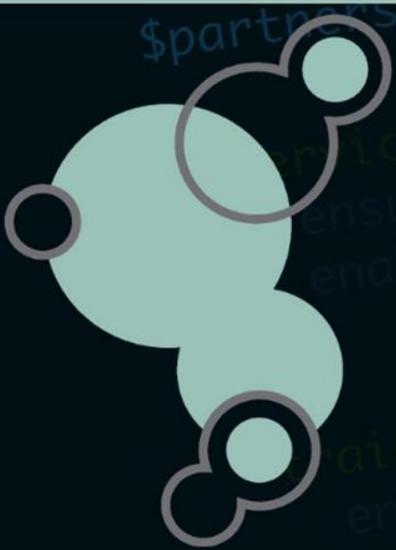


```
# Open-Future
#
# Installs, configures and keeps your open source
# infrastructure up and running.

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



open-future

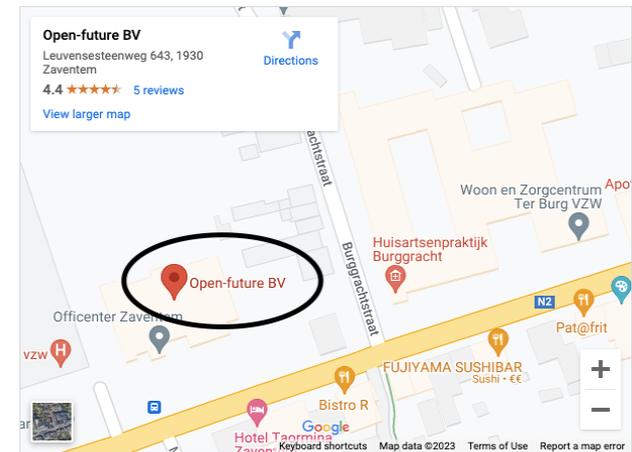
# NIFI Flow monitoring With Zabbix

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

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



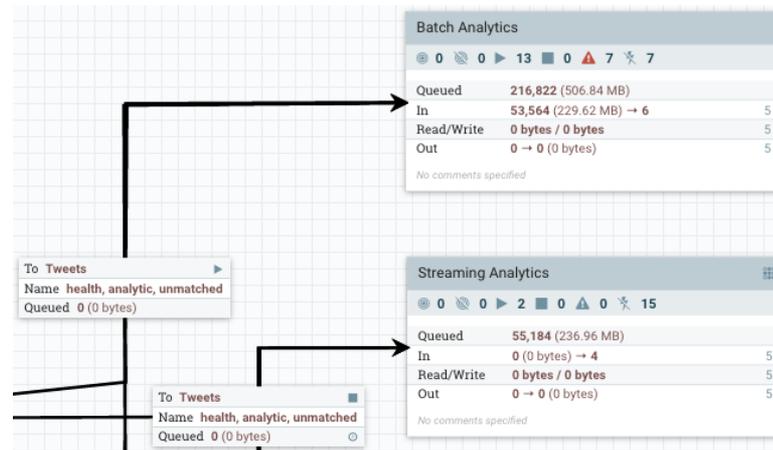
# Who is Patrik Uytterhoeven

- 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.

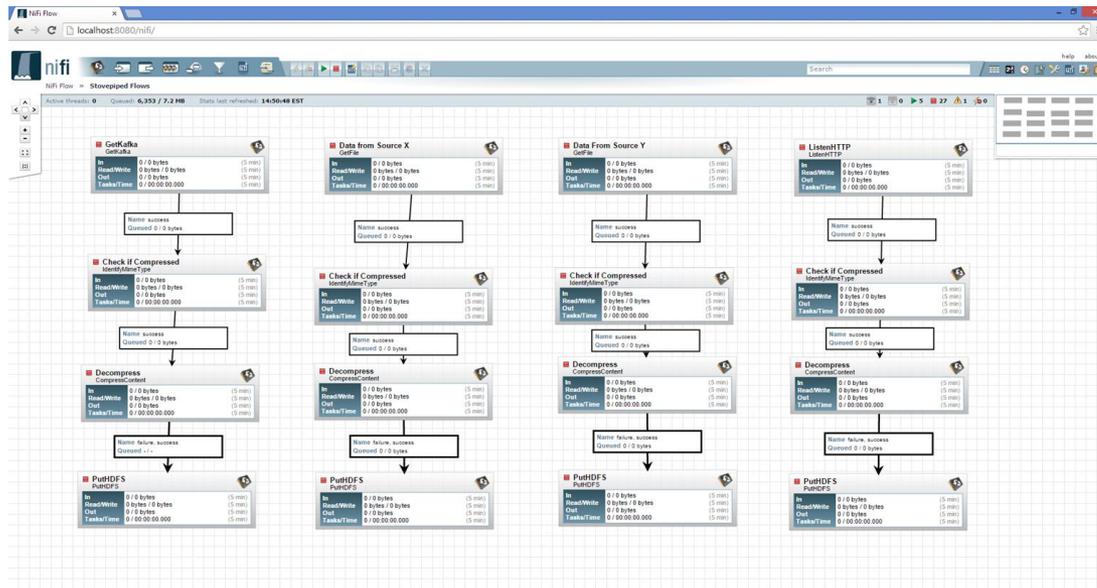
Apache NiFi supports powerful and scalable directed graphs of data routing, transformation, and system mediation logic.



# What would we like to monitor

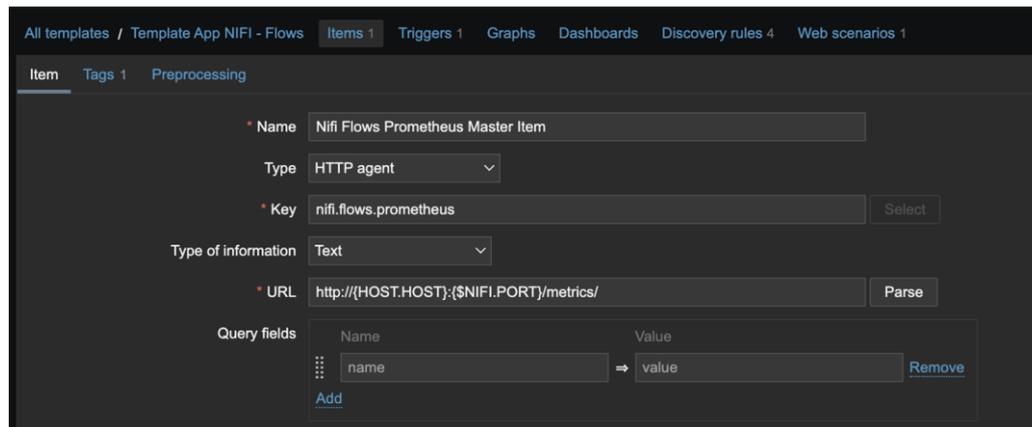
We like to monitor the flows created in NIFI for each database and send messages to the teams only if there is a problem with one of their databases.

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.



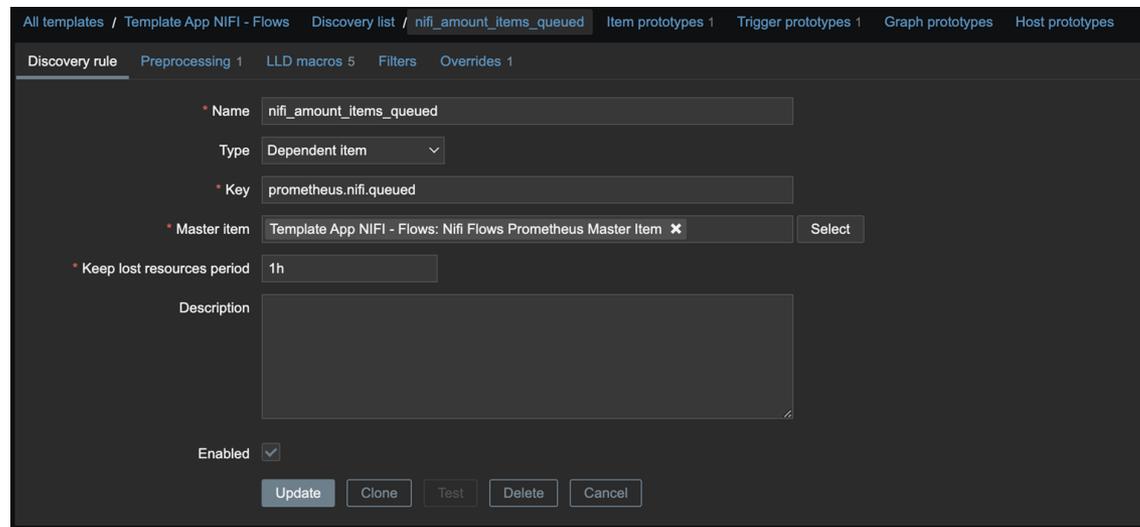
The screenshot shows the Zabbix web interface for configuring a new item. The breadcrumb navigation at the top reads: "All templates / Template App NIFI - Flows / Items 1 / Triggers 1 / Graphs / Dashboards / Discovery rules 4 / Web scenarios 1". The current page is "Item" under "Tags 1" and "Preprocessing".

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". One field is added with "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 added field.

# 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 Zabbix configuration interface for a Low Level Discovery (LLD) rule. The breadcrumb navigation at the top indicates the path: All templates / Template App NIFI - Flows / Discovery list / nifi\_amount\_items\_queued. Below the breadcrumb, there are tabs for 'Discovery rule', 'Preprocessing 1', 'LLD macros 5', 'Filters', and 'Overrides 1'. The main configuration area includes the following fields:

- 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 of the configuration area, there are five buttons: '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, ...

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

Displaying 4 of 4 found

# How did we implement it ?

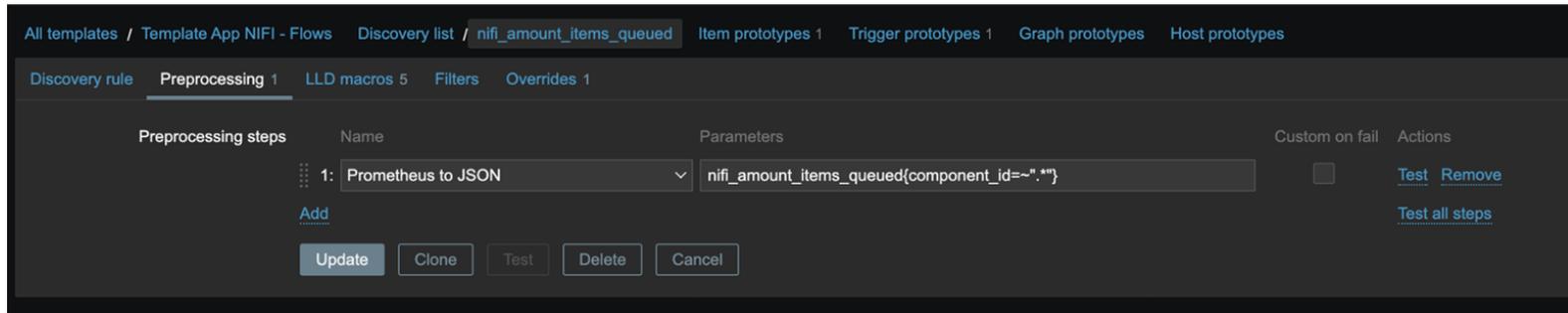
This is how our Prometheus code looks like with part of the data that we need for our items.

```
97 nifi_percent_used_count(instance="...",component_type="Connection",component_name="cn_...lookup.failure",component_id="16b
98 nifi_percent_used_count(instance="...",component_type="Connection",component_name="cn_...success",component_id="68aa36a2-7d
99 nifi_percent_used_count(instance="...",component_type="Connection",component_name="cn_...parse_parquet_filename.success",component
100 nifi_percent_used_count(instance="...",component_type="Connection",component_name="cn_...set_zip_filename.success",component_id="84
101 nifi_percent_used_count(instance="...",component_type="Connection",component_name="cn_...artifact.failure",component_id="053ab89e
102 nifi_percent_used_count(instance="...",component_type="Connection",component_name="cn_...evaluate_ingest.failure",component_id="(
103 nifi_percent_used_count(instance="...",component_type="Connection",component_name="cn_...check_schema_version.success",compor
104 nifi_percent_used_count(instance="...",component_type="Connection",component_name="cn_...duoso_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=~".*"}}
```

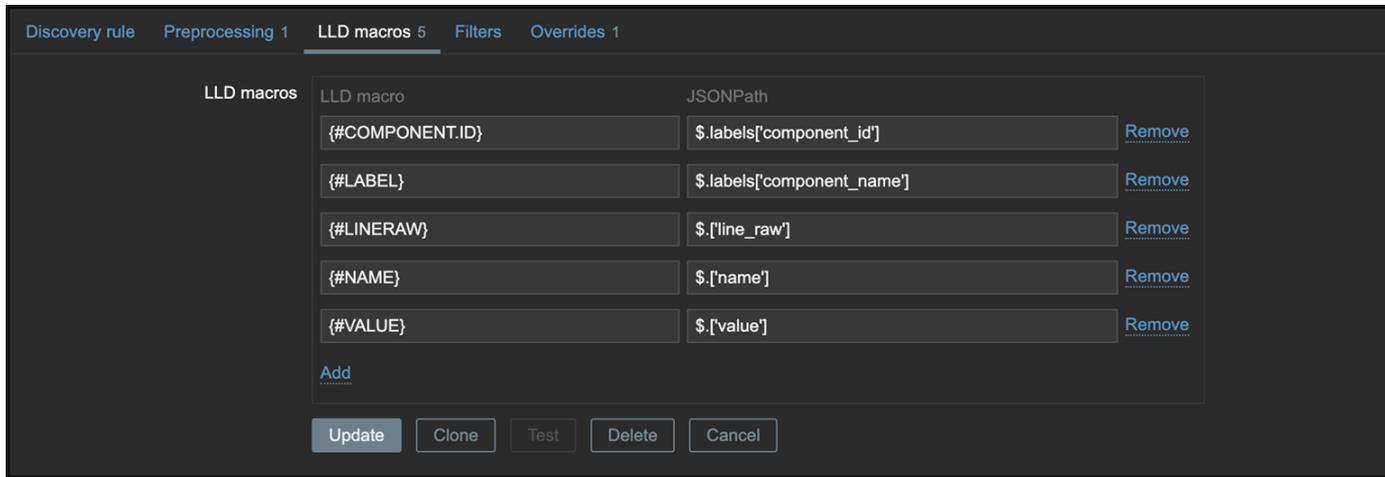


The screenshot shows the NiFi configuration interface for a Preprocessing step. The breadcrumb navigation at the top indicates the path: All templates / Template App NIFI - Flows / Discovery list / nifi\_amount\_items\_queued. Below this, the configuration is for a Preprocessing step named 'Prometheus to JSON'. The parameters field contains the query `nifi_amount_items_queued{component_id=~".*"}}`. The 'Custom on fail' checkbox is unchecked. The 'Actions' column contains links for 'Test', 'Remove', and 'Test all steps'. At the bottom, there are buttons for 'Update', 'Clone', 'Test', 'Delete', and 'Cancel'.

Preprocessing steps	Name	Parameters	Custom on fail	Actions
1:	Prometheus to JSON	<code>nifi_amount_items_queued{component_id=~".*"}}</code>	<input type="checkbox"/>	<a href="#">Test</a> <a href="#">Remove</a> <a href="#">Test all steps</a>

# 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. The top navigation bar includes tabs for "Discovery rule", "Preprocessing 1", "LLD macros 5", "Filters", and "Overrides 1". The "LLD macros" tab is active, displaying a table with two columns: "LLD macro" and "JSONPath". Each row contains a macro name, its corresponding JSONPath, and a "Remove" link. Below the table is an "Add" link and a row of action buttons: "Update", "Clone", "Test", "Delete", and "Cancel".

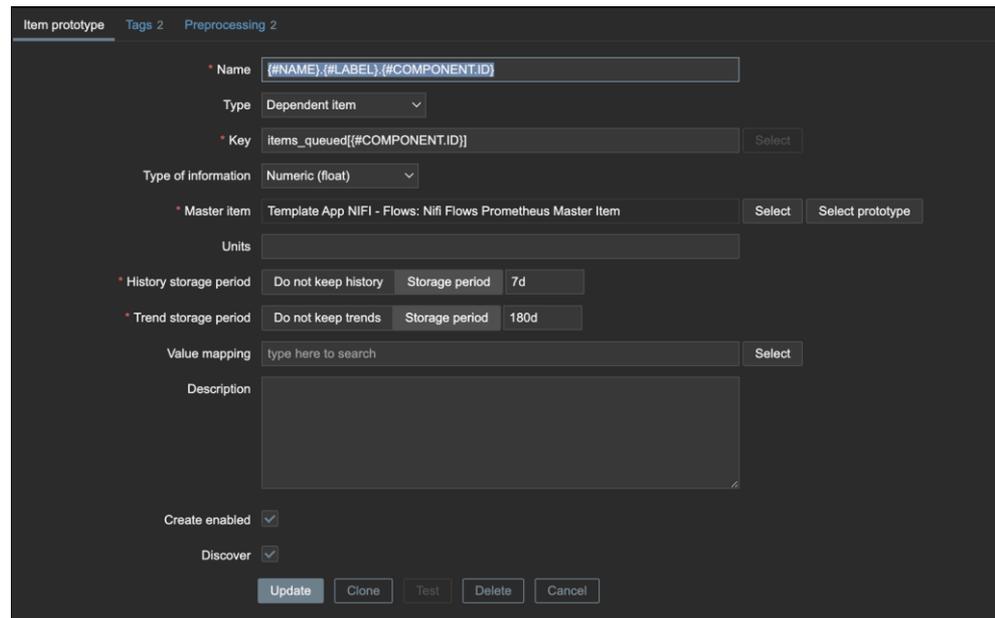
LLD macro	JSONPath	
{#COMPONENT.ID}	\$.labels['component_id']	<a href="#">Remove</a>
{#LABEL}	\$.labels['component_name']	<a href="#">Remove</a>
{#LINERAW}	\$.['line_raw']	<a href="#">Remove</a>
{#NAME}	\$.['name']	<a href="#">Remove</a>
{#VALUE}	\$.['value']	<a href="#">Remove</a>

[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.

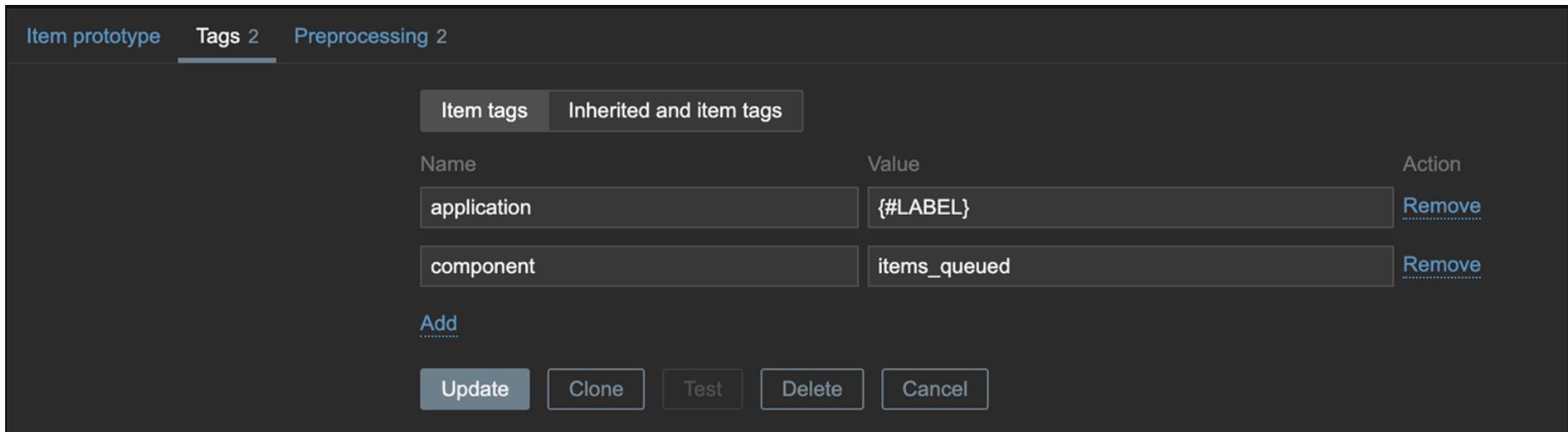


The screenshot shows the Zabbix 'Item prototype' configuration window. The interface is dark-themed and contains the following fields and options:

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

# 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 Nifi UI configuration for item tags. At the top, there are tabs for 'Item prototype', 'Tags 2', and 'Preprocessing 2'. Below the tabs, there are two buttons: 'Item tags' and 'Inherited and item tags'. The main area displays a table with columns for 'Name', 'Value', and 'Action'. The table contains two rows: one for 'application' with value '{#LABEL}' and one for 'component' with value 'items\_queued'. Below the table, there is an 'Add' button and a row of action 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 prototype configuration interface. The 'Preprocessing 2' tab is active. It displays a table of preprocessing steps:

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

Below the table, there is an 'Add' button and a 'Type of information' dropdown 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.

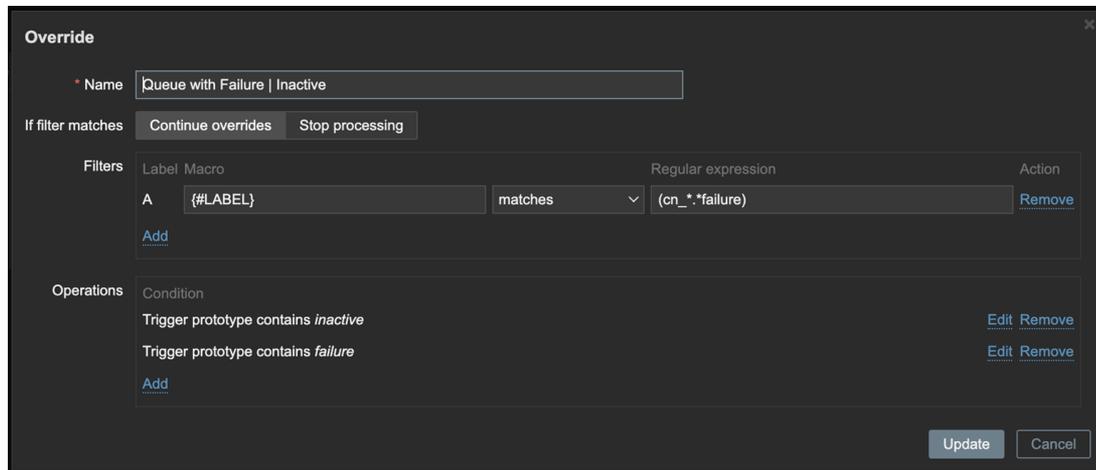
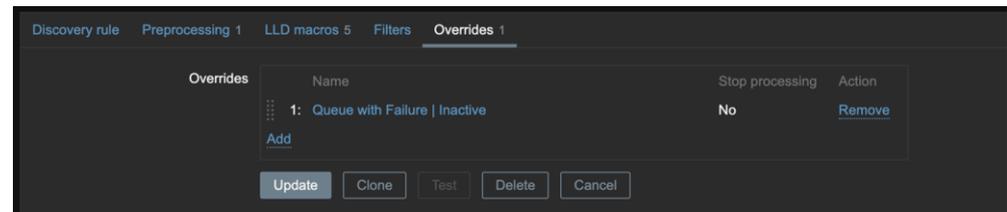
So we set "create enabled" on "no" this allows us to only create triggers with an override rule

<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

Type of calculation  A and B

Conditions	Label	Name	Action
A		Value of tag application contains XXXXXX	<a href="#">Remove</a>
B		Host group equals Nifi	<a href="#">Remove</a>
<a href="#">Add</a>			

Enabled

\* At least one operation must exist.

# 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.