Deduplication and storm control in opEvents

opEvents provides two mechanisms to handle repeated event occurrences in a practical fashion, namely stateful event deduplication and programmable event suppression.

Stateful Deduplication and Flaps

All events that are related to stateful entities (e.g. a node which can be in state up or down, an interface etc.) are automatically checked against the recent history of events and the known previous state of this entity. If the new event reports the same state as the already known one, then the new event is suppressed completely: no event record is created (except for raw logging, if that is enabled).

In practice this means that when there are multiple reports of a "Node down" around the same time, then only the first event will show up on the opEvents dashboard. This type of deduplication is essential for dealing with event storms; it is therefore always active and non-adjustable.

Related to that is the concept of a Flap, which in opEvents is defined as a transition sequence from state up to down and back up within a short time frame. opEvents uses the configuration option `state_flap_window` to define this window, by default 90 seconds.

In a flap situation the up event is marked as flap (by setting the `flap` property to 1) and as associated with the down event (using the `eventids` or `stateful_eventids` property). In versions up to 2.2.1, the up event's name is always changed to "`<state entity> Flap`"; newer versions of opEvents support the config option `opevents_flap_name`, which lets you specify a template (which can contain `node.X`, `event.Y` and `macro.Z` placeholders, e.g. "`event.event for event.stateful - Flap`").

The interaction between down and up events in a flap situation can be fine-tuned using the configuration option `opevents_no_action_on_flap` (default: "true").

- When set to "true" opEvents will automatically acknowledge the related down event and set the down event's action_required to false. This causes any actions defined in policies for the down event to be stopped (including escalation actions). The down event is thus closed and disposed of on receiving the up event.
- On the other hand, if `opevents_no_action_on_flap` is false, then the down event is not modified in any way and remains open when a flap is detected; it is thus trackable independent of the up event.

Involved Event Properties

This section outlines certain internal details, mostly relevant If you are using a custom parser to feed opEvents.

Stateful event handling relies on three core event properties: `stateful`, `element` and `state`.

- The `stateful` property indicates the type of state source, and is a free-form string.
  For example, if the event is related to an interface, `stateful` should be set to "Interface"; if it's about a service, the value "Service" would be most appropriate, etc.
  You may use any state source type you want in your parser rules, but avoid overloading already existing ones like "Node" and "Interface".
- The `element` property indicates which (of potentially many) state sources the event relates to.
  For state type interface, a unique interface identifier should be used (i.e. the ifDescr).
  Like above your parser rules may capture or set the element to anything you desire, as long as the combination of node name, `stateful` and `element` is a suitably unique identifier for the particular stateful thing you're trying to track.
- The `state` property indicates whether the observed state is "good" or "bad".
  opEvents treats the values `up`, `ok`, `good`, `normal` or `closed` as "good", anything else as "bad".
  This comparison is made case-insensitively, i.e. "`Good`" will work just as well as "`OK`".

For state tracking opEvents then combines the node name and the values of `stateful` and `element` into a lookup key, and associates that key with the `state` value.
Any repeat events with the same lookup key and the same state value are ignored.

Stateful Deduplication, Forwarded Events and Reorder Protection

If you use an Event Action or Escalation Policy with `create_remote_events` to forward events to another opEvents server elsewhere, then you might occasionally find that such forwarded events arrive out of order, i.e. an earlier 'down' event might be received **after** the later 'up' event. This can happen because of network congestion, action processing on the sending side being asynchronous and subject to process limits and similar reasons.

Out of order reception of stateful events can cause state desynchronisation at the receiving server, as the up event would be processed first and thus be deduplicated and discarded, while the down event later on causes a transition to state down which isn't cleared.

opEvents versions 2.4.2 and newer provide a reorder protection mechanism to handle such out of order situations better - which comes at the cost of temporarily delaying the processing of some forwarded stateful events.

To enable reorder protection, two steps need to be taken:

- you need to set the configuration property `state_reorder_window` to a positive number (e.g. 30) on the receiving server,
- and you must make sure that your forwarded events do carry an `authority` property, to denote the event as originating from a remote authoritative source.

If both of these conditions are met, opEvents on the receiving server will temporarily postpone processing of a forwarded stateful event, if the event would be discarded by stateful deduplication.
This allows earlier but externally delayed related events to enter the processing queue in the correct sequence, if any such do arrive within the configured time window after the out-of-order postponed event.

If a state-changing remote event does arrive within the time window configured by `state_reorder_window`, then the correct sequencing of transitions is restored and processing of postponed events resumes immediately. Otherwise, processing resumes after the time window elapses.

The `state_reorder_window` should not be set too large as it causes undesirable event processing delays; a value of 10 to 30 seconds should suffice in most environments.

Programmable Suppression

To provide fine-grained control of how to handle repeated events of any kind, opEvents also supports programmable event suppression. Using this facility the administrator can define flexible rules for when to suppress repeat events, based on the recent event history and some further refinement criteria. Please note, however, that programmable suppression is available only for classes or groups of events and cannot be enabled specifically for a single node only.

The configuration file `EventRules.nmis` can contain any number of user-defined event synthesis and suppression directives given in a simple, almost self-explanatory format.

A suppression rule consists of:

- a rule `name`, which is for display purposes only when suppression is concerned,
- a list of `events` (more precisely, their names), which are the events to consider for suppression,
- an optional list of `groupby` clauses, which define whether thresholds are to be interpreted globally for all named events, or separately within smaller groups,
- a `window` parameter, which defines the time window to examine,
- optional `delayedaction` and `autoacknowledge` parameters (in opEvents 2.0.4 and newer),
- and a `suppress` clause with a min and/or a max occurrence parameter.

Note that this configuration file can also contain rules for Event Synthesis, which differ just slightly (they have a `count` parameter and no `suppress` clause).

Here is an example rule:

```
'5' => {
         name=>"suppressing repeats", # name not relevant for suppression
         events=>['Node Configuration Change'],
        groupby=>['node.name'],
        window=>120,
        suppress=>{min=>2, max=>8},
},
```

All such rules are applied independently to an event (until one indicates suppression or the end of the rule list is reached).

All named events that are listed in a suppression rule and which have occurred in the preceding `window` seconds are checked and counted together. Listing multiple events in one rule will lump them together as far as the occurrence counting is concerned. These recent events will then be apportioned to groups if `groupby` is used, and then the event count is compared to the min/max occurrence parameters. If the count is above `min` and below `max`, then the new event is marked as a duplicate (of the oldest event that was counted) and has its `action_checked` property set to 1 which prevents any future policy actions (e.g. escalations) from being executed; the event is nevertheless shown in the opEvents GUI.

If the suppression clause contains no `min` parameter, then a minimum of 1 is assumed. If no `max` is present, then infinity is used. Both `min` and `max` include the current event, so a `min` of 2 will suppress the first and further repeats.

Delaying and Closing of Trigger Events

In opEvents 2.0.4 and newer, suppression rules can optionally specify a number for the `delayedaction` property, to delay all policy action processing for potential trigger events. If the criteria for suppression are met within the delay period, then all action processing will be aborted and skipped for these suppressed events. If the `autoacknowledge` property is also set, then the suppression includes not just aborting action processing but also marking the event as acknowledged.

Grouping

If no `groupby` clause is present, then the set of matching events is counted directly, which may be too generic for many common scenarios. For example suppressing events for a particular customer or service group wouldn't be possible. Grouping solves this problem: the set is split into groups with matching property values and the thresholds are applied to those groups.

The `groupby` clause has the form of a list of `node.X` or `event.Y` property specifications (e.g. `node.customer` or `node.group`), which are used to group events into buckets for counting: only events that share the same values for all the listed grouping properties will be counted together. For example, the `groupby` clause `[ 'node.customer', 'event.priority' ]` would cause this suppression rule to be applied independently for all combinations of customer and event priority. The clause given in the example block above will suppress 2-8 node configuration events *for any individual node* within 120 seconds; without the `groupby` repeat node configuration events would be suppressed regardless of where they happened.

The common node properties are listed here, and the standard event properties are documented on this page.