



POC : Using Fluentd with Zabbix API for Log Monitoring

By: Patrik Uytterhoeven





Introduction

One of our clients wanted to replace a monitoring tool that was not meeting their needs. They had set up a large number of custom scripts, and everything was sent to Fluentd.

As a proof of concept, we set up Zabbix and integrated Fluentd so that they could move to Zabbix without making any changes to their current setup. This would give time to replace later scripts with a built-in Zabbix functionality.

About us

Open-Future is a Belgian company specializing in Open Source and Linux solutions. It was founded on August 10, 2009, as a private limited company.

We focus on providing trainings, consultancy, and services in areas such as configuration management, backups, groupware, monitoring, security, and infrastructure solutions.



The problem

The client had a solution in place but was unhappy about it, as the solution was expensive and not doing what they wanted.

A variety of third-party tools and scripts had been placed in production to solve their needs.

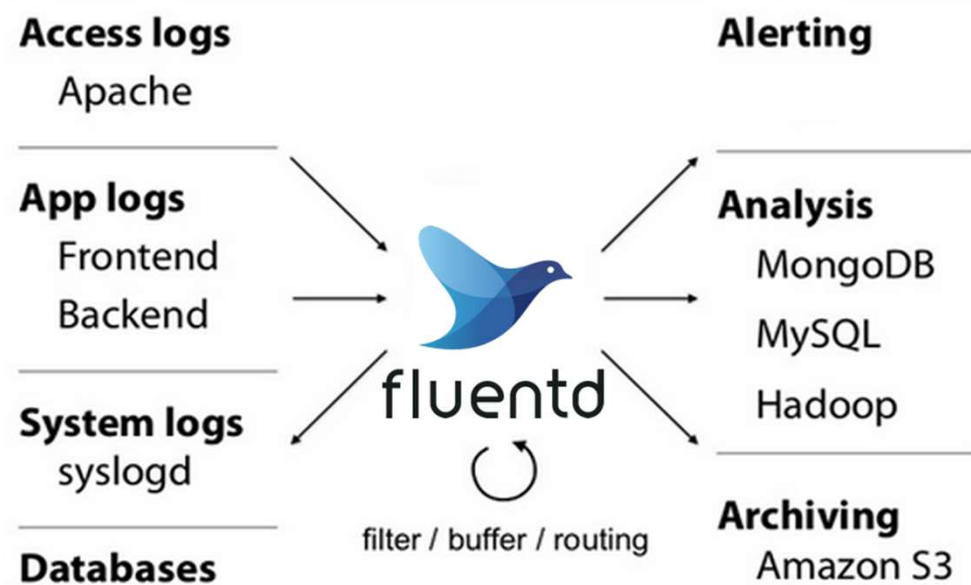
The Problem

The client had Fluentd in place to gather logs from a list of different applications and script outputs. They wanted to keep this in place instead of using Zabbix agents as a first step to a quick migration. We had to find a simple way to integrate Fluentd with Zabbix.



What is Fluentd?

Fluentd is an open-source data collector that unifies data collection and consumption for better use and understanding of data.

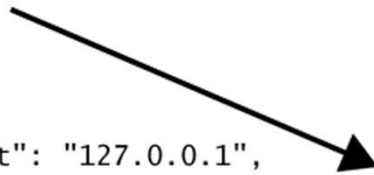


Unified Logging with JSON

Fluentd will try to structure data as JSON as much as possible.
This allows us to use Fluentd with the API of Zabbix 7.x to push data directly into an item.



```
127.0.0.1 - - [05/Feb/2012:17:11:55 +0000]  
"GET / HTTP/1.1" 200 140 "-" "Mozilla/5.0...
```



```
{  
  "host": "127.0.0.1",  
  "user": "-",  
  "method": "GET",  
  "agent": "Mozilla/5.0 (Windows..."  
}
```



How to configure Fluentd

First, we tell Fluentd to create a JSON for Zabbix and define the item where we would like to send our data.

```
##-----  
## check_healthcheck  
##-----  
  
<filter Zabbix-payload>  
  @type record_transformer  
  enable_ruby  
  <record>  
    jsonrpc "2.0"  
    method "history.push"  
    params [{"itemid": "50034", "value": "${record['message']}"} ]  
    auth null  
    id 1  
  </record>  
  remove_keys message  
</filter>
```

How to configure Fluentd

Next, we tell Fluentd to authenticate to our Zabbix API and send the content of our data.

```
##-----  
## send it  
##-----  
<match Zabbix-payload>  
  
  @type http  
  endpoint http://Zabbix-server/zabbix/api\_jsonrpc.php  
  http_method post  
  <format>  
    @type json  
  </format>  
  json_array true  
  headers {"Authorization":"Bearer <Zabbix API token>"}  
  
  <buffer>  
    flush_mode immediate  
    chunk_limit_records 1  
  </buffer>  
  
</match>
```


Create our Zabbix item

The next step is to create our trapper item to catch the data from Fluentd. We also need to add some preprocessing step, as our data from the scripts is separated with semicolons.

Item ? ×

Item Tags Preprocessing 1

* Name

Type

* Key

Type of information

* History Store up to

Allowed hosts

Populates host inventory field

Description

Enabled

What does the date look like ?

This is how our actual data will look when Zabbix receives it from Fluentd:

```
[{"dateTime":"2025-01-29 12:04:34+0100","hostname":"hostname.common.eu.corp.company.com","elementType":"logmon","elementName":"u01.app.oraopendbd.diag.rdbms.opendbd.OPENDBD.trace.alert_OPENDBD.log","parameter":"ALERT in u01.app.oraopendbd.diag.rdbms.opendbd.OPENDBD.trace.alert_OPENDBD.log","value":"ORA-1000 - please ignore","status":"ALERT"}]
```

Split the data with JS

The JS will format the data in JSON and also create a unique ID that we can use for our item key based on some parameters that are unique per item.

JavaScript

```
function (value) {  
  1 var parts = value.split(';');  
  2 var combinedString = [parts[1], parts[2], parts[3], parts[4]].join('|');  
  3  
  4  
  5 var uniqId = sha256(combinedString).slice(0, 10);  
  6  
  7 return JSON.stringify({  
  8   hostname: parts[1],  
  9   elementType: parts[2],  
 10   elementName: parts[3],  
 11   parameter: parts[4],  
 12   value: parts[5],  
 13   status: parts[6],  
 14   uniqId: uniqId // New SHA-256 encoded variable, limited to 10 characters  
 15 });  
}
```

65127 characters remaining

Apply

Cancel

Time to create our LLD rule

We create an LLD rule that is a dependent item on our master item.
Of course, we need to use JSONPath to extract our macros so we can use them in our LLD items.

Discovery rule **Preprocessing** LLD macros 7 Filters Overrides 1

* Name

Type

* Key

* Master item

* Delete lost resources ?

* Disable lost resources ?

Description

Enabled

Time to create our LLD rule

With JSONPath, we are now able to create a set of LLD macros to use in our items.

Discovery rule Preprocessing **LLD macros 7** Filters Overrides 1

LLD macros

LLD macro	JSONPath	
{#ELEMENTNAME}	\$.elementName.first()	Remove
{#ELEMENTTYPE}	\$.elementType.first()	Remove
{#HOSTNAME}	\$.hostname.first()	Remove
{#PARAMETER}	\$.parameter.first()	Remove
{#STATUS}	\$.status.first()	Remove
{#UNIQID}	\$.uniqId.first()	Remove
{#VALUE}	\$.value.first()	Remove

[Add](#)

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

The Problem

We sent one payload to Zabbix and have to use this to create our LLD items. We also need to use the same data to fill in the values, as Fluentd will only send this information to Zabbix once.

The solution is to create a calculated item to duplicate the data from our master item. This item can be used to populate the items in our prototypes.

We also add a preprocessing step to discard duplicated values.

The Solution

Item ? x

Item Tags Preprocessing 1

* Name

Type

* Key

Type of information

* Formula

* Update interval

Custom intervals

Type	Interval	Period	
<input type="checkbox"/> Flexible	<input type="checkbox"/> Scheduling	<input type="text" value="50s"/>	<input type="text" value="1-7,00:00-24:00"/> <input type="button" value="Remove"/>

[Add](#)

* History Do not store Store up to

Populates host inventory field

Description

Enabled

The LLD Item

The final step is to create our LLD item. It uses all the LLD filters we made, and it also uses the Unique ID for our item key.

Our item will extract its values from our calculated item.

Item prototype

Item prototype Tags 4 Preprocessing 1

* Name

Type

* Key

Type of information

* Master item

* History

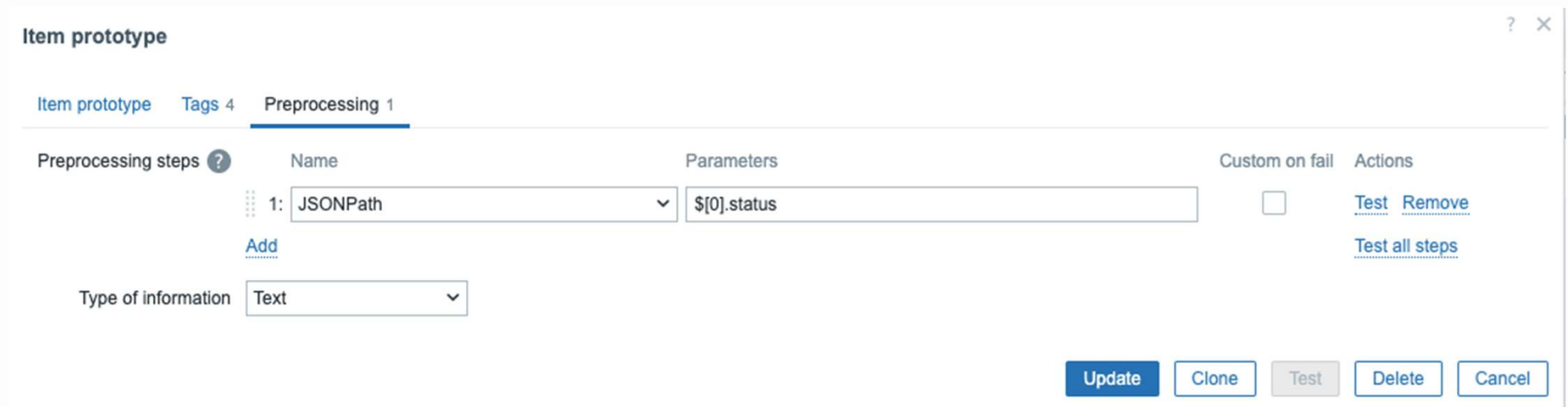
Description

Create enabled

Discover

The LLD Item

Our item value is extracted in the preprocessing step.
For this, we use the status, as the status will be used in triggers to determine the severity.



The screenshot shows the 'Item prototype' configuration window. It has a title bar with a question mark and a close button. Below the title bar, there are tabs for 'Item prototype', 'Tags 4', and 'Preprocessing 1'. The 'Preprocessing 1' tab is active. The main area is a table with columns: 'Preprocessing steps', 'Name', 'Parameters', 'Custom on fail', and 'Actions'. There is one row with the following data: '1: JSONPath' in the 'Name' column, '\$[0].status' in the 'Parameters' column, an unchecked checkbox in the 'Custom on fail' column, and 'Test Remove' and 'Test all steps' in the 'Actions' column. Below the table, there is an 'Add' button and a 'Type of information' dropdown menu set to 'Text'. At the bottom right, there are five buttons: 'Update', 'Clone', 'Test', 'Delete', and 'Cancel'.

Preprocessing steps	Name	Parameters	Custom on fail	Actions
1:	JSONPath	`\${0}.status`	<input type="checkbox"/>	Test Remove Test all steps

Type of information: Text

Update Clone Test Delete Cancel

The LLD Item

Since we extracted much more data, we use our tags to extract valuable information like hostname, value, elementType, etc.

This allows us to see which host sent the alert and what the error messages were.

Item prototype

Item prototype Tags 4 Preprocessing 1

Item tags Inherited and item tags

Name	Value	
elementType	{#ELEMENTTYPE}	Remove
hostname	{#HOSTNAME}	Remove
status	{#STATUS}	Remove
value	{#VALUE}	Remove

[Add](#)

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

The LLD triggers

Our final step is to create some useful triggers.

Since our items will be populated with values like ALERT, WARN, and INFO, we can map this with the correct severity levels in Zabbix.

The screenshot shows the 'Trigger prototype' configuration page in Zabbix. The form is titled 'Trigger prototype' and has tabs for 'Trigger prototype', 'Tags', and 'Dependencies'. The configuration includes the following fields and options:

- Name:** ALERT detected
- Event name:** ALERT detected
- Operational data:** (empty field)
- Severity:** Not classified, Information, Warning, **Average** (selected), High, Disaster
- Expression:** find(/Fluentd_template/lld.alert.{{#ONTGID}},"like","ALERT")=0
- Expression constructor:** (link)
- OK event generation:** Expression, Recovery expression, None
- PROBLEM event generation mode:** Single, Multiple
- OK event closes:** All problems, All problems if tag values match
- Allow manual close:**
- Menu entry name:** Trigger URL
- Menu entry URL:** (empty field)
- Description:** (empty text area)
- Create enabled:**
- Discover:**

At the bottom of the form, there are buttons for 'Update', 'Clone', 'Delete', and 'Cancel'.

The LLD triggers

Trigger prototype ? ×

Trigger prototype Tags Dependencies

* Name

Event name

Operational data

Severity Not classified Information Warning Average High Disaster

* Expression

[Expression constructor](#)

OK event generation Expression Recovery expression None

PROBLEM event generation mode Single Multiple

OK event closes All problems All problems if tag values match

Allow manual close

Menu entry name ?

Menu entry URL

Description

Create enabled

Discover

When the pieces fall together

The final step was to add a host dashboard with the alerts.
With the information in the tags, we can get more information on what host the problem was in, what the values were, the exact error message, etc.

Host dashboards

All hosts / Fluentd_collector < Fluentd Dashboard

Problems

Time ▼	Recovery time	Status	Info	Host	Problem • Severity	Operational data	Duration	Update	Actions	Tags
02:50:48 PM		PROBLEM	Fluentd_collect or		ALERT detected	WARN	8m 33s	Update		elementType: check_h... hostname: lxorabdb15 ...
02:49:48 PM		PROBLEM	Fluentd_collect or		ALERT detected	WARN	9m 33s	Update		elementType: check_fi... hostname: lxorabdb15 ...
02:49:48 PM		PROBLEM	Fluentd_collect or		ALERT detected	WARN	9m 33s	Update		elementType: check_cpu hostname: lxorabdb15 ...
02:40:18 PM		PROBLEM	Fluentd_collect or		ALERT detected	WARN	19m 3s	Update		elementType: check_fi... hostname: lxorabdb14 ...

Limitations and the possible future

We are only able to send the data in one call to the Zabbix server.
This means we have to use the data to create our LLD discovery and populate our items with values.

This is possible only with the workaround of the calculated item.
This poses limitations on the amount of data we will be able to process.

We are looking into Fluentd ATM for ways to solve this. A possible solution is to fix this with the @copy option.

Another option is to replace Fluentd in a later phase with Zabbix agents and the original scripts by making use of Userparameters.



THANK YOU!



Patrik Uytterhoeven

Senior Consultant & Trainer

 +32 2 255 70 70

 www.open-future.be

 Leuvensesteenweg 643 1930
Zaventem (Belgium)

