NIFI Flow monitoring With Zabbix

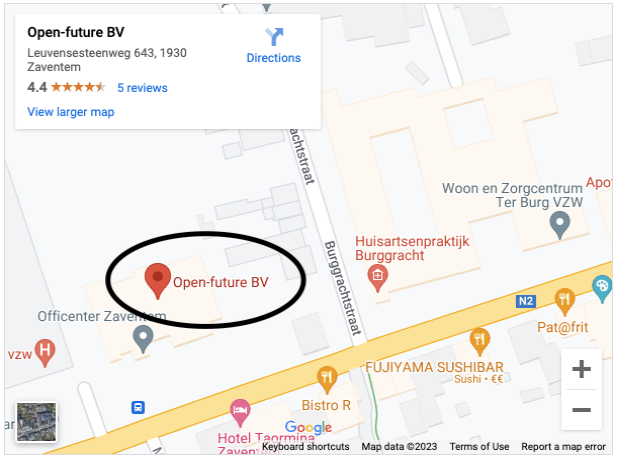
How to monitor NIFI flows with Zabbix by using Prometheus LLD, Tags and overrides

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

ZABBIX
PREMIUM PARTNER



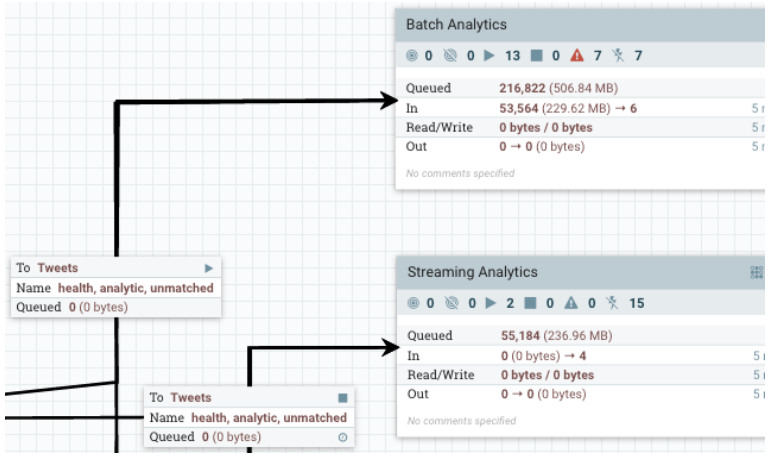
Who is Patrik Uytterhoeven

- 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.

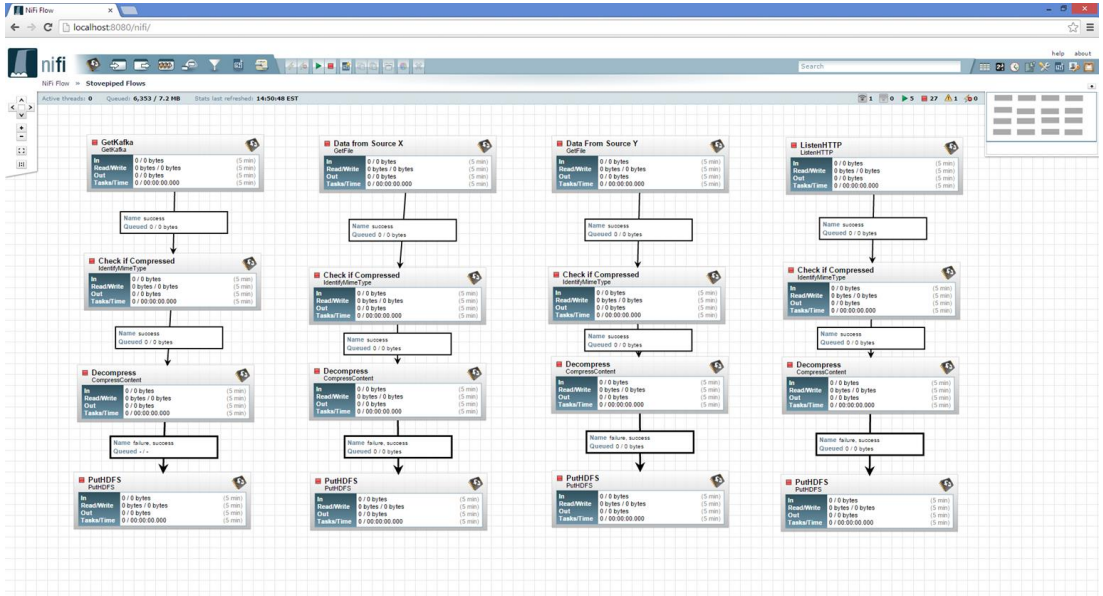
Apache NiFi supports powerful and scalable directed graphs of data routing, transformation, and system mediation logic.



What would we like to monitor

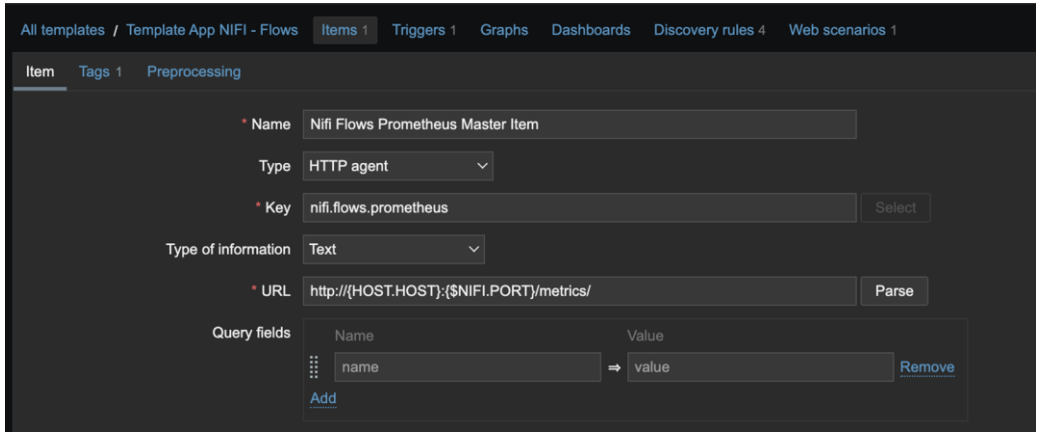
We like to monitor the flows created in NIFI for each database and send messages to the teams only if there is a problem with one of their databases.

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



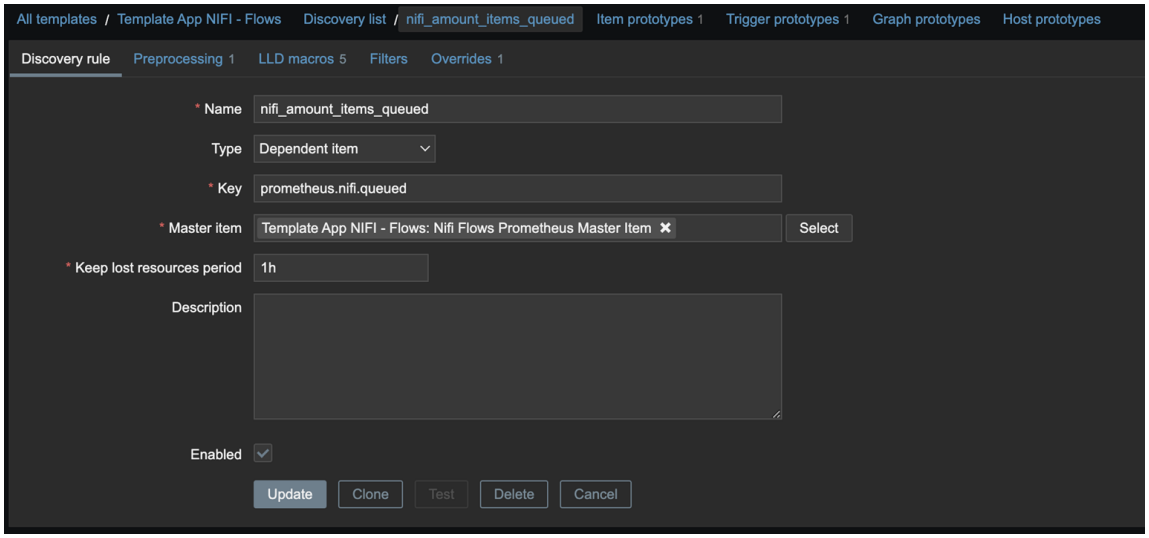
How did we implement it ?

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



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.



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
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
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
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
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 ?

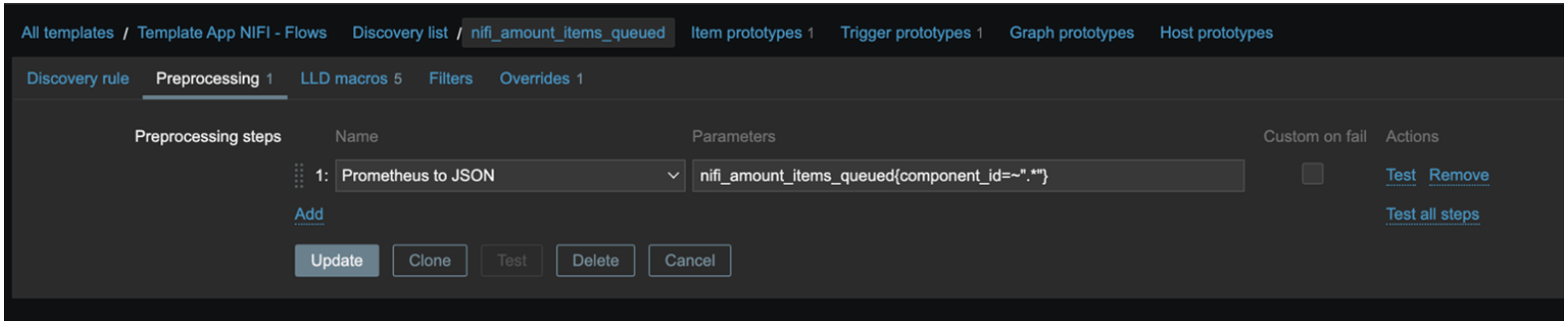
This is how our Prometheus code looks like with part of the data that we need for our items.

```
97  nifi_percent_used_count{instance="[redacted]",component_type="Connection",component_name="cn_[redacted]_lookup.failure",component_id="16b[redacted]
98  nifi_percent_used_count{instance="[redacted]",component_type="Connection",component_name="cn_[redacted]cess",component_id="68aa36a2-7d[redacted]
99  nifi_percent_used_count{instance="[redacted]",component_type="Connection",component_name="cn_[redacted]_parse_parquet_filename.success",component[redacted]
100 nifi_percent_used_count{instance="[redacted]",component_type="Connection",component_name="cn_[redacted]_set_zip_filename.success",component_id="84[redacted]
101 nifi_percent_used_count{instance="[redacted]",component_type="Connection",component_name="cn_[redacted]_artifact.failure",component_id="053ab89e[redacted]
102 nifi_percent_used_count{instance="[redacted]",component_type="Connection",component_name="cn_[redacted]evaluate_ingest.failure",component_id="[redacted]
103 nifi_percent_used_count{instance="[redacted]",component_type="Connection",component_name="cn_[redacted]check_schema_version.success",compor[redacted]
104 nifi_percent_used_count{instance="[redacted]",component_type="Connection",component_name="cn_[redacted]so_upload_artefact.all",component[redacted]
```

How did we implement it ?

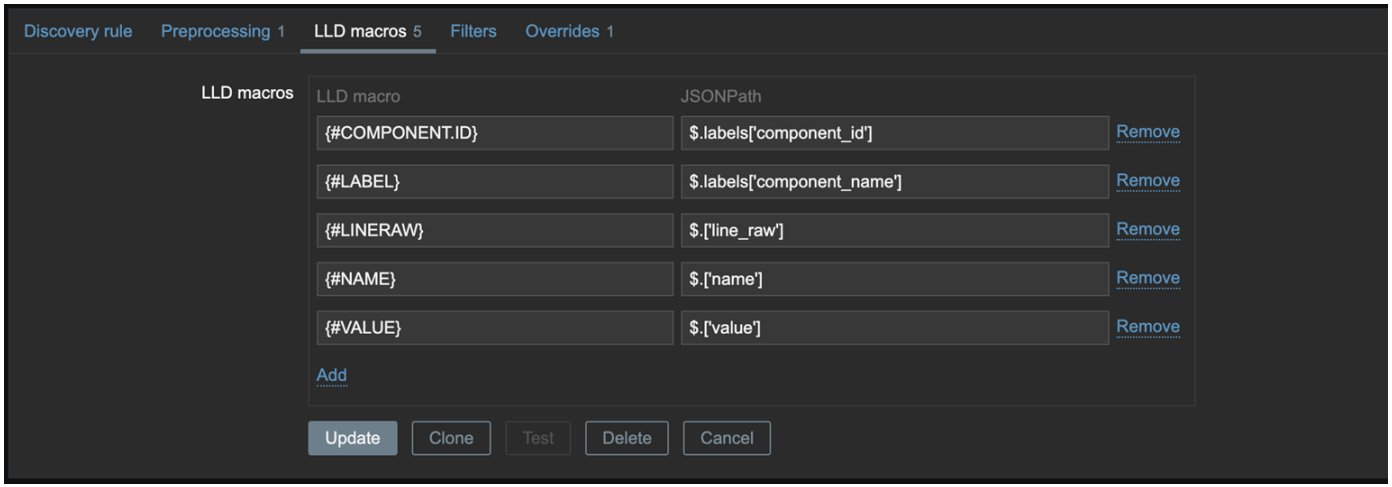
Next thing is to create our Preprocessing filters to get the needed information out the prometheus stream in this case the queued items. The data will be converted to JSON. We will use component_id as it is the unique part of the stream.

nifi_amount_items_queued{component_id=~".*"}



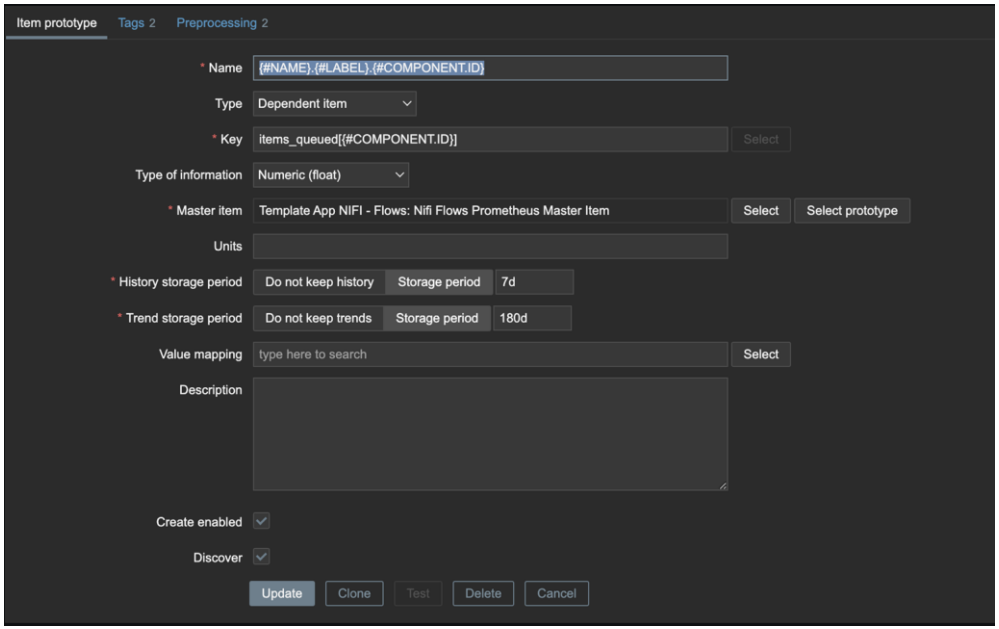
How did we implement it ?

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



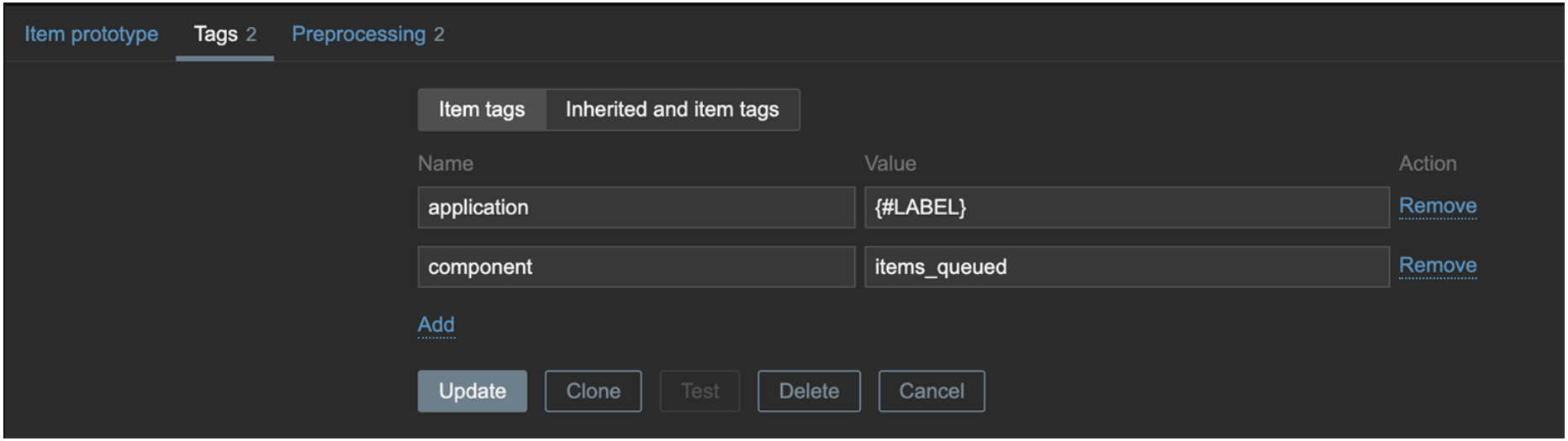
How did we implement it ?

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.



How did we implement it ?

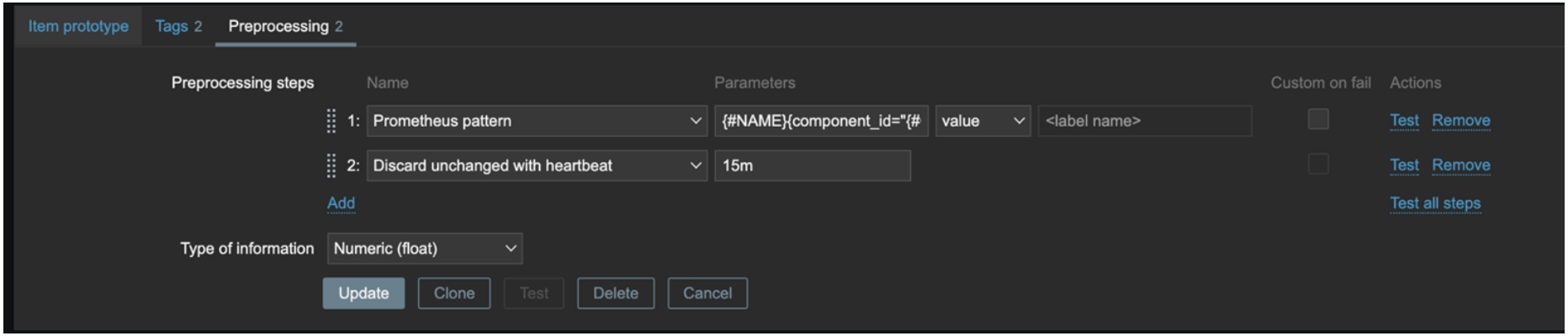
For the reporting it's also important to create dynamic tags as they will contain our application name. So we add the {#LABEL} macro in our tag as our tag value for the application.



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.



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

Name ▲	Key	Interval	History	Trends	Type	Create enabled	Discover	Tags
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.

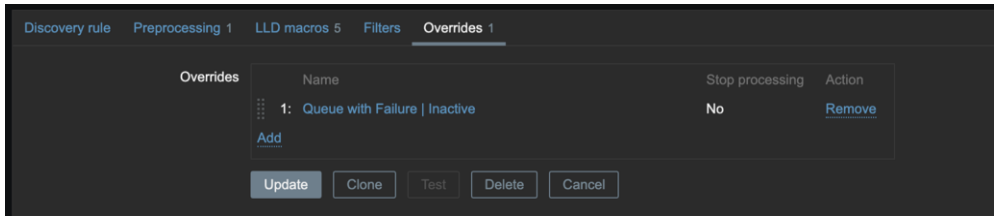
So we set ”create enabled” on ”no” this allows us to only create triggers with an override rule

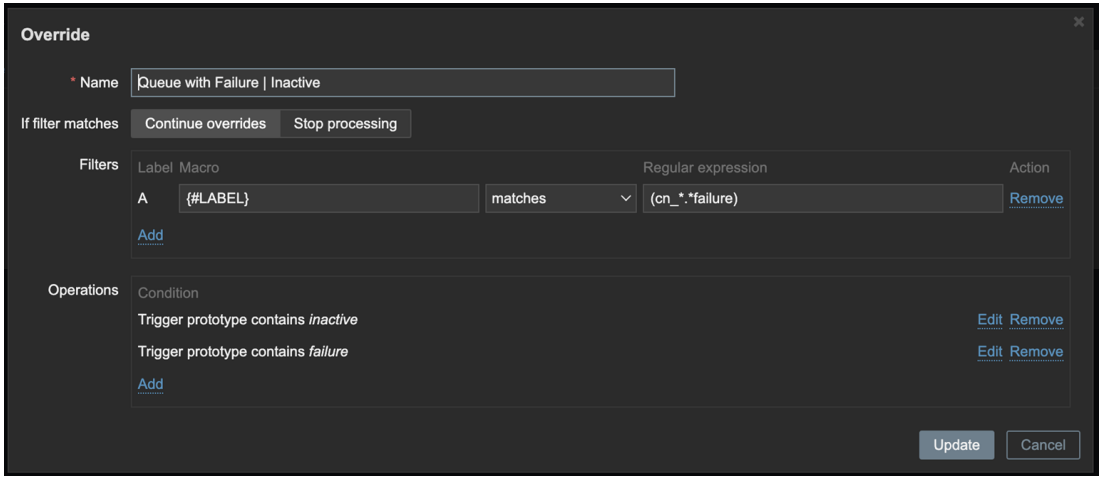
Severity	Name ▲	Operational data	Expression	Create enabled	Discover	Tags
Warning	{#LABEL}.{#COMPONENT.ID}is bigger then > 0	{ITEM.LASTVALUE}	last(/Template App NIFI - Flows/items_queued[{#COMPONENT.ID}])>0	No	Yes	application: {#LABEL} scope: performance

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

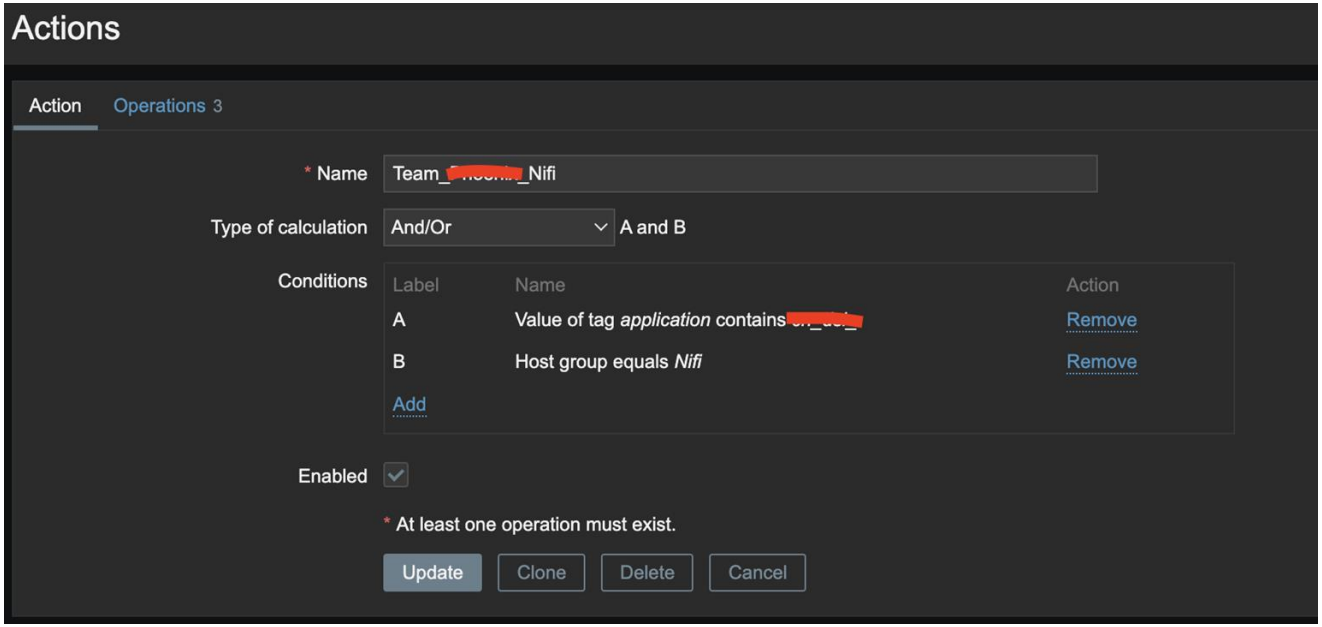




The override rule will create triggers when it matches our override filter so our original trigger prototype has create enabled no.

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



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.