Zabbix meets television
Clever use of Zabbix features

IntelliTrend GmbH
www.intellitrend.de

Contact: Wolfgang Alper
wolfgang.alper@intellitrend.de
ZDF – Zweites Deutsche Fernsehen

ZDF - „Second german television“
Some history
• **In 1961**, the federal states established a central **non-profit** television broadcaster „Zweites Deutsches Fernsehen“.

• **In 1963** on April 1, ZDF officially went on air and had reached 61 percent of television viewers.

• On the Internet, a selection of programs is offered via live stream or video-on-demand through the „ZDFmediathek“, which has been in existence since **2001**.

• Since February **2013**, ZDF has been broadcasting its programs around the clock as an Internet livestream.

• As of today ZDF is one of the largest public broadcasters in Europe with permanent bureaus worldwide, and is also present on various platforms like youtube, facebook etc.
ZDF – Monitoring with Zabbix

Let’s get technical
Some clever uses of Zabbix features

A special thanks goes to Mr. Uwe Grunert for the good joint work!
ZDF – Monitoring with Zabbix

Get states from infrastructure monitoring with dynamic severities using LLD

Goal

- Monitor and alert data coming from an external monitoring system that controls infrastructure components such as power generators, transmission stations and the like.

- The external system should automatically define the services to be monitored in Zabbix.

- The external system should automatically define the trigger severity levels to be used in Zabbix for each service.

Challenges

- How can the external system automatically define the severity levels to be used by Zabbix triggers?
ZDF – Monitoring with Zabbix

Get states from infrastructure monitoring with dynamic severities using LLD

**Approach**

- Use Zabbix build in HTTP check to get LLD discovery data.
- Use Zabbix build in HTTP check as a collector get metrics.
- Define item prototypes as dependant items to extract data from collector item.
- Create „smart“ trigger prototypes to respect severity information from LLD discovery data.

Note: HTTP item allows to use zabbix-sender, which is great for testing.
ZDF – Monitoring with Zabbix

Get states from infrastructure monitoring with dynamic severities using LLD

```json
{
    "#NAME": "generator-secondary",
    "#DISPLAYNAME": "Secondary power generator",
    "#DESCRIPTION": "Secondary emergency power generator",
    "#CATEGORY": "Powersupply",
    "#PRIORITY.INFORMATION": -1,
    "#PRIORITY.WARNIING": -1,
    "#PRIORITY.AVERAGE": -1,
    "#PRIORITY.HIGH": 1,
    "#PRIORITY.DISASTER": 2
}
```

Metrics returned by /metrics

In this example, „status“ = 1 refers to a priority of {#PRIORITY.HIGH}.

status = 0 means, no issues.

```json
"generator-primary": {
    "status": 0,
    "message": "Generator is healthy."
},
"generator-secondary": {
    "status": 1,
    "message": "Generator is not working properly."
}
```

LLD returned by /discovery

Each component defines its assignment from the „status“ value to a specific severity level.

A value of -1 means: not used.
ZDF – Monitoring with Zabbix

Get states from infrastructure monitoring with dynamic severities using LLD

Item prototypes

<table>
<thead>
<tr>
<th>Planit</th>
<th>Enabled</th>
<th>Zabbix</th>
<th>Discovery list / Planit Discovery</th>
<th>Item prototypes 2</th>
<th>Trigger prototypes 5</th>
<th>Graph prototypes</th>
<th>Host prototypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>Name</td>
<td>Key</td>
<td>Interval</td>
<td>History</td>
<td>Trends</td>
<td>Type</td>
<td>Create enabled</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>------</td>
<td>---------</td>
<td>---------</td>
<td>--------</td>
<td>-------</td>
<td>--------------</td>
</tr>
<tr>
<td>TPL Planit: Planit Collector: (#DISPLAYNAME) [message]</td>
<td>message[#{NAME}]</td>
<td>90d</td>
<td>Dependend item</td>
<td>Yes</td>
<td>Yes</td>
<td>Application: /#CATEG/...</td>
<td></td>
</tr>
<tr>
<td>TPL Planit: Planit Collector: (#DISPLAYNAME) [status]</td>
<td>status[#{NAME}]</td>
<td>90d</td>
<td>365d</td>
<td>Dependend item</td>
<td>Yes</td>
<td>Yes</td>
<td>Application: /#CATEG/...</td>
</tr>
</tbody>
</table>

Preprocessing 1

<table>
<thead>
<tr>
<th>Step</th>
<th>Name</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>JSONPath</td>
<td>$[#{NAME}][message]</td>
</tr>
</tbody>
</table>

© Copyright 2021 IntelliTrend GmbH • Germany • www.intellitrend.de
ZDF – Monitoring with Zabbix

Get states from infrastructure monitoring with dynamic severities using LLD

Trigger prototypes

<table>
<thead>
<tr>
<th>Severity</th>
<th>Name</th>
<th>Operational data</th>
<th>Expression</th>
<th>Create enabled</th>
<th>Discover</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>planit Service: {#DISPLAYNAME} has Status AVERAGE</td>
<td></td>
<td>last(TPL.Plantl/status[{#NAME}]) = {#PRIO:RITY:AVERAGE}</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Disaster</td>
<td>planit Service: {#DISPLAYNAME} has Status DISASTER</td>
<td></td>
<td>last(TPL.Plantl/status[{#NAME}]) = {#PRIO:RITY:DISASTER}</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>High</td>
<td>planit Service: {#DISPLAYNAME} has Status HIGH</td>
<td></td>
<td>last(TPL.Plantl/status[{#NAME}]) = {#PRIO:RITY:HIGH}</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Information</td>
<td>planit Service: {#DISPLAYNAME} has Status INFORMATION</td>
<td></td>
<td>last(TPL.Plantl/status[{#NAME}]) = {#PRIO:RITY:INFORMATION}</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Warning</td>
<td>planit Service: {#DISPLAYNAME} has Status WARNING</td>
<td></td>
<td>last(TPL.Plantl/status[{#NAME}]) = {#PRIO:RITY:WARNING}</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

The prototype definitions will automatically create specific triggers, depending on the value of the LLD macro for a given service.
ZDF – Monitoring with Zabbix

Get states from infrastructure monitoring with dynamic severities using LLD

The result

<table>
<thead>
<tr>
<th>Time</th>
<th>Severity</th>
<th>Recovery time</th>
<th>Status</th>
<th>Info</th>
<th>Problem</th>
<th>Duration</th>
<th>Ack</th>
<th>Actions</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>18:48:21</td>
<td></td>
<td></td>
<td></td>
<td>Planit</td>
<td>Plant! Service: Elasticsearch Service has Status INFORMATION</td>
<td>1m 6s</td>
<td>No</td>
<td>-</td>
<td>Application: PlanIt</td>
</tr>
<tr>
<td>18:48:21</td>
<td></td>
<td></td>
<td></td>
<td>Planit</td>
<td>Plant! Service: Transmitting station #3 has Status AVERAGE</td>
<td>1m 6s</td>
<td>Yes</td>
<td>1</td>
<td>Application: Radio</td>
</tr>
<tr>
<td>18:48:21</td>
<td></td>
<td></td>
<td></td>
<td>Planit</td>
<td>Plant! Service: Secondary power generator has Status AVERAGE</td>
<td>1m 6s</td>
<td>No</td>
<td>-</td>
<td>Application: Powerup</td>
</tr>
</tbody>
</table>
ZDF – Monitoring with Zabbix

Get states from infrastructure monitoring with dynamic severities using LLD

Well it works, but we can do better
ZDF – Monitoring with Zabbix

Get states from infrastructure monitoring with dynamic severities using LLD

Current solution creates unnecessary triggers for individual components ...
Get states from infrastructure monitoring with dynamic severities using LLD

**LLD-Overrides for the rescue**

<table>
<thead>
<tr>
<th>Overides</th>
<th>Name</th>
<th>Stop processing</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1: PRIORITY_INFORMATION</td>
<td>No</td>
<td>Remove</td>
</tr>
<tr>
<td></td>
<td>2: PRIORITY.WARNING</td>
<td>No</td>
<td>Remove</td>
</tr>
<tr>
<td></td>
<td>3: PRIORITY.AVERAGE</td>
<td>No</td>
<td>Remove</td>
</tr>
<tr>
<td></td>
<td>4: PRIORITY.HIGH</td>
<td>No</td>
<td>Remove</td>
</tr>
<tr>
<td></td>
<td>5: PRIORITY.DISASTER</td>
<td>No</td>
<td>Remove</td>
</tr>
</tbody>
</table>

LLD Overrides were introduced in Zabbix Version 5.0.
ZDF – Monitoring with Zabbix

Get states from infrastructure monitoring with dynamic severities using LLD

LLD-Override configuration – Disable discovery depending on LLD data

Filter: Based on LLD macro content = -1

Condition: Based on Trigger prototype name

```json
{
   ...
   "#{PRIORITY.INFORMATION}": -1,
   ...
}
```
ZDF – Monitoring with Zabbix

Get states from infrastructure monitoring with dynamic severities using LLD

Final result with LLD-Overrides

<table>
<thead>
<tr>
<th>Severity</th>
<th>Value</th>
<th>Name</th>
<th>Operational data</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disaster</td>
<td>OK</td>
<td>Planit Discovery: planit Service. Primary power generator has Status DISASTER</td>
<td>last(Planit LLD/status[generator-primary]) = 2</td>
<td></td>
</tr>
<tr>
<td>High</td>
<td>OK</td>
<td>Planit Discovery: planit Service. Primary power generator has Status HIGH</td>
<td>last(Planit LLD/status[generator-primary]) = 1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Severity</th>
<th>Value</th>
<th>Name</th>
<th>Operational data</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>OK</td>
<td>Planit Discovery: planit Service. Planit Infrastructure Service has Status HIGH</td>
<td>last(Planit LLD/status[plint]) = 2</td>
<td></td>
</tr>
<tr>
<td>Warning</td>
<td>OK</td>
<td>Planit Discovery: planit Service. Planit Infrastructure Service has Status WARNING</td>
<td>last(Planit LLD/status[plint]) = 1</td>
<td></td>
</tr>
</tbody>
</table>
Goal

- Use Zabbix for evaluating error messages from the "Sphinx" application in Graylog (log management) and alert them. Graylog is used for log management only, not for alerting.
- Monitoring the number of errors in user-defined time intervals for different components and alert when a threshold is exceeded.
- Analyse incoming error messages and prepare them for a user friendly output sorted by error types.

Challenges

- How to get the information from Graylog about the Sphinx components (App, Web and WCF Gateway)?
- How to handle certificate problems (DH_KEY_TOO_SMALL / Diffie–Hellman key) due to an outdated version of the installed Graylog server?
- How to sort the error messages coming in "free form" without explicit error types?
ZDF – Monitoring with Zabbix

Approach

- Use Zabbix „external check“ item type to solve the certificate problem.

- Configure three master items to make the HTTP API Get request and retrieve the raw data for each component. All additional information is retrieved via dependent item with preprocessing to save further API requests and to be resource friendly.

- Use Zabbix dependent item with Java-Script preprocessing to parse incoming error messages and sort them by error type. Use dependent item for LLD to create the items for the stats data and also the data for the visualization for each error type.

- Create an user friendly dashboard.

„Sphinx“ application monitoring using Graylog REST API
ZDF – Monitoring with Zabbix

"Sphinx“ application monitoring using Graylog REST API

Item design and dependency using „Sphinx App“ component as an example

External Check
Graylog Stream
“Sphinx App Raw”

Dependent Item (Analyse)
(using Pre-Processing)
"Sphinx App Raw Filtered"

Discovery Item (LLD)
(using Pre-Processing)
"Sphinx App Error LLD"

Item Prototype
(using Pre-Processing)
"Sphinx App Error Count"

Dependent Item
Error Type A Count

Dependent Item
Error Type A List

Dependent Item
Error Type B Count

Dependent Item
Error Type B List

Dependent Item
Error Type … Count

Dependent Item
Error Type … List

Similar setup for other „Sphinx“ components

External Check
Graylog Stream
“Sphinx WCF Raw”

Dependent Item
(using Pre-Processing)
"Sphinx WCF Raw Filtered"

External Check
Graylog Stream
“Sphinx Web Raw”

Dependent Item
(using Pre-Processing)
"Sphinx Web Raw Filtered"
ZDF – Monitoring with Zabbix

„Sphinx“ application monitoring using Graylog REST API

Three master items to make the HTTP Get request and retrieve the data for each application level with defined intervals

```
graylog2zabbix.sh[{$GRAYLOG_USERNAME},{GRAYLOG_PASSWORD},{HOST.CONN},{GRAYLOG_PORT},search/universal/relative?query=name:3Asphinx-app:20AND:20stage:3Aproduction:20AND:20level:3AERROR:20OR:20FATAL)&range=1800&limit=50&filter=streams:3A60000a8c1c09f9862279966e&fields=name:2Clevel:2Cmessage&decorate=true]
```

Application level: App  Application level: WCF  Application level: Web
ZDF – Monitoring with Zabbix

“Sphinx“ application monitoring using Graylog REST API

Simplified JSON output from the unsorted error messages of „Sphinx Web“ via REST API
ZDF – Monitoring with Zabbix

"Sphinx“ application monitoring using Graylog REST API

Zabbix dependent item to analyse the error messages

Pre-Processing Steps:

- Analyse the error messages.
- Define the error type.
- Sort the raw data.
ZDF – Monitoring with Zabbix

“Sphinx” application monitoring using Graylog REST API

Preprocessing steps

<table>
<thead>
<tr>
<th>Preprocessing steps</th>
<th>Name</th>
<th>Parameters</th>
<th>Custom on fail</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>JSONPath</td>
<td>$.messages[*].message.message</td>
<td></td>
<td>Test, Remove</td>
</tr>
<tr>
<td>2</td>
<td>JavaScript</td>
<td>var errorList = JSON.parse(value);...</td>
<td></td>
<td>Test, Remove</td>
</tr>
</tbody>
</table>

© Copyright 2021 IntelliTrend GmbH ● Germany ● www.intellitrend.de
ZDF – Monitoring with Zabbix

„Sphinx“ application monitoring using Graylog REST API

Java script to define the error message type and sort the messages

```javascript
function (value) {
    var errorList = JSON.parse(value);
    // get unique list of error types
    var regex = new RegExp(/(\w+?):\[(\w+)\]/g);
    var result = [];
    for (var i = 0; i < errorList.length; i++) {
        var errorType = regex.exec(errorList[i]);
        if (errorType == null) {
            errorType = "undefined";
        } else {
            errorType = errorType[1];
        }
        // add error to result array if existing or create new array in dict
        if (errorType in result) {
            result[errorType].push(errorList[i]);
        } else {
            result[errorType] = [errorList[i]];
        }
    }
    return JSON.stringify(result);
}
```
ZDF – Monitoring with Zabbix

„Sphinx“ application monitoring using Graylog REST API

Error messages after preprocessing sorted by type

Error types

Error types with errors
ZDF – Monitoring with Zabbix

“Sphinx“ application monitoring using Graylog REST API

Dependent LLD item to define the error message type and sort the raw data
ZDF – Monitoring with Zabbix

“Sphinx“ application monitoring using Graylog REST API

Everyone loves dashboard
ZDF – Monitoring with Zabbix

"Sphinx" application monitoring using Graylog REST API

Everyone loves dashboard

List of active error types per component

Total number of errors per component

Number of errors per error type
ZDF – Monitoring with Zabbix

Monitor something different
ZDF – Monitoring with Zabbix

TV broadcast truck („Übertragungs-Wagen“)
Monitoring a TV broadcast truck („Übertragungs-Wagen“)

**Goal**
- Monitor several metrics from different technologies used in the TV broadcast truck.
- Monitor communication availability and quality between the broadcast truck and the transmitting station.
- Only monitor the broadcast truck when in use.

**Challenges**
- How can false positive alarms be avoided if a broadcast truck can be put into operation spontaneously (without notifying the monitoring team)?
ZDF – Monitoring with Zabbix

Monitoring a TV broadcast truck („Übertragungs-Wagen“)

**Approach**

- Treat a broadcast truck and its components as a host that can be put into maintenance.

- Create a control host (as entity in Zabbix) to monitor the connection states of all broadcasting trucks.

- Create a middleware that implements a smart logic to start/stop monitoring a given broadcasting truck by switching maintenance using the Zabbix API.

- A specific application in the broadcasting truck then tells Zabbix how long to monitor and when to enable maintenance again.
ZDF – Monitoring with Zabbix

Monitoring a TV broadcast truck („Übertragungs-Wagen“)

- Truck control host
- Zabbix Server
- Middleware to manage maintenance via Zabbix API

get status for each truck, decide when to start monitoring

Enabled truck tells Zabbix the monitoring duration. Metrics collection starts

get status for each truck, decide when to start monitoring
ZDF – Monitoring with Zabbix

Monitoring a TV broadcast truck („Übertragungs-Wagen“)

- Trigger - Start Service Truck 001
- Trigger - Start Service Truck 002
- Trigger - Start Service Truck 003
- Trigger - Start Service Truck ...

Zabbix Action - Initiated by truck control host

Middleware Disable maintenance for truck xxx via Zabbix API

Start monitoring Get monitoring „duration“ from truck xxx

Middleware Enable maintenance for truck xxx via Zabbix-API
ZDF – Monitoring with Zabbix

Monitoring a TV broadcast truck („Übertragungs-Wagen“)
Zabbix meets television
Clever use of Zabbix features

Thank You!

IntelliTrend GmbH
www.intellitrend.de

Contact: Wolfgang Alper
wolfgang.alper@intellitrend.de