

3 Low-level discovery

Overview

Low-level discovery provides a way to automatically create items, triggers, and graphs for different entities on a computer. For instance, Zabbix can automatically start monitoring file systems or network interfaces on your machine, without the need to create items for each file system or network interface manually. Additionally it is possible to configure Zabbix to remove unneeded entities automatically based on actual results of periodically performed discovery.

A user can define their own types of discovery, provided they follow a particular JSON protocol.

The general architecture of the discovery process is as follows.

First, a user creates a discovery rule in “Configuration” → “Templates” → “Discovery” column. A discovery rule consists of (1) an item that discovers the necessary entities (for instance, file systems or network interfaces) and (2) prototypes of items, triggers, and graphs that should be created based on the value of that item.

An item that discovers the necessary entities is like a regular item seen elsewhere: the server asks a Zabbix agent (or whatever the type of the item is set to) for a value of that item, the agent responds with a textual value. The difference is that the value the agent responds with should contain a list of discovered entities in a specific JSON format. While the details of this format are only important for implementers of custom discovery checks, it is necessary to know that the returned value contains a list of macro → value pairs. For instance, item “net.if.discovery” might return two pairs: “{#IFNAME}” → “lo” and “{#IFNAME}” → “eth0”.

Return values of a low-level discovery rule are limited to 2048 bytes on a Zabbix proxy run with IBM DB2 database. This limit does not apply to Zabbix server as return values are processed without being stored in a database.

These macros are used in names, keys and other prototype fields where they are then substituted with the received values for creating real items, triggers, graphs or even hosts for each discovered entity. See the full list of [options](#) for using LLD macros.

When the server receives a value for a discovery item, it looks at the macro → value pairs and for each pair generates real items, triggers, and graphs, based on their prototypes. In the example with “net.if.discovery” above, the server would generate one set of items, triggers, and graphs for the loopback interface “lo”, and another set for interface “eth0”.

For more detail on the process described above see the following sections with how-tos for performing all types of out-of-the-box discovery:

- discovery of [file systems](#);
- discovery of [network interfaces](#);
- discovery of [CPUs and CPU cores](#);
- discovery of [SNMP OIDs](#);
- discovery of [JMX objects](#);
- discovery using [ODBC SQL queries](#);
- discovery of [Windows services](#);
- discovery of [host interfaces](#) in Zabbix.

For more detail on the JSON format for discovery items and an example of how to implement your own file system discoverer as a Perl script, see [creating custom LLD rules](#).

Multiple LLD rules for same item

Since Zabbix agent version 3.2 it is possible to define several low-level discovery rules with the same discovery item.

To do that you need to define the Alias agent [parameter](#), allowing to use altered discovery item keys in different discovery rules, for example `vfs.fs.discovery[foo]`, `vfs.fs.discovery[bar]`, etc.

Creating custom LLD rules

It is also possible to create a completely custom LLD rule, discovering any type of entities - for example, databases on a database server.

To do so, a custom item should be created that returns JSON, specifying found objects and optionally - some properties of them. The amount of macros per entity is not limited - while the built-in discovery rules return either one or two macros (for example, two for filesystem discovery), it is possible to return more.

The required JSON format is best illustrated with an example. Suppose we are running an old Zabbix 1.8 agent (one that does not support “`vfs.fs.discovery`”), but we still need to discover file systems. Here is a simple Perl script for Linux that discovers mounted file systems and outputs JSON, which includes both file system name and type. One way to use it would be as a UserParameter with key “`vfs.fs.discovery_perl`”:

```
#!/usr/bin/perl

$first = 1;

print "{\n";
print "\t\"data\": [\n\n";

for (`cat /proc/mounts`)
{
    ($fsname, $fstype) = m/\S+ (\S+) (\S+)/;

    print "\t,\n" if not $first;
    $first = 0;

    print "\t{\n";
    print "\t\t\"{#FSNAME}\" : \"$fsname\", \n";
    print "\t\t\"{#FSTYPE}\" : \"$fstype\" \n";
    print "\t}\n";
}
```

```

}

print "\n\t]\n";
print "]\n";

```

Allowed symbols for LLD macro names are **0-9** , **A-Z** , **_** , **.**

Lowercase letters are not supported in the names.

An example of its output (reformatted for clarity) is shown below. JSON for custom discovery checks has to follow the same format.

```

{
  "data": [
    { "#FSNAME": "/", "#FSTYPE": "rootfs" },
    { "#FSNAME": "/sys", "#FSTYPE": "sysfs" },
    { "#FSNAME": "/proc", "#FSTYPE": "proc" },
    { "#FSNAME": "/dev", "#FSTYPE": "devtmpfs" },
    { "#FSNAME": "/dev/pts", "#FSTYPE": "devpts" },
    { "#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" }
  ]
}

```

Then, in the discovery rule's "Filter" field, we could specify "{#FSTYPE}" as a macro and "rootfs|ext3" as a regular expression.

You don't have to use macro names FSNAME/FSTYPE with custom LLD rules, you are free to use whatever names you like.

Using LLD macros in user macro contexts

User macros [with context](#) can be used to accomplish more flexible thresholds in trigger expressions. Different thresholds may be defined on user macro level and then used in trigger constants depending on the discovered context. Discovered context appears when the [low-level discovery macros](#) used in the macros are resolved to real values.

To illustrate we can use data from the example above and assume that the following file systems will be discovered: /, /home, /tmp, /usr, /var.

We may define a free-disk-space trigger prototype for a host, where the threshold is expressed by a user macro with context:

```
{host:vfs.fs.size[#{FSNAME},pfree].last()}<{$LOW_SPACE_LIMIT: " #{FSNAME} "}
```

Then add user macros:

- `{$LOW_SPACE_LIMIT} 10`
- `{$LOW_SPACE_LIMIT:/home} 20`
- `{$LOW_SPACE_LIMIT:/tmp} 50`

Now, once the file systems are discovered, events will be generated if `/`, `/usr` and `/var` filesystems have less than **10%** of free disk space, the `/home` filesystem - less than **20%** of free disk space or the `/tmp` filesystem - less than **50%** of free disk space.

LLD macros are not supported inside of user macro contexts in [trigger function parameters](#).

From: <https://www.zabbix.com/documentation/3.4/> - **Zabbix Documentation 3.4**

Permanent link: https://www.zabbix.com/documentation/3.4/manual/discovery/low_level_discovery?rev=1499848238

Last update: **2017/07/12 08:30**

