Leveraging opEvents and opConfig to Automate Operational Changes

Purpose

This article will provide an example of opEvents triggering opConfig to make an operational change.

Use Case

If an interface starts registering input errors, we want to automatically shift traffic off the circuit in order to maintain transmission quality.

Related Pages

Before attempting this configuration the admin should be familiar with the following wiki articles.

- Basic and Advanced Thresholds in NMIS8
- Event Actions and Escalation
- Automating Configuration Changes with opConfig

Sequence Overview

- NMIS polls a router with an SNMP query.
- The router returns a 'interface input error' counter value that has increased; thus triggering a pre-defined threshold.
- NMIS generates a 'input error' alert that is processed by opEvents.
- opEvents has a predefined action rule matching on node, interface and input errors. This action will will fire a opConfig 'Configuration Set'.
- The associated opConfig Configuration Set will increase the OSPF cost on the associated interfaces, thereby causing the router to select another path if available.

Configuration

NMIS

Be default NMIS has the necessary configuration for alerting on input errors. This is done with the NMIS thresholding system. The thresholds for the different alerting levels may be adjusted in the appropriate section of /usr/local/nmis8/models/Common-threshold.nmis. The levels below represent a percentage of input error packets as compared to good packets.

/usr/local/nmis8/models/Common-threshold.nmis

```
      'pkt_errors_in' => {
        'item' => 'ifInErrorsProc',
        'event' => 'Proactive Interface Error Input Packets',
        'title' => "Input Error Packets",
        'unit' => 'packets',
        'select' => {
          'default' => {
            'value' => {
              'fatal' => '0.5',
              'critical' => '0.25',
              'major' => '0.1',
              'minor' => '0.05',
              'warning' => '0.02',
            }
          }
        }
      },
```

opEvents

By default opEvents processes the NMIS event log. All event will be evaluated by /usr/local/omk/conf/EventActions.nmis. If an event matches a rule the appropriate actions will be taken. EventActions.nmis is also were we define the scripts that opEvents can fire. The first step is to define the scripts that will shift traffic off a link that's running input errors. Since we want to shift all traffic off this link we will need to run scrips for both ends of the circuit. Notice the reference to a configset; these will be defined in the opConfig section.

> Changes to /usr/local/omk/conf/EventActions.nmis require that the omkd service be restarted.

/usr/local/omk/conf/EventActions.nmis

```
         'script' => {
                'bnelab_p2_fa0_0_route_not' => {
                        arguments => 'act=push_configset name=bnelab-p2_fa0-0_route_not at=now+1minute
nodes=bnelab-p2',
                        exec => '/usr/local/omk/bin/opconfig-cli.exe',
                        output => 'save'
                },
                'bnelab_rr1_e1_2_route_not' => {
                        arguments => 'act=push_configset name=bnelab-rr1_e1-2_route_not at=now+1minute
nodes=bnelab-rr1',
                        exec => '/usr/local/omk/bin/opconfig-cli.exe',
                        output => 'save'
                },
        },
```

With the scripts defined let's add the matching rule to the policy section.

/usr/local/omk/conf/EventActions.nmis

```
         'policy' => {
                '10' => {
                        IF => 'event.any',
                        THEN => {
                                '10' => {
                                        IF => 'event.node eq "bnelab-rr1" and event.element eq "Ethernet1/2"
and event.event eq "Proactive Interface Error Input Packets"',
                                        THEN => 'script.bnelab_rr1_e1_2_route_not() and script.
bnelab_p2_fa0_0_route_not()',
                                        BREAK => 'false'
                                },
```

opConfig

The next step is to define the config sets. Config sets are opConfig talk for the configuration commands you'd like ran on the router. Because this step is complicated, yet very repeatable I've supplied this script: writeConfigSet.sh. Run the script and it will prompt you for the commands you want ran on the router and install the config set in opConfig. In order to verify config sets use the opConfig GUI, from the top menu bar select views, then Configuration Set Overview.

Here is what our example config set looks like.

```
{
  "name": "bnelab-rr1_e1-2_route_not",
  "commands": [
    "int e1/2",
    "ip ospf cost 9999",
    "exit"
  ],
  "post-commands": ["write mem"]
}
```

Testing and Verification

Generate Input Errors

There are several different kinds of input errors but the easiest kind to create in a lab environment are giants. This is done by having mismatched MTU's on either side of the same circuit; then sending packets that are too big from the side with the larger mtu.

In this example we'll send giants from bnelab-p2 like so:

```
bnelab-p2#ping 10.248.2.6 size 1530 repeat 1000 timeout 0
```

On benlab-rr1 we"ll see the error counters increment.

```
bnelab-rr1#show int e1/2 | inc error|giants
     0 runts, 4073 giants, 0 throttles
     4073 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored
     0 output errors, 0 collisions, 1 interface resets
```

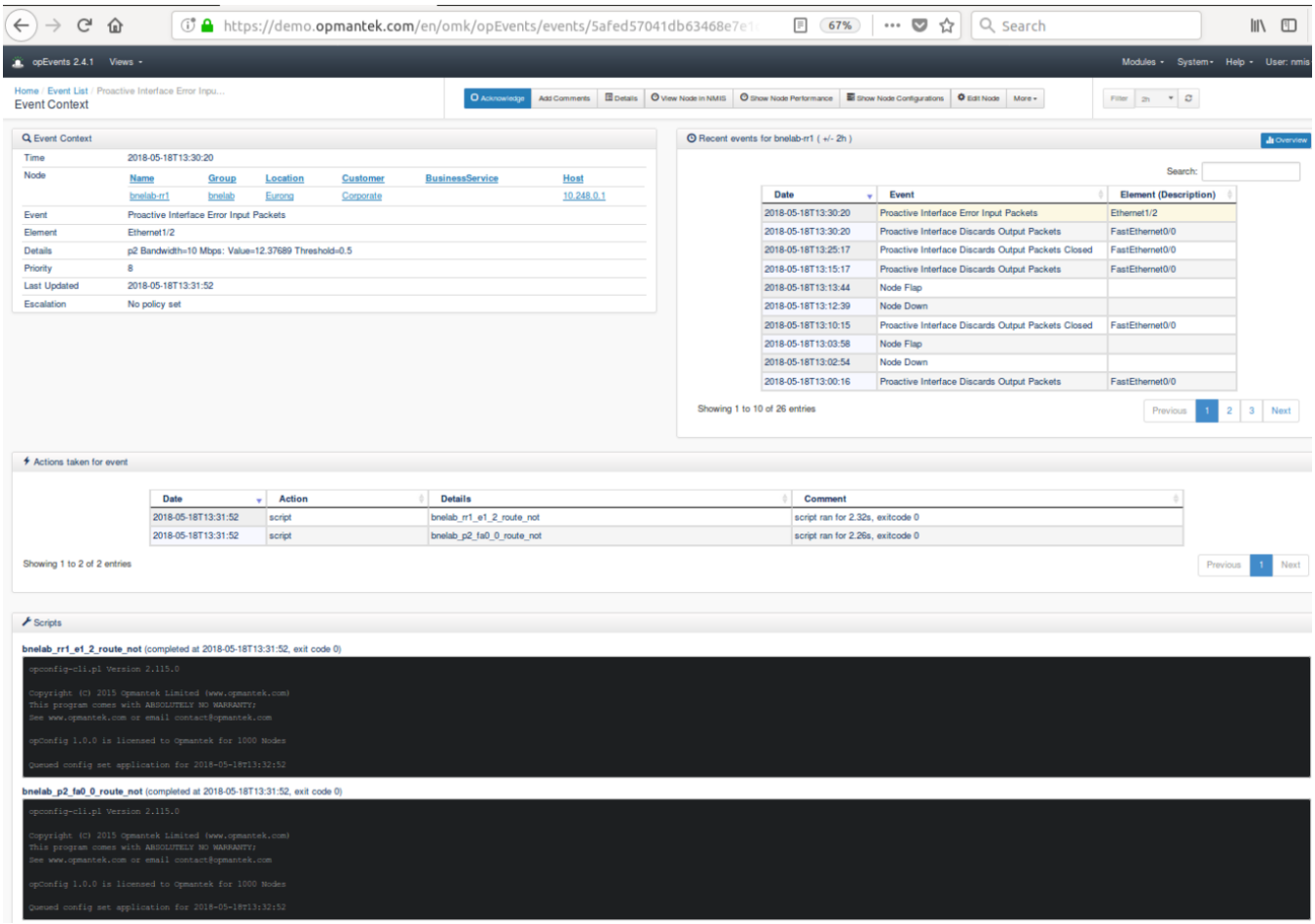
Observe Input Error Event in NMIS

After the next NMIS collect cycle for bnelab-rr1 we should see an event similar to the following:

blocked URL 18-May-2018 13:30:20 bnelab-rr1 Proactive Interface Error Input Packets Fatal Ethernet1/2 p2 Bandwidth=10 Mbps: Value=12.37689 Threshold=0.5

Observe Input Error event in opEvents

Next find the input error event in opEvents.



Notice the actions taken and scripts sections. Based on this we know the script was successful and what time the config change has been scheduled for.

Confirm Successful Configuration Push in opConfig

From the opConfig GUI top menu bar select Views, Configuration Change History. Find and select the config push that relates to our test event.