

3 Низкоуровневое обнаружение

Обзор

Низкоуровневое обнаружение (LLD) даёт возможность автоматического создания элементов данных, триггеров и графиков для различных объектов на компьютере. Например, Zabbix может автоматически начать мониторить файловые системы или сетевые интерфейсы с вашего устройства, без необходимости создания вручную элементов данных для каждой файловой системы или сетевого интерфейса. Кроме того, в Zabbix имеется возможность настроить удаление ненужных объектов, основываясь на фактических результатах периодически выполняемого обнаружения.

Пользователь имеет возможность определить свои собственные типы обнаружения, обеспечив их функционирование согласно спецификации JSON протокола.

Общая архитектура процессов обнаружения заключается в следующем.

Сначала, пользователь создает правило обнаружения в “Настройка” → “Шаблоны” → колонка “Обнаружение”. Правило обнаружения состоит из (1) элемента данных, который осуществляет обнаружение необходимых объектов (например, файловые системы или сетевые интерфейсы) и (2) прототипов элементов данных, триггеров и графиков, которые должны быть созданы на основании полученных значений этого элемента данных.

Элемент данных, который осуществляет обнаружение необходимых объектов, подобен обычным элементам данных, которые видны в других местах: Zabbix сервер запрашивает у Zabbix агента (или любой другой указанный тип элемента данных) значение этого элемента данных, и агент отвечает текстовым значением. Разница в том, что значение, которое возвращает агент, должно содержать список обнаруженных объектов в специальном JSON формате. Хотя детали этого формата важны только для создателей собственных проверок обнаружения, всё же всем необходимо знать, что возвращаемое значение содержит список из пар: макрос → значение. Например, элемент данных “net.if.discovery” может вернуть две пары: “{#IFNAME}” → “lo” и “{#IFNAME}” → “eth0”.

Zabbix прокси на IBM DB2 базе данных имеет ограничение в 2048 байт на возвращаемое значение правил низкоуровневого обнаружения. Это ограничение не распространяется на Zabbix сервер, так как возвращаемые значения обрабатываются без предварительной записи в базу данных.

Эти макросы затем используются в именах, ключах и в других полях прототипов, которые являются основой для создания реальных элементов данных, триггеров и графиков каждому обнаруженному объекту. Смотрите полный список [опций](#) по использованию макросов в низкоуровневом обнаружении.

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.

1 Создание пользовательских LLD правил

Также имеется возможность создать полностью пользовательское правило низкоуровневого обнаружения, для обнаружения любого типа объектов - к примеру, баз данных на сервере баз данных.

Чтобы это сделать, необходимо создать пользовательский элемент данных, который будет возвращать JSON, определяющий найденные объекты и опционально - некоторые свойства этих объектов. Количество макросов на объект не ограничено - в то время как встроенные правила обнаружения возвращают либо один, либо два макроса (например, два в случае обнаружения файловых систем), имеется возможность возвращать больше.

Требуемый JSON формат лучше всего иллюстрируется в примере. Предположим, что мы оставим старый Zabbix агент версии 1.8 (который не поддерживает "vfs.fs.discovery"), но нам также нужно обнаруживать файловые системы. Вот простой Perl скрипт для Linux, который обнаруживает примонтированные файловые системы и выдает на выходе данные JSON, в которых включено и имя, и тип файловой системы. Одним из способов его использования является UserParameter с ключем "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";

```

Допустимыми символами в именах макросов низкоуровневых правил обнаружения являются **0-9** , **A-Z** , **_** , **.**

Буквы в нижнем регистре в именах не поддерживаются.

Пример его вывода (переформатирован для наглядности) представлен ниже. JSON данные от пользовательской проверки обнаружения следуют такому же формату.

```

{
  "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" }
  ]
}

```

Тогда, в правилах обнаружения в поле “Фильтр” мы можем указать “{#FSTYPE}”, как макрос, и “rootfs|ext3”, как регулярное выражение.

Вы не обязаны использовать имена макросов FSNAME/FSTYPE в пользовательских правилах низкоуровневого обнаружения, вы можете использовать любые другие имена, которые вам нравятся.

2 Использование макросов LLD в контекстах пользовательских макросов

Мы будем использовать данные из последнего примера для иллюстрации использования макроса LLD в контекстах [пользовательских макросов](#). Исходя из примера, будут обнаружены следующие файловые системы: /, /home, /tmp, /usr и /var.

Добавьте прототип триггера свободного места на диске к узлу сети:

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

И добавьте макросы:

- {LOW_SPACE_LIMIT} **10**
- {LOW_SPACE_LIMIT:/home} **20**
- {LOW_SPACE_LIMIT:/tmp} **50**

Тогда события сгенерируются, когда на файловых системах /, /usr и /var станет свободного места на диске меньше чем **10%**, файловой системе /tmp станет свободного места на диске менее чем **50%** или на файловой системе /home станет свободного места на диске менее чем **20%**.

From: <https://www.zabbix.com/documentation/4.0/> - **Zabbix Documentation 4.0**

Permanent link: https://www.zabbix.com/documentation/4.0/ru/manual/discovery/low_level_discovery?rev=1505328236

Last update: **2017/10/06 14:12**

