Going Down!

Using Low-Level Discoveries in practice
Who am I

Raymond Kuiper

- Infrastructure Specialist @ Competa IT
- Zabbix fan since 2006
- zbxtutorials.org
- NLZGG – Dutch Zabbix user group
What is LLD?

“Low-level discovery provides a way to automatically create items, triggers, and graphs for different entities on a computer.”
- the Zabbix manual
What is LLD?

A Low-Level Discovery rule uses:

- a discovery item that returns the discovery data in JSON formatted
- that discovery data on prototypes to create new items triggers and graphs.
What is LLD?

```json
{
  "data": [
    {
      "#FSNAME": "V",
      "#FSTYPE": "ext3"
    },
    {
      "#FSNAME": "/lib/init/rw",
      "#FSTYPE": "tmpfs"
    },
    {
      "#FSNAME": "/dev/shm",
      "#FSTYPE": "tmpfs"
    },
    {
      "#FSNAME": "/home",
      "#FSTYPE": "ext3"
    },
    {
      "#FSNAME": "/tmp",
      "#FSTYPE": "ext3"
    },
    {
      "#FSNAME": "/usr",
      "#FSTYPE": "ext3"
    },
    {
      "#FSNAME": "/var",
      "#FSTYPE": "ext3"
    },
    {
      "#FSNAME": "/sys/fs/fuse/connections",
      "#FSTYPE": "fusectl"
    },
    {
      "#FSNAME": "/sys/vfs/fuse/connections",
      "#FSTYPE": "fusectl"
    }
  ]
}
```
What is LLD?

- JSON Macros
- Prototypes
  - Item
  - Trigger
  - Graph
What is LLD?

These macros can be used for

item prototypes in:
• names
• keys
• SNMP OIDs
• calculated item formulas
• SSH and Telnet scripts
• database monitor item parameters

trigger prototypes in:
• names
• expressions (when referencing an item key prototype)

graph prototypes in:
• names
What is LLD?

The item used for discovery can be *any* item as long as it outputs the JSON format Zabbix expects.
What is LLD?

Items, triggers and graphs that are no longer discovered are cleaned up after a predefined period of time.
Native Zabbix Agent LLD

The Zabbix agent has two types of LLD available by default:

- Filesystem discovery (vfs.fs.discovery)
- Network interface discovery (net.if.discovery)
Native Zabbix Agent LLD

Filesystem discovery (vfs.fs.discovery) returns these macros:

- `{#FSNAME}` - Mount location or drive letter
- `{#FSTYPE}` - Filesystem type (e.g. ext4, vfat)
Native Zabbix Agent LLD

Network interface discovery (net.if.discovery) returns this macro:

- `{#IFNAME}` – Interface name (e.g. eth0, lo)
Native SNMP LLD

The SNMP Discovery:

• can use any SNMP OID as a discovery item (uses an SNMP walk)

• will return these two macros:
  {#SNMPINDEX} – the last part of the OID discovered (after the “.”)
  {#SNMPVALUE} – the value of the OID discovered
Native SNMP LLD

SNMP Walk:
$ snmpwalk -v 2c -c public 192.168.1.1 IF-MIB::ifDescr
IF-MIB::ifDescr.1 = STRING: WAN
IF-MIB::ifDescr.2 = STRING: LAN1
IF-MIB::ifDescr.3 = STRING: LAN2

Zabbix Discovery of IF-MIB::ifDescr:
{#SNMPINDEX} -> 1 {#SNMPVALUE} -> WAN
{#SNMPINDEX} -> 2 {#SNMPVALUE} -> LAN1
{#SNMPINDEX} -> 3 {#SNMPVALUE} -> LAN2
LLD Filters

You can define one regex filter per discovery rule to match macro values used for populating prototypes.

For example:

`^(btrfs|ext2|ext3|ext4|jfs|reiser|xfs|ffs|ufs|jfs|jfs2|vxfs|hfs|ntfs|fat32)$`
Ok, so what can I do with this?
Case 1: FS discovery

Discovery rule

- **Name**: Mounted filesystem discovery
- **Type**: Zabbix agent
- **Key**: vfs.fs.discovery
- **Update interval (in sec)**: 3600
- **Filter Macro**: #FSTYPE
- **Regexp**: @File systems for discovery
- **Description**: Discovery of file systems of different types as defined in global regular expression "File systems for discovery".
- **Status**: Enabled

Flexible intervals

- **Interval**: 50
- **Period**: 1-7,00:00-24:00
- **Add**
- **Keep lost resources period**:
  - **(in days)**: 30

No flexible intervals defined.
Case 1: FS discovery
Case 1: FS discovery
Case 1: FS discovery

![Image of trigger configuration for FS discovery]
Case 2: Windows Services

Need:

Activate triggers for services that are auto started but no longer running.
Case 2: Windows Services

Zabbix Agent has no capability of discovering Windows services

ZBXNEXXT-1368
(Please vote!)
Case 2: Windows Services

My Solution:

Powershell script to grab autostart services from WMI and return their attributes as LLD macros.

( https://raw.github.com/q1x/zabbix-templates/master/service-discovery/servdisc.ps1 )
Case 2: Windows Services

{#SERVICENAME}  The name of the Windows service
{#SERVICEDISPLAY}  The displayname of the Windows service
{#SERVICESTATE}  The state of the Windows service
{#SERVICEDESC}  The Windows service description
Case 2: Windows Services

Template does a discovery automatically started services currently in the running state.

It filters the `{#SERVICESTATE}` macro for the string "Running".

(https://raw.github.com/q1x/zabbix-templates/master/service-discovery/Template_Windows_Service_Discovery.xml)
## Case 2: Windows Services

<table>
<thead>
<tr>
<th>Windows service discovery: Service Windows Update state</th>
<th>Triggers (2)</th>
<th>service_state[wuauserv]</th>
<th>30</th>
<th>90</th>
<th>365</th>
<th>Zabbix agent</th>
<th>Windows Services</th>
<th>Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows service discovery: Service Workstation state</td>
<td>Triggers (2)</td>
<td>service_state[LenmanWorkstation]</td>
<td>30</td>
<td>90</td>
<td>365</td>
<td>Zabbix agent</td>
<td>Windows Services</td>
<td>Enabled</td>
</tr>
<tr>
<td>Windows service discovery: Service Zabbix Agent state</td>
<td>Triggers (2)</td>
<td>service_state[Zabbix Agent]</td>
<td>30</td>
<td>90</td>
<td>365</td>
<td>Zabbix agent</td>
<td>Windows Services</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

### Average

Service Windows Update state: Service Windows Update is not running on `{HOSTNAME}`

Service Workstation is not running on `{HOSTNAME}`

Service Zabbix Agent is not running on `{HOSTNAME}`

<table>
<thead>
<tr>
<th>Average</th>
<th>Windows service discovery: Service Windows Update is not running on <code>{HOSTNAME}</code></th>
<th>{win7test:service_state[wuauserv],last(0)} #0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>Windows service discovery: Service Workstation is not running on <code>{HOSTNAME}</code></td>
<td>{win7test:service_state[LenmanWorkstation],last(0)} #0</td>
</tr>
<tr>
<td>Average</td>
<td>Windows service discovery: Service Zabbix Agent is not running on <code>{HOSTNAME}</code></td>
<td>{win7test:service_state[Zabbix Agent],last(0)} #0</td>
</tr>
</tbody>
</table>

### Service Logs

| Service Windows Update state | 14 Dec 2012 11:36:49 | 0 | - | Graph |
| Service Workstation state   | 14 Dec 2012 11:36:55 | 0 | - | Graph |
| Service Zabbix Agent state  | 14 Dec 2012 11:36:51 | 0 | - | Graph |
Case 3: Linux processes

Question in #zabbix:

“How to monitor CPU usage of separate processes under Linux?”
Case 3: Linux processes

Solution: Template and 2 custom user parameters

- ps.discovery - Returns a list of monitorable processes
- proc.cpu[*] - Calculates CPU usage

(https://github.com/qlx/zabbix-templates/tree/master/process-discovery)
Case 3: Linux processes

ps.discovery returns:

{#PSNAME} - The name of the found process
{#PSUSER} - The user account running the process
Case 3: Linux processes

Template includes items for the number of processes and the memory usage of each process.

(https://raw.github.com/q1x/zabbix-templates/master/service-discovery/Template_Windows_Service_Discovery.xml)
Case 3: Linux processes

<disclaimer>
!!! Highly experimental, use at your own risk !!!
</disclaimer>
Case 3: Linux processes
Case 3: Linux processes

Use filters to limit the number of created items!

Item update interval will hammer the agent!
### Case 3: Linux processes

<table>
<thead>
<tr>
<th>Description</th>
<th>Time</th>
<th>Value</th>
<th>Delta</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU usage of zabbix_agentd processes owned by zabbix</td>
<td>16 Apr 12:05:10</td>
<td>3.6%</td>
<td>-</td>
<td>Grafiek</td>
</tr>
<tr>
<td>CPU usage of zabbix_server processes owned by zabbix</td>
<td>16 Apr 12:05:11</td>
<td>0%</td>
<td>-</td>
<td>Grafiek</td>
</tr>
<tr>
<td>CPU usage of grep processes owned by zabbix</td>
<td>16 Apr 12:05:22</td>
<td>0%</td>
<td>-</td>
<td>Grafiek</td>
</tr>
<tr>
<td>CPU usage of top processes owned by root</td>
<td>16 Apr 12:05:25</td>
<td>0%</td>
<td>-</td>
<td>Grafiek</td>
</tr>
<tr>
<td>CPU usage of apt processes owned by root</td>
<td>16 Apr 12:05:28</td>
<td>0%</td>
<td>-</td>
<td>Grafiek</td>
</tr>
<tr>
<td>Memory usage of xfslogd processes owned by root</td>
<td>16 Apr 12:05:34</td>
<td>0 B</td>
<td>-</td>
<td>Grafiek</td>
</tr>
<tr>
<td>Memory usage of rsyslogd processes owned by syslog</td>
<td>16 Apr 12:05:35</td>
<td>243.63 MB</td>
<td>-</td>
<td>Grafiek</td>
</tr>
<tr>
<td>Memory usage of whoopsie processes owned by whoopsie</td>
<td>16 Apr 12:05:36</td>
<td>183.19 MB</td>
<td>-</td>
<td>Grafiek</td>
</tr>
<tr>
<td>Memory usage of apache2 processes owned by www-data</td>
<td>16 Apr 12:05:37</td>
<td>2.8 GB</td>
<td>+1.25 MB</td>
<td>Grafiek</td>
</tr>
<tr>
<td>Memory usage of php5-fpm processes owned by www-data</td>
<td>16 Apr 12:05:38</td>
<td>871.73 MB</td>
<td>-</td>
<td>Grafiek</td>
</tr>
<tr>
<td>Number of zabbix_agentd processes owned by zabbix</td>
<td>16 Apr 12:06:20</td>
<td>6</td>
<td>-</td>
<td>Grafiek</td>
</tr>
<tr>
<td>Number of zabbix_server processes owned by zabbix</td>
<td>16 Apr 12:06:21</td>
<td>27</td>
<td>-</td>
<td>Grafiek</td>
</tr>
<tr>
<td>Number of grep processes owned by zabbix</td>
<td>16 Apr 12:06:24</td>
<td>1</td>
<td>+1</td>
<td>Grafiek</td>
</tr>
<tr>
<td>Number of top processes owned by root</td>
<td>16 Apr 12:06:27</td>
<td>0</td>
<td>-</td>
<td>Grafiek</td>
</tr>
<tr>
<td>Number of apt processes owned by root</td>
<td>16 Apr 12:06:06</td>
<td>0</td>
<td>-</td>
<td>Grafiek</td>
</tr>
</tbody>
</table>
Sudden Realization

I CAN USE
ANY TYPE OF ITEM
Trapper Discovery

Use zabbix_sender to send new 'discovered' items to Zabbix.

Allows for a very dynamic set of items
Case 4: NetFlow

*Could* be ideal for things like syslog, snmp traps or netflow!

( ZBX-6315 : LLD triggers are deleted immediately if not discovered anymore )
Case 4: NetFlow

“NetFlow is a network protocol developed by Cisco Systems for collecting IP traffic information. NetFlow has become an industry standard for traffic monitoring and is supported on various platforms.”

- Wikipedia
Case 4: NetFlow
Case 4: NetFlow

Using Ncapd and scripting Nfdump and Zabbix_sender, we can push Netflow data to Zabbix.

(http://nfdump.sourceforge.net/)
Case 4: NetFlow

Diagram showing a sequence of events involving Nfcapd and a script, leading to a server or server-like device.
Case 4: NetFlow
Case 4: NetFlow
Case 4: NetFlow

<table>
<thead>
<tr>
<th>Name</th>
<th>Triggers</th>
<th>Key</th>
<th>Interval</th>
<th>History</th>
<th>Trends</th>
<th>Type</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netflow discovery: Traffic flow between 10.10.10.10 and 192.168.100.5</td>
<td>netflow[10.10.10.10 192.168.100.5]</td>
<td>90</td>
<td>365</td>
<td>Zabbix trigger</td>
<td>Netflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Netflow discovery: Traffic flow between 10.10.10.10 and 192.168.100.5</td>
<td>netflow[10.10.10.10 192.168.100.5]</td>
<td>90</td>
<td>365</td>
<td>Zabbix trigger</td>
<td>Netflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Netflow discovery: Traffic flow between 10.10.10.10 and 192.168.100.5</td>
<td>netflow[10.10.10.10 192.168.100.5]</td>
<td>90</td>
<td>365</td>
<td>Zabbix trigger</td>
<td>Netflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Netflow discovery: Traffic flow between 10.10.10.10 and 192.168.100.5</td>
<td>netflow[10.10.10.10 192.168.100.5]</td>
<td>90</td>
<td>365</td>
<td>Zabbix trigger</td>
<td>Netflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Netflow discovery: Traffic flow between 10.10.10.10 and 192.168.100.5</td>
<td>netflow[10.10.10.10 192.168.100.5]</td>
<td>90</td>
<td>365</td>
<td>Zabbix trigger</td>
<td>Netflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Netflow discovery: Traffic flow between 10.10.10.10 and 192.168.100.5</td>
<td>netflow[10.10.10.10 192.168.100.5]</td>
<td>90</td>
<td>365</td>
<td>Zabbix trigger</td>
<td>Netflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Netflow discovery: Traffic flow between 10.10.10.10 and 192.168.100.5</td>
<td>netflow[10.10.10.10 192.168.100.5]</td>
<td>90</td>
<td>365</td>
<td>Zabbix trigger</td>
<td>Netflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Netflow discovery: Traffic flow between 10.10.10.10 and 192.168.100.5</td>
<td>netflow[10.10.10.10 192.168.100.5]</td>
<td>90</td>
<td>365</td>
<td>Zabbix trigger</td>
<td>Netflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Netflow discovery: Traffic flow between 10.10.10.10 and 192.168.100.5</td>
<td>netflow[10.10.10.10 192.168.100.5]</td>
<td>90</td>
<td>365</td>
<td>Zabbix trigger</td>
<td>Netflow</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Case 4: NetFlow
Conclusion

Low-Level Discovery:

• Makes SNMP life easier
• Makes FS and Network items a breeze
• Opens up a whole world of new possibilities (get creative!)
Thanks for listening!

Questions?