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”.

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.

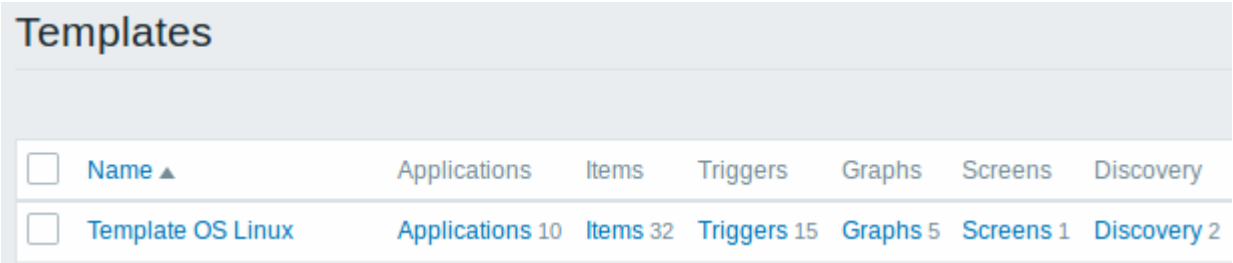
Когда сервер получает значение элемента данных обнаружения, он смотрит на пару макрос → значение и для каждой пары создает реальные элементы данных, триггеров и графиков, основанных на их прототипах. В приведенном выше примере с “net.if.discovery”, сервер будет создавать один набор элементов данных, триггеров и графиков для локального интерфейса “lo” и другой набор для интерфейса “eth0”.

Настройка низкоуровневого обнаружения

Мы проиллюстрируем низкоуровневое обнаружение на примере обнаружения файловых систем.

Для настройки обнаружения, выполните следующее:

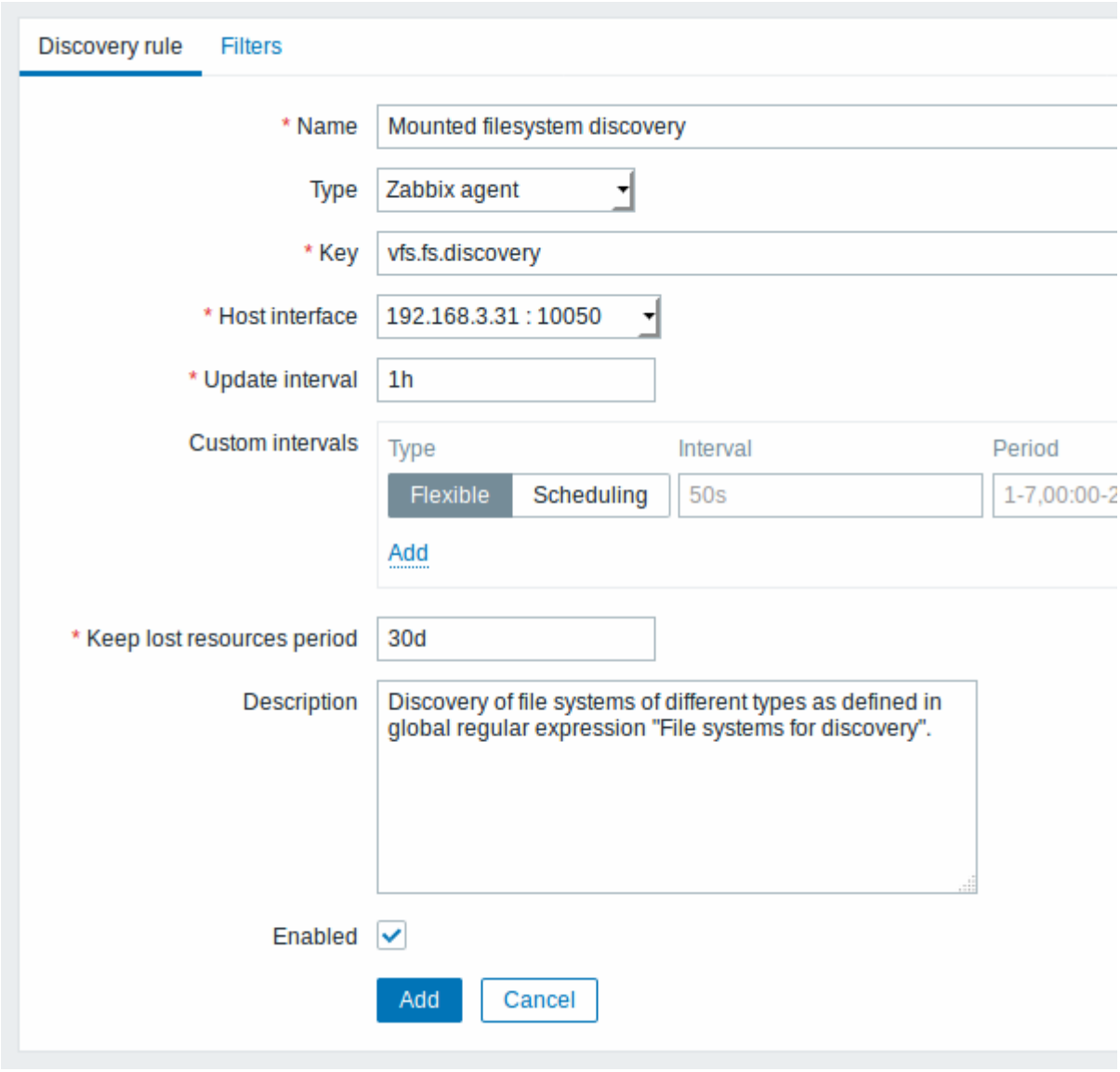
- Перейдите в: *Настройка* → *Шаблоны*
- Нажмите на *Обнаружение* в строке с соответствующим шаблоном



- Нажмите на *Создать правило обнаружения* в верхнем правом углу экрана
- Заполните диалог правила обнаружение необходимыми деталями

Правило обнаружения

Вкладка **Правило обнаружения** содержит общие атрибуты правила обнаружения:

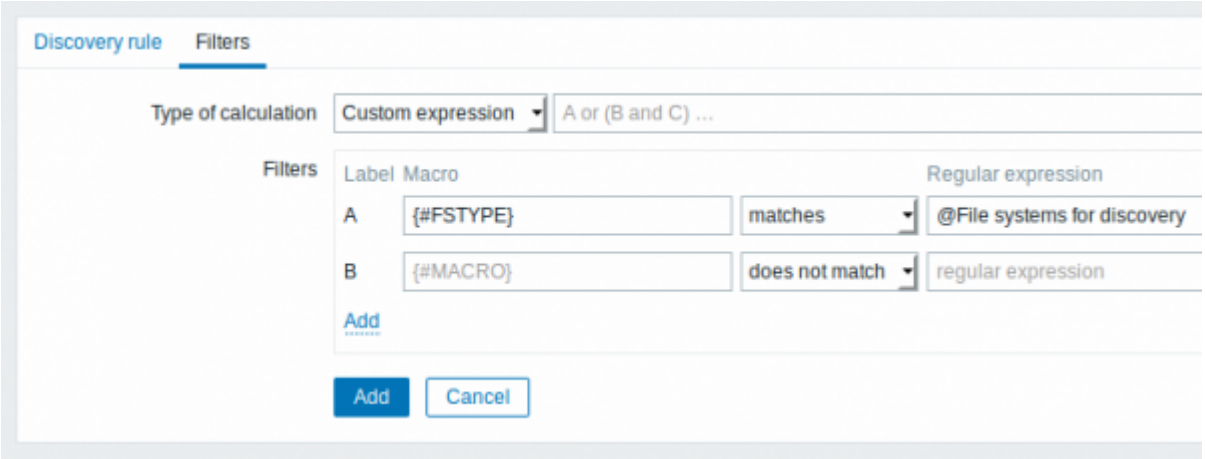


Все обязательные поля ввода отмечены красной звёздочкой.

Параметр	**Описание**
Имя	Имя правила обнаружения.
Тип	Тип проверки выполняемого обнаружения; должен быть *Zabbix агент* или *Zabbix агент (активный)* при обнаружении файловых систем.
Ключ	Элемент данных “vfs.fs.discovery” уже встроен в Zabbix агент начиная с версии 2.0 для многих платформах (для получения более детальных сведений смотрите список поддерживаемых ключей элементов данных), который возвращает список файловых систем, присутствующих в компьютере, и их типы в формате JSON.
Интервал обновления	Это поле задает как часто Zabbix выполняет обнаружение. В начале, когда вы только настраиваете обнаружение файловых систем, вы можете указать маленький интервал, но как только вы удостоверитесь что всё работает, вы можете установить его в 30 минут или более, потому что обычно файловые системы не меняются очень часто. Поддерживаются суффиксы времени, например 30s, 1m, 2h, 1d, начиная с Zabbix 3.4.0. Пользовательские макросы поддерживаются начиная с Zabbix 3.4.0. *Обратите внимание*: Если укажите значение равное '0', элемент данных не будет обрабатываться. Однако, если также существует переменный интервал с ненулевым значением, элемент данных будет обрабатываться в течении действия переменного интервала. *Обратите внимание*, что уже созданное правило обнаружение можно выполнить незамедлительно нажатием кнопки *Проверить сейчас* .
Пользовательские интервалы	Вы можете создавать пользовательские правила проверки элемента данных: **Гибкий** - создание исключений из Интервала обновления (интервал с другой частотой обновления) **По расписанию** - создание пользовательского расписания проверки. Для получения более подробной информации смотрите Пользовательские интервалы. Проверка по расписанию поддерживается начиная с Zabix 3.0.0.
Период сохранения потерянных ресурсов	Это поле позволяет вам указать как много дней обнаруженный объект будет храниться (не будет удален), как только его состояние обнаружения станет “Не обнаруживается более” (мин 1 час, макс 25 лет). Поддерживаются суффиксы времени, например 30s, 1m, 2h, 1d, начиная с Zabbix 3.4.0. Пользовательские макросы поддерживаются начиная с Zabbix 3.4.0. *Обратите внимание:* Если значение равно “0”, объекты будут удалены сразу. Использование значения “0” не рекомендуется, так как простое ошибочное изменение фильтра может закончится тем, что объект будет удален вместе со всеми данными истории.
Описание	Введите описание.
Состояние	Если отмечено, правило будет обрабатываться.

Фильтр правила обнаружения

Вкладка **Фильтры** содержит определения фильтрации правила обнаружения:



Параметр	Описание
Тип вычисления	Доступны следующие опции расчета фильтров: **И** - должны выполниться все фильтры; **Или** - достаточно выполнения одного фильтра; **И/Или** - используется *И* для разных имен макросов и *Или* с одинаковым именем макроса; **Пользовательское выражение** - появляется возможность указать пользовательское вычисление фильтров. Формула должна включать в себя все фильтры из списка. Ограничено 255 символами.
Фильтры	Фильтр можно использовать только для генерирования реальных элементов данных, триггеров и графиков конкретных файловых систем. Ожидается использование Perl Compatible Regular Expression (PCRE). Например, если вы заинтересованы только в файловых системах C:, D: и E:, вы можете поместить {#FSNAME} в поле “Макрос” и регулярное выражение "^C|^D|^E" в текстовые поля “Регулярное выражение”. Фильтрация также возможна по типам файловых систем, при использовании макроса {#FSTYPE} (например, "^ext|^reiserfs") и по типу диска (поддерживается только Windows агентов), используя макрос {#FSDRIVETYPE} (например, "fixed"). Вы можете ввести в поле “Регулярное выражение” регулярное выражение или ссылку на глобальное регулярное выражение. Для проверки регулярного выражения вы можете использовать “grep -E”, например: `for f in ext2 nfs reiserfs smbfs; do echo $f | grep -E '^ext|^reiserfs' || echo "SKIP: $f"; done` Макрос {#FSDRIVETYPE} на Windows поддерживается начиная с Zabbix **3.0.0**. Определение нескольких фильтров поддерживается начиная с **2.4.0**. Обратите внимание, что если какой-то макрос из фильтра пропущен в ответе, найденный объект будет игнорироваться. Выпадающее меню в фильтре представлены два значения задать, которые можно использовать для соответствия регулярному выражению или наоборот, отсутствию соответствия.

Чтобы обнаружение сработало корректно, база данных Zabbix в MySQL должна быть создана чувствительной к регистру, если имена файловых систем различаются только по регистру.
Ошибка или опечатка в регулярном выражении, которое используется в LLD правиле, может привести к удалению тысяч элементов конфигурации, данных истории и событий на большом количестве узлов сети. Например, некорректное регулярное выражение “File systems for discovery” может привести к удалению тысяч элементов данных, триггеров, данных истории и событий.

История правил обнаружения не сохраняется.

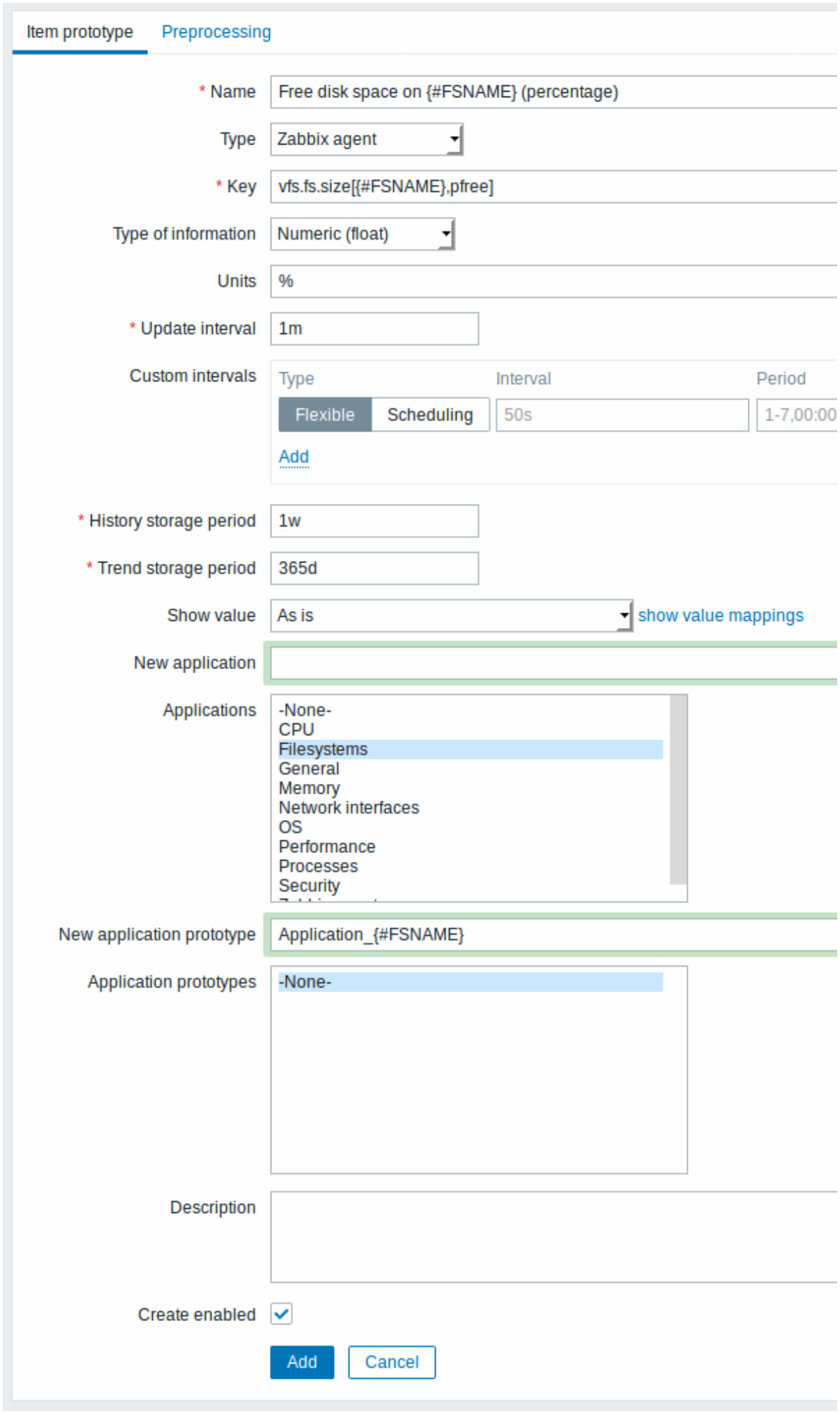
Кнопки диалога

Кнопки в нижней части диалога позволяют выполнить несколько видов операций.

Add	Добавление правила обнаружения. Эта кнопка доступна только для новых правил обнаружения.
Update	Обновление свойств правила обнаружения. Эта кнопка доступна только для уже существующих правил обнаружения.
Clone	Создание другого правила обнаружения на основе свойств текущего правила обнанужения.
Check now	Выполнение немедленного обнаружения на основе правила обнаружения. Правило обнаружения должно существовать. Смотрите более подробную информацию. *Обратите внимание*, что когда обнаружение выполняется немедленно, кэш конфигурации не обновляется, поэтому на результат не повлияют совсем недавние изменения настроек правила обнаружения.
Delete	Удаление правила обнаружения.
Cancel	тмена изменения свойств правила обнаружения.

Прототипы элементов данных

Как только правило будет создано, перейдем к элементам данных этого правила и нажмем “Создать прототип”, чтобы создать прототип элементов данных. Обратите внимание на то, как используется макрос {#FSNAME}, где требуется указать имя файловой системы. Когда правило будет обрабатываться, этот макрос будет заменен обнаруженной файловой системой.


Item prototype
Preprocessing
* Name
Free disk space on {#FSNAME} (percentage)
Type
Zabbix agent
* Key
vfs.fs.size[{#FSNAME},pfree]
Type of information
Numeric (float)
Units
%
* Update interval
1m
Custom intervals
Type
Interval
Period
Flexible
Scheduling
50s
1-7,00:00
Add
* History storage period
1w
* Trend storage period
365d
Show value
As is
show value mappings
New application
Applications
-None-
CPU
Filesystems
General
Memory
Network interfaces
OS
Performance
Processes
Security
New application prototype
Application_{#FSNAME}
Application prototypes
-None-
Description
Create enabled
Add
Cancel

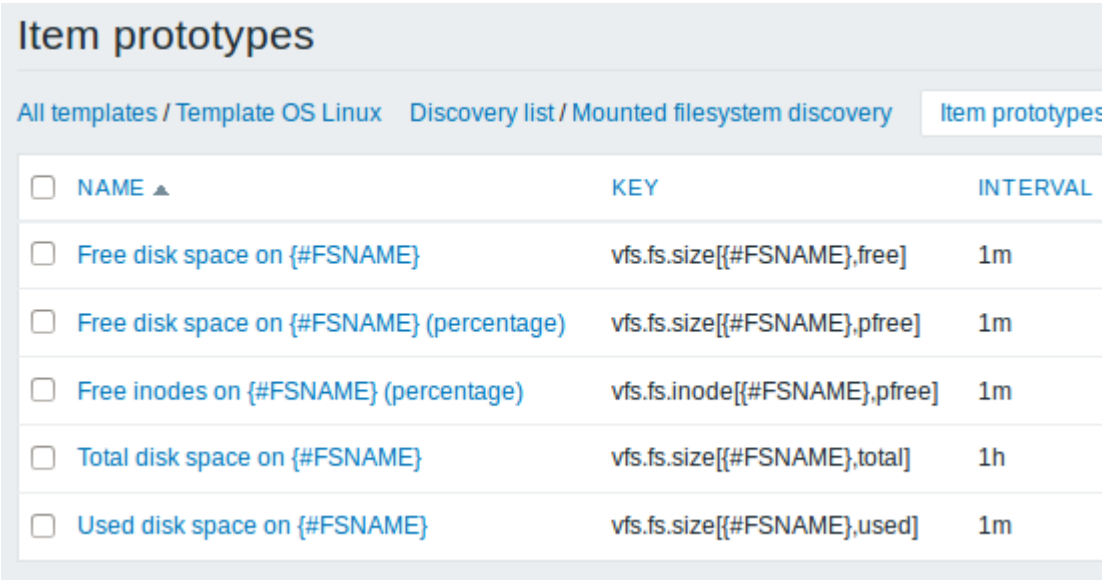
Можно использовать макросы низкоуровневого обнаружения и пользовательские макросы в настройках прототипа элементов данных и в параметрах предварительной обработки значений элемента данных.

Контекстное экранирование макросов низкоуровневого обнаружения для безопасного их использования в регулярных выражениях и параметрах предварительной обработки XPath.

Специфичные для прототипов элементов данных атрибуты:

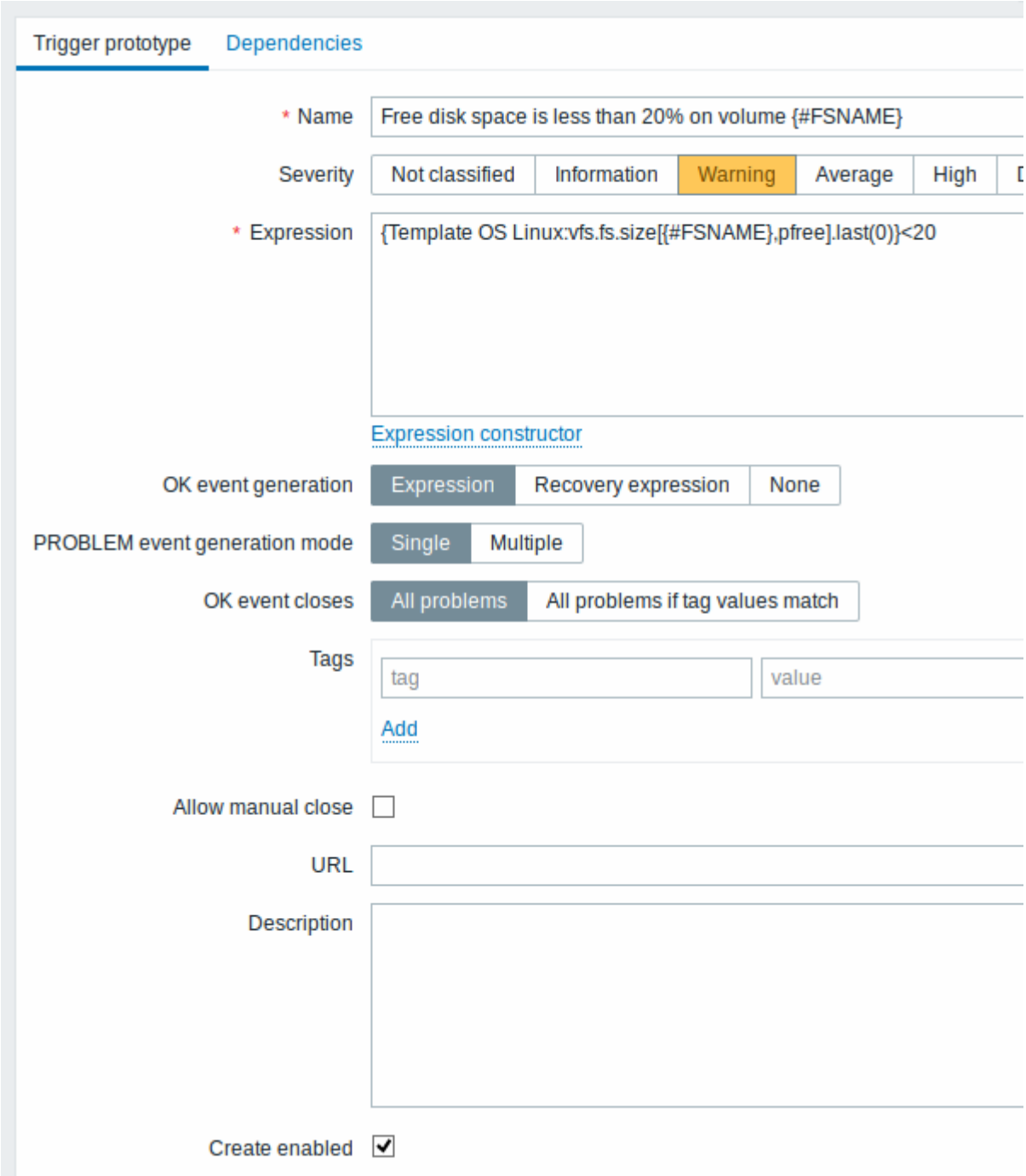
Параметр	**Описание**
Новый прототип группы элементов данных	Вы можете задать новый прототип группы элементов данных. В свойствах группы элементов данных вы можете использовать макросы низкоуровневого обнаружения, которые, после выполнения обнаружения, будут заменены реальными значениями при создании групп элементов данных, которые специфичны для обнаруженного объекта. Смотрите также заметки по обнаружению групп элементов данных для получения более подробной информации.
Прототипы групп элементов данных	Выберите из существующих прототипов групп элементов данных.
Создать активированным	Если выбрано, элемент данных будет создан в активированном состоянии. Если не выбрано, элемент данных будет добавлен как обнаруженный объект, но в деактивированном состоянии.

Мы можем создать несколько прототипов элементов данных для каждой интересующей нас характеристики файловой системы:



Прототипы триггеров

Мы создадим прототипы триггеров похожим способом как и прототипы элементов данных:



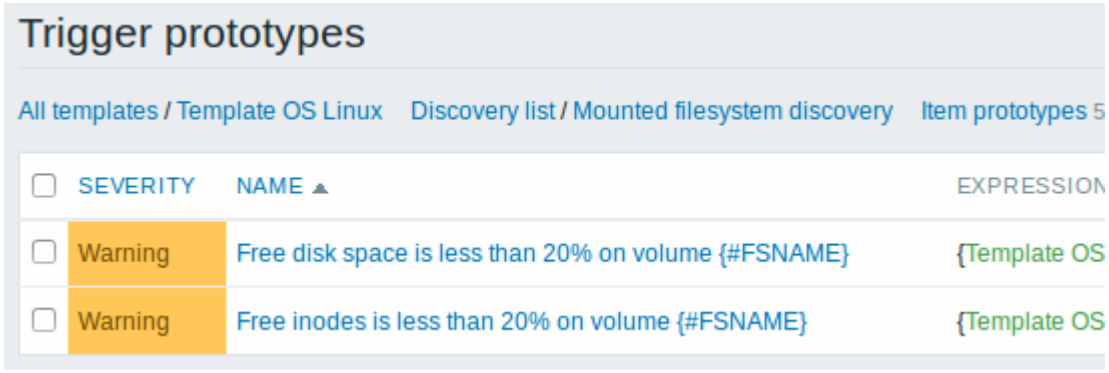
Специфичные для прототипов триггеров атрибуты:

Параметр	**Описание**
Создать активированным	Если выбрано, триггер будет создан в активированном состоянии. Если не выбрано, триггер будет добавлен как обнаруженный объект, но в деактивированном состоянии.

Когда будут созданы реальные триггера из их прототипов, возможно потребуется большая гибкость чем использованная константа ('20' в нашем примере) для сравнения в выражении. Смотрите каким образом пользовательские макросы с контекстом могут быть полезны для получения подобной гибкости.

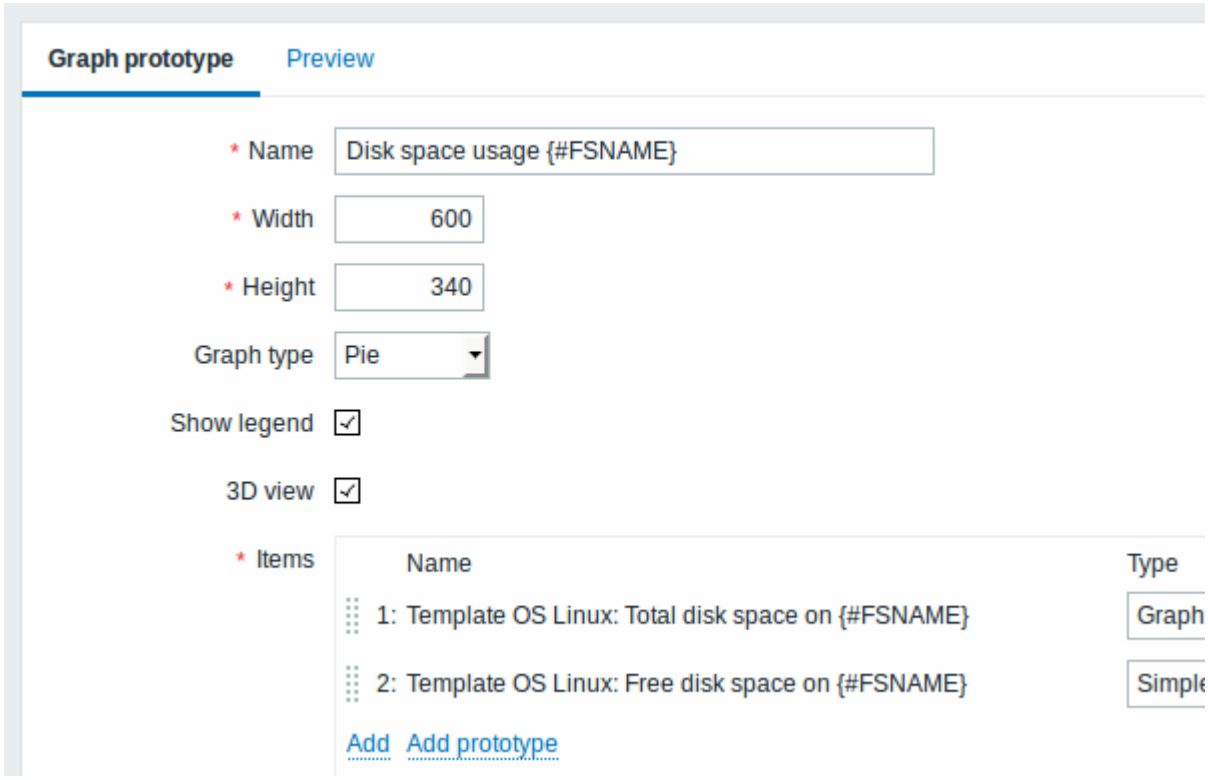
Также вы можете задать зависимости между прототипами триггеров (поддерживается начиная с Zabbix 3.0). Чтобы это сделать, перейдите на вкладку *Зависимости*. Прототип триггеров может зависеть от другого прототипа триггеров из этого же правила

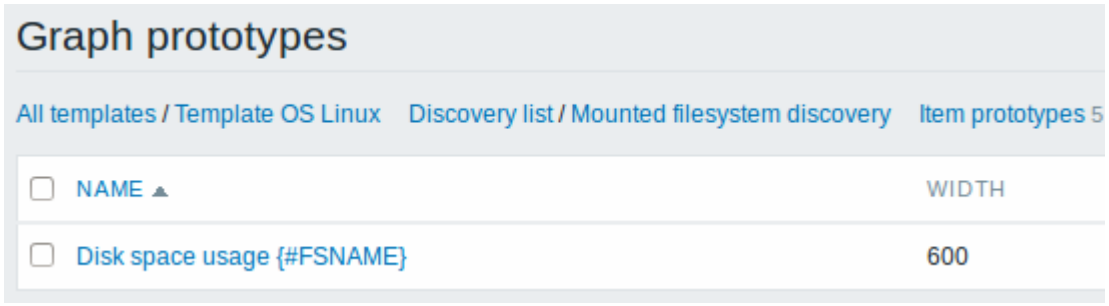
низкоуровневого обнаружения (LLD) или от обычного триггера. Прототип триггеров не может зависеть от прототипа триггеров из другого правила LLD и от триггера созданного другим прототипом триггеров. Прототип триггеров узла сети не может зависеть от триггера из шаблона.



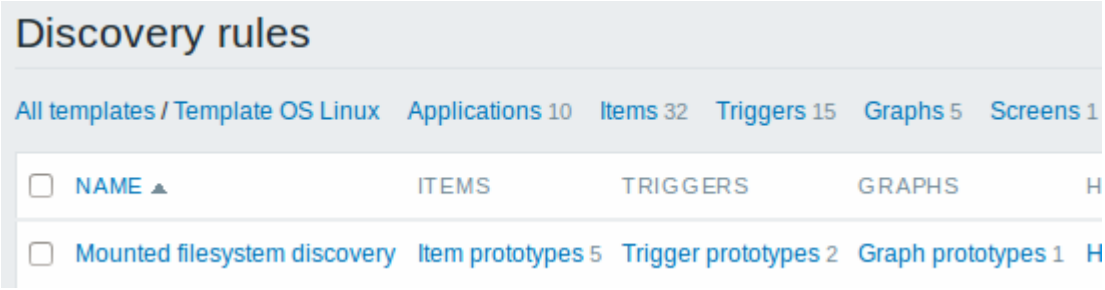
Прототипы графиков

Мы также можем создать прототипы графиков:





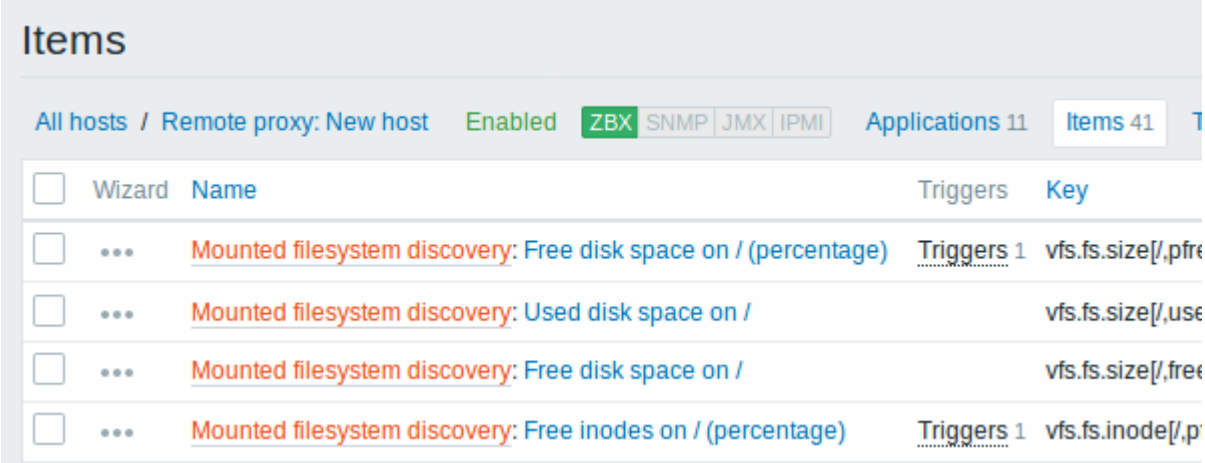
В конце концов, мы создали правило обнаружения, которое выглядит как видно ниже. Оно имеет пять прототипов элементов данных, два прототипа триггеров и один прототип графиков.



Обратите внимание: Для получения информации по настройке прототипов узлов сети, смотрите в разделе мониторинга виртуальных машин о настройке прототипов узлов сети.

Обнаруженные объекты

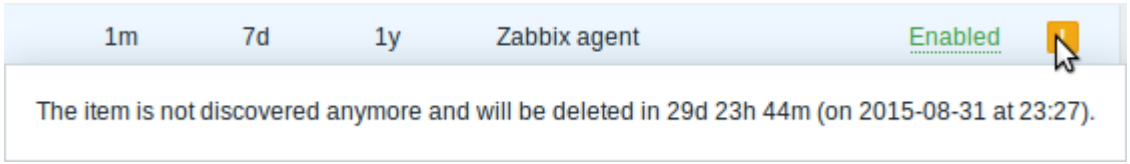
Представленные снимки экрана ниже иллюстрируют как выглядят уже обнаруженные элементы данных, триггера и графики в настройке узла сети. Обнаруженные объекты имеют префикс ссылку золотистого цвета, которая ведет к правилу обнаружения, создавшего эти объекты.



Обратите внимание, что обнаруженные объекты не будут созданы в случае, если объекты с такими же условиями уникальности уже существуют, например, элемент данных с таким же ключем или график с таким же именем.

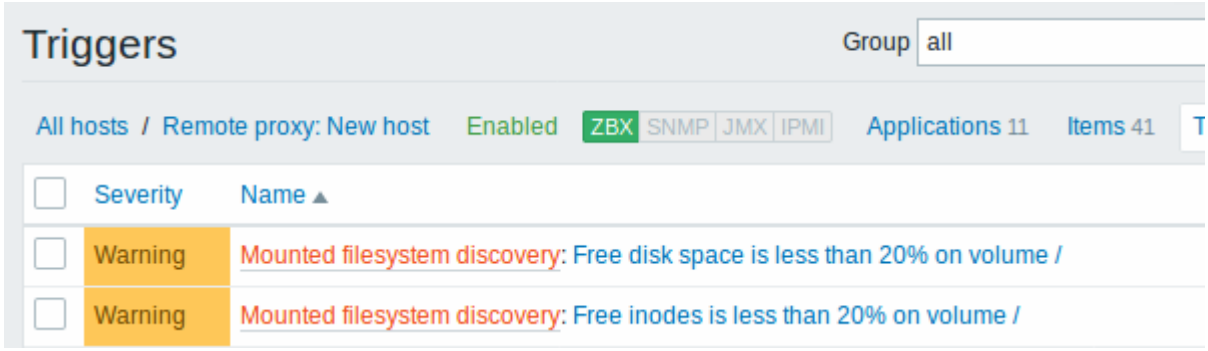
Элементы данных (а также, триггеры и графики) созданые с помощью низкоуровневого правила обнаружения невозможно удалить вручную. Тем не менее, они будут удалены автоматически, если обнаруженный объект (файловая система, интерфейс и т.д.) более не обнаруживается (или более не попадает под фильтр). В этом случае они будут удалены спустя некоторое количество дней указанное в поле *Период сохранения потерянных ресурсов*.

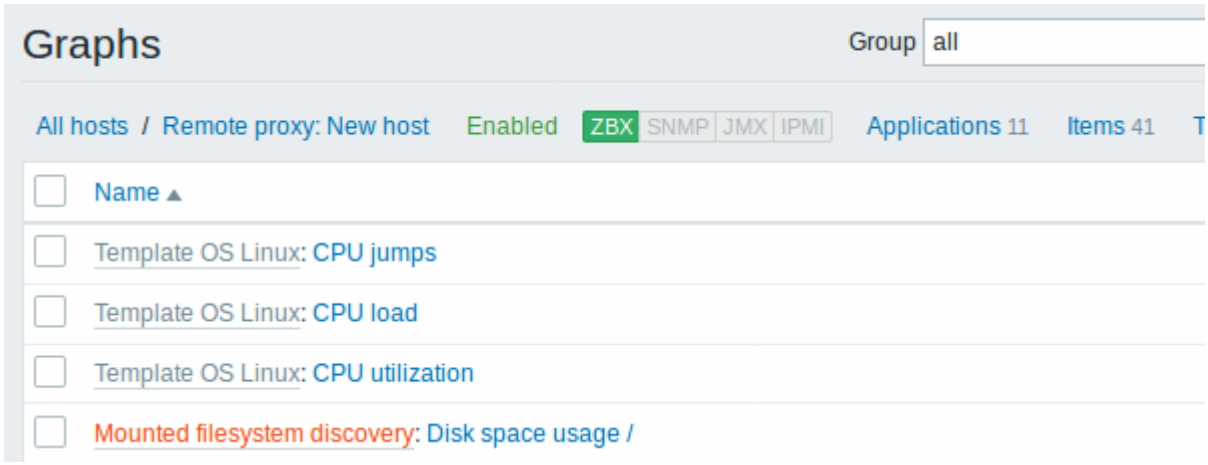
Когда обнаруженный объект становится 'Более не обнаруживается', в списке элементов данных будет отображаться оранжевый индикатор времени жизни. Переместите курсор мыши на этот индикатор и вы увидите сообщение с количеством дней до момента удаления элемента данных.



Если объекты помечены на удаление, но не были удалены в назначенное время (деактивировано правило обнаружения или элемент данных узла сети), они удалятся при следующем выполнении правила обнаружения.

Объекты, которые содержат другие объекты, которые помечены на удаление, не будут обновлены, если будут изменены на уровне правила обнаружения. Например, триггеры на основе LLD не будут обновлены, если они содержат элементы данных, которые помечены на удаление.





Другие типы обнаружения

Для получения более детальных сведений и инструкций по остальным типам доступных обнаружений сморите следующие разделы:

- обнаружение сетевых интерфейсов;
- обнаружение CPU и ядер CPU;
- обнаружение SNMP OID'ов;
- обнаружение JMX объектов;
- обнаружение с использованием ODBC SQL запросов;
- обнаружение Windows служб;
- обнаружение интерфейсов хостов в Zabbix.

Для получения более подробных сведений касательно JSON формата по обнаружению элементов данных и примера каким образом реализовать своё собственное обнаружение файловых систем при помощи Perl скрипта, смотрите создание пользовательских LLD правил.

Ограничения данных для возвращаемых значений

Ограничения для JSON данных низкоуровневого правила обнаружения отсутствуют, если эти данные получены напрямую Zabbix сервером, так как полученные значения обрабатываются без сохранения в базу данных. Также ограничения отсутствуют и для пользовательских правил низкоуровневого обнаружения, однако, если предполагается получение пользовательских LLD данных при помощи пользовательского параметра, тогда накладывается ограничение по размеру значения (512 КБ) на сам пользовательский параметр.

Если данные поступают от Zabbix прокси, этот прокси вынужден сначала записать их в базу данных. В таком случае накладываются ограничения к базе данных, например, 2048 байт для Zabbix прокси, который работает с IBM DB2 базой данных.

Несколько LLD правил по одному и тому же элементу данных

Начиная с Zabbix агента версии 3.2, имеется возможность задать несколько правил низкоуровневого обнаружения по одному и тому же элементу данных обнаружения.

Чтобы это сделать, вам необходио указать параметр агента Alias, разрешив использование измененных ключей элемента данных обнаружения в разных правилах обнаружения, например `vfs.fs.discovery[foo]`, `vfs.fs.discovery[bar]` и так далее.

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 (нппример, 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.

Обратите внимание на то, что при использовании пользовательского параметра, возвращаемые данные ограничены 512 КБ. Для получения более подробных сведений смотрите ограничения данных для возвращаемых значений LLD.

2 Использование макросов LLD в контекстах пользовательских макросов

Пользовательские макросы с контекстом можно использовать для получения более гибких порогов в выражениях триггеров. Разные пороги можно задать на уровне пользовательского макроса и затем их можно использовать в константах триггеров, в зависимости от обнаруженного контекста. Обнаруженный контекст появляется, когда используемые макросы низкоуровневого обнаружения в макросах раскрываются в реальные значения.

Для иллюстрации мы можем использовать данные из приведенного примера выше, предположим, что будут обнаружены следующие файловые системы: `/`, `/home`, `/tmp`, `/usr`, `/var`.

Мы можем задать узлу сети прототип триггера на свободное место на диске, где порог выражается при помощи пользовательского макроса с контекстом:

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

Затем добавим пользовательские 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**%.