is also extremely useful in situations such as an opEvents instance not being reachable across the internet, but another central instance of opEvents is reachable.

Events are forwarded using http or https which is setup separately on your server, independent of opEvents. A typical Apache SSL configuration works just fine.



Related Wiki Articles

Configuring opEvents

opEvents Configuration Settings

Create remote event

Configuring SSL on apache for NMIS and OMK

PreConditions

We're assuming you already have a poller running NMIS and opEvents along with another instance of opEvents setup, configured and working.

This functionality is intended for users with advanced knowledge of NMIS and opEvents.

If you need HTTPS security, this should already have been configured (see link above).

The details here are related to opEvents 4.x and higher which integrates with NMIS9.

References

- NMIS - Common Node Properties

- NMIS - NMIS Event List
- NMIS - Description of the NMIS Event Log
- opEvents - opEvents Normalised Event Properties
- opEvents - Event Actions and Escalation
- opEvents - opEvents input sources

Configuration

To enable this functionality, you must edit the opEvents "Event Actions" JSON file. We recommend using the web GUI to do this as there is a very handy "Validate" button that can be used to ensure your configuration changes are indeed valid JSON and won't break opEvents. If you must, you can also edit the JSON file directly at /usr/local/omk/conf/EventActions.json file. Beware that making changes that result in an invalid JSON file will result in your actions not functioning as intended.

Validating a JSON file on the command line can be done using the command:

```
python -mjson.tool /usr/local/omk/conf/EventActions.json
```

opCommon.json

Certain node properties must be defined in opCommon.json in order for opEvents to make use of them.

Disable Auto Acknowledge of Up Events

The opEvents engine will normally auto acknowledge up events as they are clearing a down event, to make this solution work you will need to disable this feature.

Verify and if necessary modify the configuration file /usr/local/omk/conf/opCommon.json and change the setting opevents_auto_acknowledge_up to be false.

Setup Macros

```
    "macro" : {
       "authority" : "YOUR_SERVER_NAME"
    },
```

Copy Node Properties

opEvents supports the ability to copy node properties to the event so they persist with the event and can be used. In earlier versions for example, *group* and *location* were at the root level. These are now under *configuration* so should be defined as below.

```
       "opevents_event_copy_node_properties" : [
          "configuration.group",
          "configuration.location"
       ],
```

Not like below.

```
       "opevents_event_copy_node_properties" : [
          "group",
          "location"
       ],
```

This is mostly of relevance to customers who have upgraded from opEvents 2.x and if you already have opEvents 4.x running successfully, it likely is simply something to double check.

Verify the opCommon.json file is valid JSON

Validating a JSON file on the command line can be done using the command:

```
python -mjson.tool /usr/local/omk/conf/opCommon.json
```

Restart the opEvents Daemon

Once you have made these changes you will need to restart the opEvents Daemon (opeventsd).

```
sudo service opeventsd restart
```

EventRules.json

Also check for similar naming in conf/EventRules.json, ie: *node.serviceStatus* is **incorrect**. It should be *node.configuration.serviceStatus*.

Again, this is mostly for those users setting up opEvents after upgrading from an earlier version.

Just something to double check and be aware of.

Event Policy

There are a few sections in EventActions.json, but the one we're concerned with first is the **policy** section. Entries in a given section are numbered for determining the order in which they are processed. Usually we make forwarding the last entry (the entry with the highest number).

Here is the full policy together, the following will explain how it flows.

```
    "policy": {
         "10":
         {
             "BREAK": "false",
             "IF": "event.any",
             "THEN":
             {
                 "10":
                 {
                     "IF": "event.state =~ /up|closed/",
                     "THEN": "tag.escalateToCentral(FALSE) and tag.sendToCentral(TRUE)",
                     "BREAK": "true"
                 },
                 "20":
                 {
                     "IF": "node.configuration.serviceStatus eq 'Production'",
                     "THEN": "tag.escalateToCentral(TRUE) and tag.sendToCentral(FALSE)",
                     "BREAK": "true"
                 }
             }
         },
         "20":
         {
             "BREAK": "false",
             "IF": "event.any",
             "THEN":
             {
                 "10":
                 {
                     "IF": "event.tag_escalateToCentral eq 'TRUE'",
                     "THEN": "escalate.central()",
                     "BREAK": "true"
                 },
                 "20":
                 {
                     "IF": "event.tag_sendToCentral eq 'TRUE'",
                     "THEN": "script.sendToCentral()",
                     "BREAK": "true"
                 }
             }
         }
     },
```

Check All Events

The first policy block says process all events, e.g. IF event.any

First we want to check *all* events and filter them in sub-sections. This just makes things easier for a human to read and understand. So our IF section at the first level policy.10 is a simple event.any. For any event, the THEN section is again numbered for order, that will then be processed.

Tag any up events

Next we need to decide which events we would like to send to the central instance. We should send all 'Node Up' or 'Closed' events to the second instance, regardless of the node so it cancels out any escalation. It does not matter that no 'down' event might not have been sent previously. We set our event tags to be `escalateToCentral = FALSE and sendToCentral = TRUE`

```
                "10":
                {
                    "IF": "event.state =~ /up|closed/",
                    "THEN": "tag.escalateToCentral(FALSE) and tag.sendToCentral(TRUE)",
                    "BREAK": "true"
                },
```

You may also have noticed the 'up|closed' entry. These entries take perl regular expressions, so in this case if up or closed is present in the event/state, it will match and trigger the THEN.

Tag any Production node events to escalate important events

We're going to escalate (for the purposes of this example) all events with a node that has a serviceStatus of 'Production', this could be some other criteria like the criticality of the event, e.g. "event.priority > 3".

These events will go into a second list, inside the top level, we set our event tags to be `escalateToCentral = TRUE and sendToCentral = FALSE`

```
                "20":
                {
                    "IF": "node.configuration.serviceStatus eq 'Production'",
                    "THEN": "tag.escalateToCentral(TRUE) and tag.sendToCentral(FALSE)",
                    "BREAK": "true"
                }
```

Now you may have noticed we have two items to action - escalateToMaster and sendToMaster. escalateToMaster sends an escalation where-as sendToMaster doesn't do anything other than simply send the event. No escalation required.

Check tags

Next we need to check the tags and if required, escalate or send it to the second instance.

```
        "20":
        {
            "BREAK": "false",
            "IF": "event.any",
            "THEN":
            {
                "10":
                {
                    "IF": "event.tag_escalateToCentral eq 'TRUE'",
                    "THEN": "escalate.central()",
                    "BREAK": "true"
                },
                "20":
                {
                    "IF": "event.tag_sendToCentral eq 'TRUE'",
                    "THEN": "script.sendToCentral()",
                    "BREAK": "true"
                }
            }
        }
```

Escalate

As you can see, we reference two different routines in the THEN sections for policy.20.10 and policy.20.20

For the THEN action escalate.central in policy.20.10, we are simply checking if the event has a priority high enough to send and our hours of operation correspond. If it does, send it. This is defined in the **escalate** section of JSON. The 360 refers to the escalation to take place after 360 seconds has passed. Implementing like this will prevent flaps and noisy events from going to the central server, this greatly reduces the noise of transient events in the environment, you can use a lower time here if you want them faster. You can also perform other action here, so many options.

```
"escalate": {
    "central": {
      "name": "central",
      "IF": {
        "priority": ">= 1",
        "days": "Monday,Tuesday,Wednesday,Thursday,Friday,Saturday,Sunday",
        "begin": "0:00",
        "end": "24:00",
        },
        "360": "script.sendToCentral()",
    }
}
```

Script

Any 'script.X' THEN items are defined in the **script** section of EventActions.json - hey, big surprise, I know 🙂

For policy.20.20, we don't bother testing for a priority, we just send them. This is because events such as 'up' or 'closed' have low priority - but obviously we want to send them to cancel the 'down' type events.

So our script blocks looks as below.

```
"script": {
    "sendToCentral": {
        "arguments": [
            "-s",
            "https://your_central_opevents_server/omk",
            "-u",
            "your_opevents_user",
            "-p",
            "your_opevents_password",
            "authority=macro.authority",
            "location=https://your_local_opevents_server/omk/opEvents/events/event._id/event_context",
            "node=event.node",
            "event=event.event",
            "details=event.details",
            "time=event.time",
            "date=event.date",
            "element=event.element",
            "interface_description=event.interface_description",
            "type=event.type",
            "priority=event.priority",
            "level=event.level",
            "state=event.state",
            "stateful=event.stateful"
        ],
        "exec": "/usr/local/omk/bin/create_remote_event.exe",
        "output": "save",
        "stderr": "save",
        "exitcode": "save"
    }
}
```

Secure Wrapper

If you wish to not include your username and password in the Event Actions file, you can use a wrapper script.

Create a script e.g. /usr/local/omk/bin/create_remote_event.sh and make the ownership and permissions for root only.

```
sudo touch /usr/local/omk/bin/create_remote_event.sh
sudo chown root:root /usr/local/omk/bin/create_remote_event.sh
sudo chmod 700 /usr/local/omk/bin/create_remote_event.sh
```

The contents of this script would be:

```
#/usr/bin/env bash

# username and password below, pass through all other arguments.
/usr/local/omk/bin/create_remote_event.pl -u "your_opevents_user" -p "your_opevents_password" $@
```

Note: If your user's password has special characters (such as $) you may need to escape them in the script. For example a password of "opeventsPas$w0rd" would be written as:

```
-p "opeventspas\$w0rd"
```

Event Actions Script would be updated as such, leaving out the username and password.

```
"script": {
    "sendToCentral": {
        "arguments": [
            "-s",
                        "https://your_central_opevents_server/omk",
            -- snip same as above in here --
            "stateful=event.stateful"
        ],
        "exec": "/usr/local/omk/bin/create_remote_event.sh",
        "output": "save",
        "stderr": "save",
        "exitcode": "save"
    }
}
```

Handling a High Volume of Events Being Sent to Primary Event Server

A faster CLI tool was developed written in GO, which executes in less than half the time, you can find details here Create remote event Fast create remote event

You can download the latest version from the link in the page above, copy the binary into /usr/local/omk/bin, and rename the file or make a symbolic link so you can deal with shorter name.

```
ln -s fast-remote-event-1.x.x-LinuxX86_64.bin fast-remote-event
```

Then update your Event actions or your shell wrapper /usr/local/omk/bin/create_remote_event.sh to use this binary instead of create_remote_event.pl or create_remote_event.exe

e.g.

```
"exec": "/usr/local/omk/bin/fast-remote-event",
```

To have it return the resulting event id from the primary, include "-q=0" in the arguments.

Finishing Up

And that's it!

You have selected events to be forwarded and tagged them. Tested those tags for actions and depending on the tag and priority, forwarded it to your central instance.

So for future versions, you might define an event on a node with serviceStatus of "Testing" and choose only to forward it during office hours (for example). Give it a different tag, check that tag and use escalate with an additional item (say "testing_office_hours_central"). The ways to configure opEvents really are limited only by your imagination.

Hopefully this article has given you some ideas and pointed you in the right direction.