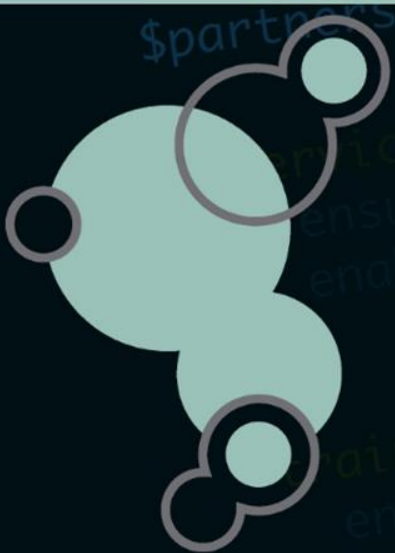


```
# Open-Future
#
# Installs, configures and keeps your open source
# infrastructure up and running.
```

```
class openfuture (
  $consultants = ['you?', 'johan', 'bert', 'patrik', 'johan'],
  $sales       = ['ann'],
  $services    = ['infrastructure', 'consultancy', 'training'],
  $trainings   = ['zabbix', 'bacula', 'puppet', 'linux'],
  $partners    = ['nico', 'danny']
```

NIFI Flow monitoring With Zabbix

How to monitor NIFI flows with Zabbix by using Prometheus LLD, Tags and overrides

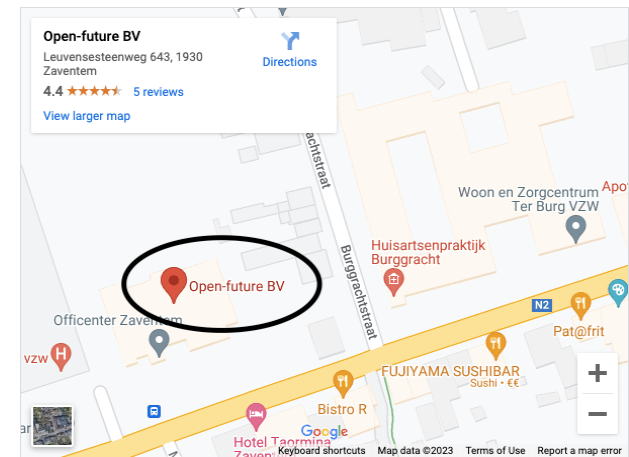


open-future

Author : Patrik Uytterhoeven

Who is Open-Future

- Open-Future was founded in 2009 by Danny and Nico
- We are specialized in open-source solutions.
- We focus on open-source partnerships with vendors but are not limited to .
- We have partnerships with RedHat, Bacula, SEP, Zabbix, ...
- We provide Official Trainings for Bacula, Puppet and Zabbix
- We are one of the oldest Zabbix partners
- We provide trainings in our office, onsite and online



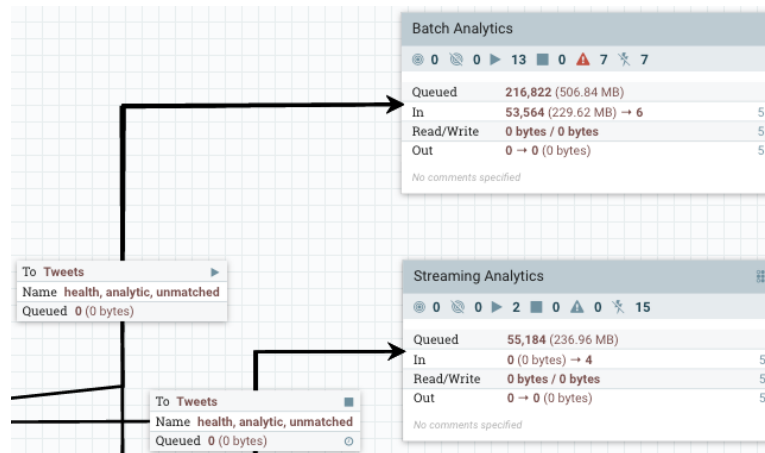
Who is Patrik Uytterhoeven

- Patrik is an open-source consultant working for Open-Future
- Has more than 20y of experience in IT
- Started with HP Unix 11 and RedHat 5
- Has a strong interest in monitoring with Zabbix.
- Is a certified Zabbix trainer since Zabbix 2.2
- Has written several Zabbix books in the past
- Tries to stay up to date when it comes to Zabbix, PostgreSQL, Ansible and Security like SeLinux.

What is NiFi

Apache NiFi is an easy to use, powerful, and reliable system to process and distribute data.

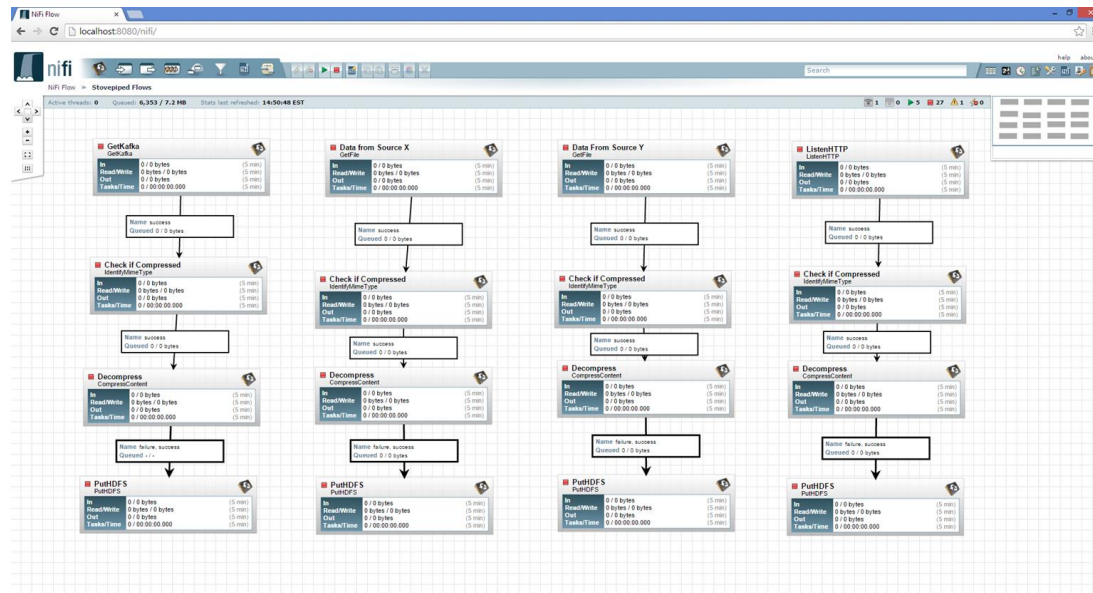
Apache NiFi supports powerful and scalable directed graphs of data routing, transformation, and system mediation logic.



What would we like to monitor

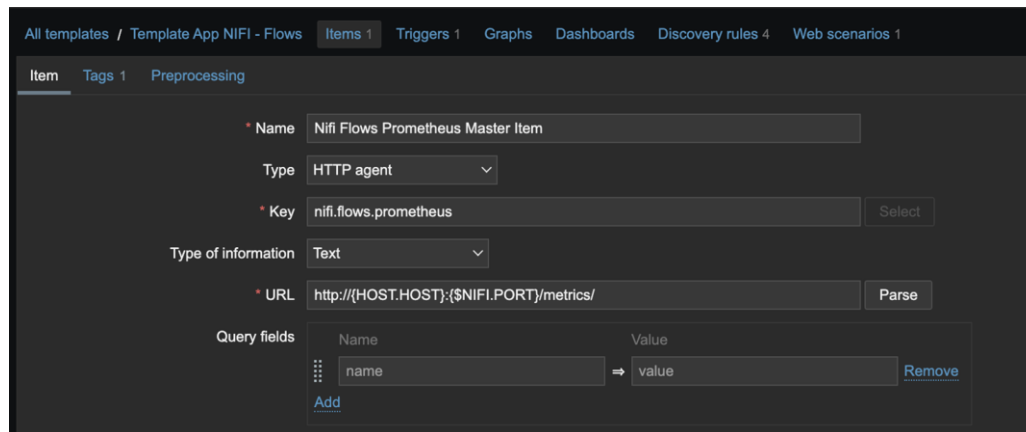
We like to monitor the flows created in NIFI for each database and send messages to the teams only if there is a problem with one of their databases.

NIFI has a plugin for Prometheus that we can use in Zabbix for this.



How did we implement it ?

We have to create a HTTP Agent item in Zabbix to read out the information from Prometheus endpoints.



The screenshot shows the Zabbix web interface for creating a new item. The breadcrumb trail is "All templates / Template App NIFI - Flows". The "Items" tab is selected, showing a count of 1. Below the tabs, the "Item" sub-tab is active, with "Tags" (1) and "Preprocessing" also visible. The configuration form includes the following fields:

- Name:** Nifi Flows Prometheus Master Item
- Type:** HTTP agent (selected from a dropdown)
- Key:** nifi.flows.prometheus (with a "Select" button)
- Type of information:** Text (selected from a dropdown)
- URL:** http://{HOST.HOST}:{\$NIFI.PORT}/metrics/ (with a "Parse" button)
- Query fields:** A table with two columns, "Name" and "Value". It contains one entry: "name" in the Name column and "value" in the Value column. There is an "Add" link below the table and a "Remove" link next to the entry.

How did we implement it ?

Once the master item is finished there is only the LLD to take care of. The LLD rule is dependent on the Master item.

The screenshot shows the Nifi configuration interface for a discovery rule. The breadcrumb trail at the top is: All templates / Template App NIFI - Flows / Discovery list / nifi_amount_items_queued. Below this, there are tabs for Discovery rule, Preprocessing 1, LLD macros 5, Filters, and Overrides 1. The 'Discovery rule' tab is active. The configuration fields are as follows:

- Name:** nifi_amount_items_queued
- Type:** Dependent item (dropdown menu)
- Key:** prometheus.nifi.queued
- Master item:** Template App NIFI - Flows: Nifi Flows Prometheus Master Item (with a 'Select' button)
- Keep lost resources period:** 1h
- Description:** (empty text area)
- Enabled:** ☒

At the bottom, there are buttons for Update, Clone, Test, Delete, and Cancel.

How did we implement it ?

Now we can create our different LLD rules for the things we would like to monitor like items queued, flowfiles received, ...

<input type="checkbox"/> Template	Name ▲	Items	Triggers	Graphs	Hosts	Key	Interval	Type	Status
<input type="checkbox"/> Template App NIFI - Flows	Nifi Flows Prometheus Master Item: nifi_amount_flowfiles_received	Item prototypes 1	Trigger prototypes	Graph prototypes	Host prototypes	prometheus.nifi.received		Dependent item	Enabled
<input type="checkbox"/> Template App NIFI - Flows	Nifi Flows Prometheus Master Item: nifi_amount_flowfiles_transferred	Item prototypes 1	Trigger prototypes	Graph prototypes	Host prototypes	prometheus.nifi.transferred		Dependent item	Enabled
<input type="checkbox"/> Template App NIFI - Flows	Nifi Flows Prometheus Master Item: nifi_amount_items_queued	Item prototypes 1	Trigger prototypes 1	Graph prototypes	Host prototypes	prometheus.nifi.queued		Dependent item	Enabled
<input type="checkbox"/> Template App NIFI - Flows	Nifi Flows Prometheus Master Item: nifi_average_lineage_duration	Item prototypes 1	Trigger prototypes	Graph prototypes	Host prototypes	prometheus.nifi.duration		Dependent item	Enabled

Displaying 4 of 4 found

How did we implement it ?

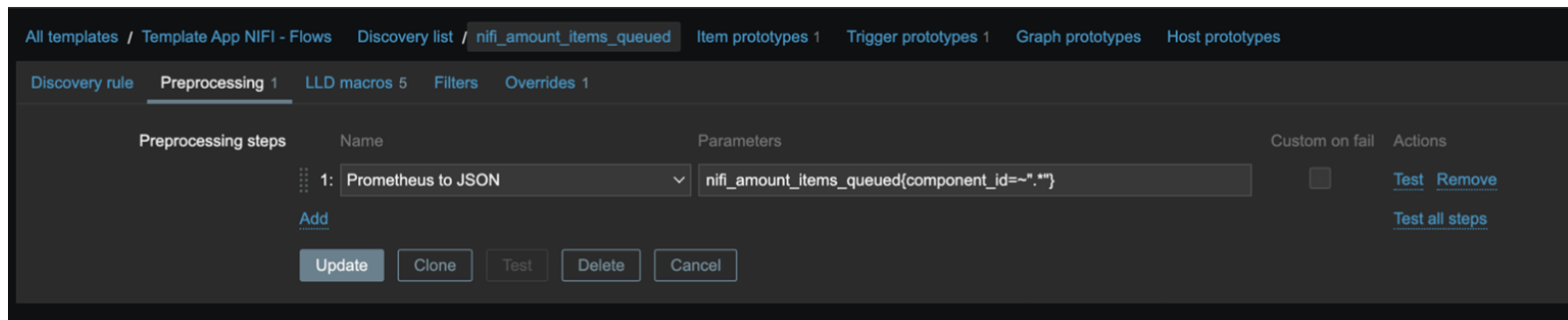
This is how our Prometheus code looks like with part of the data that we need for our items.

```
97 nifi_percent_used_count{instance="nifi-01",component_type="Connection",component_name="cn_nifi-01-nifi-02_lookup.failure",component_id="16b  
98 nifi_percent_used_count{instance="nifi-01",component_type="Connection",component_name="cn_nifi-01-nifi-02_success",component_id="68aa36a2-7d  
99 nifi_percent_used_count{instance="nifi-01",component_type="Connection",component_name="cn_nifi-01-nifi-02_parse_parquet_filename.success",component  
100 nifi_percent_used_count{instance="nifi-01",component_type="Connection",component_name="cn_nifi-01-nifi-02_set_zip_filename.success",component_id="84  
101 nifi_percent_used_count{instance="nifi-01",component_type="Connection",component_name="cn_nifi-01-nifi-02_upload_artifact.failure",component_id="053ab89e  
102 nifi_percent_used_count{instance="nifi-01",component_type="Connection",component_name="cn_nifi-01-nifi-02_evaluate_ingest.failure",component_id="(C  
103 nifi_percent_used_count{instance="nifi-01",component_type="Connection",component_name="cn_nifi-01-nifi-02_check_schema_version.success",compor  
104 nifi_percent_used_count{instance="nifi-01",component_type="Connection",component_name="cn_nifi-01-nifi-02_upload_artefact.all",component
```

How did we implement it ?

Next thing is to create our Preprocessing filters to get the needed information out the prometheus stream in this case the queued items. The data will be converted to JSON. We will use `component_id` as it is the unique part of the stream.

```
nifi_amount_items_queued{component_id=~".*"} }
```



How did we implement it ?

Our next step is to make some mapping between the LLD macros and the JSON code in the LLD macros tab.

The screenshot shows the Zabbix configuration interface for LLD macros. At the top, there are tabs: "Discovery rule", "Preprocessing 1", "LLD macros 5", "Filters", and "Overrides 1". The "LLD macros 5" tab is selected. Below the tabs, there is a table with two columns: "LLD macro" and "JSONPath". The table contains five rows of mappings, each with a "Remove" button to its right. Below the table is an "Add" button. At the bottom of the interface, there are five buttons: "Update", "Clone", "Test", "Delete", and "Cancel".

LLD macro	JSONPath	
{#COMPONENT.ID}	\$.labels['component_id']	Remove
{#LABEL}	\$.labels['component_name']	Remove
{#LINERAW}	\$.['line_raw']	Remove
{#NAME}	\$.['name']	Remove
{#VALUE}	\$.['value']	Remove

[Add](#)

[Update](#) [Clone](#) [Test](#) [Delete](#) [Cancel](#)

How did we implement it ?

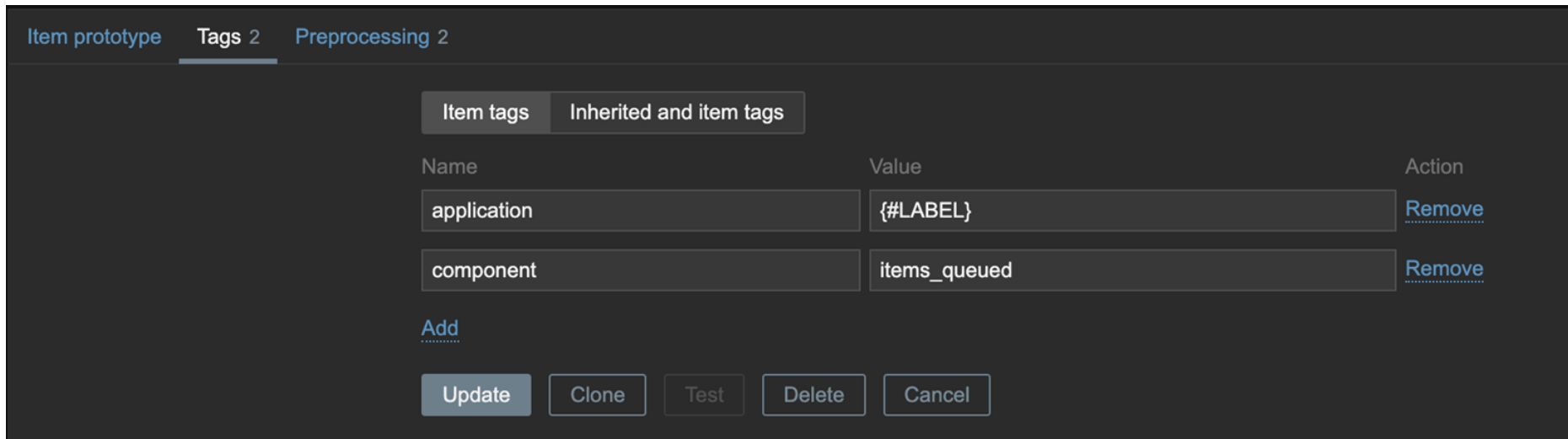
Now it's time to create our item prototype. This is a dependent item on our master item where we can use the macros from our LLD rule.

The screenshot shows the 'Item prototype' configuration window in NiFi. The 'Preprocessing 2' tab is active. The configuration includes:

- Name:** A text field containing the macro `{#NAME} {#LABEL} {#COMPONENT.ID}`.
- Type:** A dropdown menu set to 'Dependent item'.
- Key:** A text field containing the macro `items_queued[{#COMPONENT.ID}]` with a 'Select' button to the right.
- Type of information:** A dropdown menu set to 'Numeric (float)'.
- Master item:** A text field containing 'Template App NIFI - Flows: Nifi Flows Prometheus Master Item' with 'Select' and 'Select prototype' buttons to the right.
- Units:** An empty text field.
- History storage period:** A section with a 'Do not keep history' checkbox and a 'Storage period' dropdown set to '7d'.
- Trend storage period:** A section with a 'Do not keep trends' checkbox and a 'Storage period' dropdown set to '180d'.
- Value mapping:** A text field containing 'type here to search' with a 'Select' button to the right.
- Description:** A large empty text area.
- Create enabled:** A checked checkbox.
- Discover:** A checked checkbox.
- Buttons:** 'Update', 'Clone', 'Test', 'Delete', and 'Cancel' buttons at the bottom.

How did we implement it ?

For the reporting it's also important to create dynamic tags as they will contain our application name. So we add the `{#LABEL}` macro in our tag as our tag value for the application.



The screenshot shows the Apache NiFi 'Tags' configuration page. At the top, there are tabs for 'Item prototype', 'Tags 2' (which is selected), and 'Preprocessing 2'. Below the tabs, there are two sub-tabs: 'Item tags' and 'Inherited and item tags'. The 'Item tags' sub-tab is active, displaying a table with two columns: 'Name' and 'Value'. There are two rows in the table. The first row has 'application' in the 'Name' column and '{#LABEL}' in the 'Value' column. The second row has 'component' in the 'Name' column and 'items_queued' in the 'Value' column. To the right of each row is a 'Remove' button. Below the table, there is an 'Add' button. At the bottom of the configuration area, there are five buttons: 'Update', 'Clone', 'Test', 'Delete', and 'Cancel'.

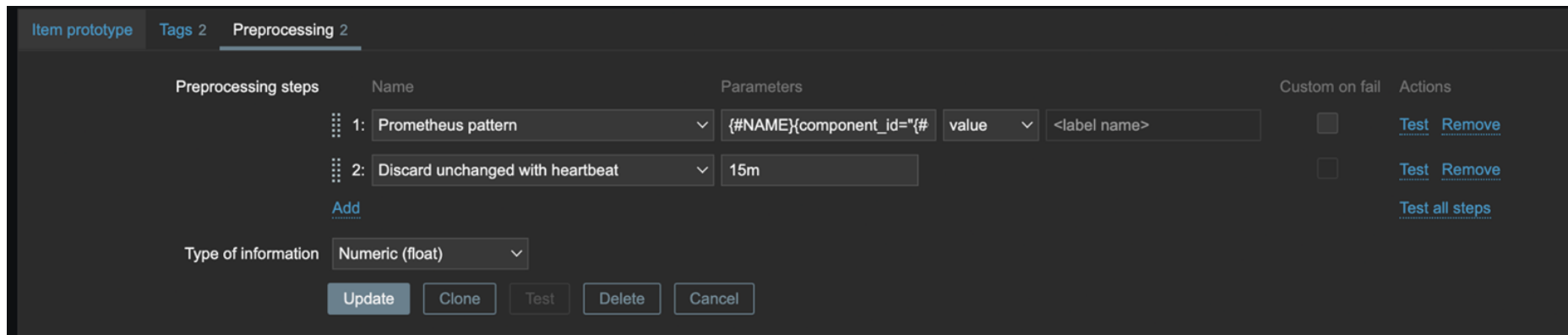
Name	Value	Action
application	{#LABEL}	Remove
component	items_queued	Remove

Buttons: Add, Update, Clone, Test, Delete, Cancel

How did we implement it ?

And don't forget to filter our prometheus data for the pattern we like to use for our item by making use of the filters we made in our LLD rule.

Our item isn't changing all the time so we don't need to keep it in our database every minute. For this we add some "Discard unchanged with heartbeat" to our preprocessing step.



The screenshot shows the Zabbix Item configuration interface. The top navigation bar includes 'Item prototype', 'Tags 2', and 'Preprocessing 2'. The main section is titled 'Preprocessing steps' and contains a table with two rows:

	Name	Parameters	Custom on fail	Actions
1:	Prometheus pattern	{#NAME}{component_id}="# value <label name>	<input type="checkbox"/>	Test Remove
2:	Discard unchanged with heartbeat	15m	<input type="checkbox"/>	Test Remove

Below the table, there is an 'Add' button and a 'Type of information' dropdown menu set to 'Numeric (float)'. At the bottom, there are buttons for 'Update', 'Clone', 'Test', 'Delete', and 'Cancel'.

How did we implement it ?

Our item prototype once it's ready

All templates / Template App NIFI - Flows									
Discovery list / nifi_amount_items_queued									
Item prototypes 1									
Trigger prototypes 1									
Graph prototypes									
Host prototypes									
<input type="checkbox"/>	Name ▲	Key	Interval	History	Trends	Type	Create enabled	Discover	Tags
<input type="checkbox"/>	... Nifi Flows Prometheus Master Item: {#NAME}.{#LABEL}.{#COMPONENT.ID}	items_queued[{#COMPONENT.ID}]	7d	180d		Dependent item	Yes	Yes	application: {#LABEL} component: items_qu...
Displaying 1 of 1 found									

How did we implement it ?

And finally we create our trigger.

But we would only like to see triggers fire off when the queues are failed or inactive so we need tell zabbix to not make any trigger where the queue is higher then 0 unless the queue is inactive or in failure state.

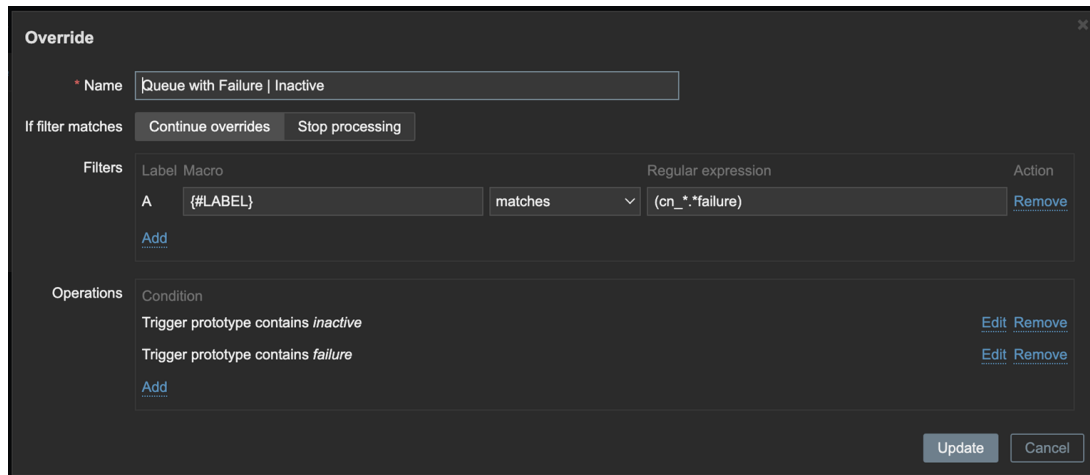
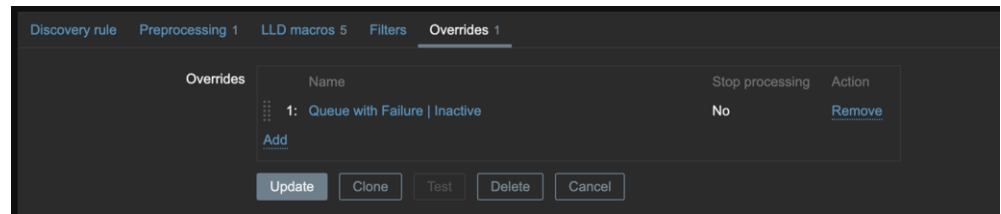
So we set "create enabled" on "no" this allows us to only create triggers with an override rule

<input type="checkbox"/> Severity	Name ▲	Operational data	Expression	Create enabled	Discover	Tags
<input type="checkbox"/> Warning	{#LABEL}. {#COMPONENT.ID} is bigger then > 0	{ITEM.LASTVALUE}	last(/Template App NIFI - Flows/items_queued[{#COMPONENT.ID}])>0	No	Yes	application: {#LABEL} scope: performance

Displaying 1 of 1 found

How did we implement it ?

Since we only like to see items with queues that have failed or that are inactive. This is done by creating an override rule on our LLD discovery rule in the overrides tab



The override rule will create triggers when it matches our override filter so our original trigger prototype has create enabled no.

How did we implement it ?

Finally to report only to the groups needed we create our trigger action where we tell to send a notification if the value of our tag application has a certain value

Actions

Action Operations 3

* Name

Team_XXXXXXXX_Nifi

Type of calculation

And/Or

A and B

Conditions

Label	Name	Action
A	Value of tag application contains XXXXXXXX	Remove
B	Host group equals Nifi	Remove
Add		

Enabled

☒

* At least one operation must exist.

Update

Clone

Delete

Cancel

Future Ideas

There is no standard Zabbix NIFI Template and it seems that NIFI has a rest API. I would like to get rid of the prometheus plugin for monitoring and call the rest API and make it more generic in the future.