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.

В Zabbix поддерживаются три встроенных типа элементов данных для обнаружения:

- обнаружение файловых систем;
- обнаружение сетевых интерфейсов;
- обнаружение CPU и ядер CPU;
- обнаружение SNMP OID'ов;
- обнаружение с использованием SQL запросов ODBC;
- обнаружение Windows служб.

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”.

Элементы данных низкоуровневого обнаружения - vfs.fs.discovery, net.if.discovery поддерживаются Zabbix агентом начиная с версии 2.0.
Элемент данных низкоуровневого обнаружения “system.cpu.discovery” поддерживается Zabbix агентом начиная с версии 2.4.
Обнаружение SNMP OID'ов поддерживается Zabbix сервером и прокси начиная с версии 2.0.
Обнаружение с использованием SQL запросов ODBC поддерживается Zabbix сервером и прокси начиная с версии 3.0.

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”.

Следующий раздел иллюстрирует весь процесс, описанный выше, в деталях и служит руководством, как осуществлять все упомянутые выше типы обнаружений. Последний раздел описывает формат JSON элементов данных обнаружения и дает пример того, как реализовать ваш собственный скрипт для обнаружения файловых систем, используя Perl скрипт.

Ограничения данных для возвращаемых значений

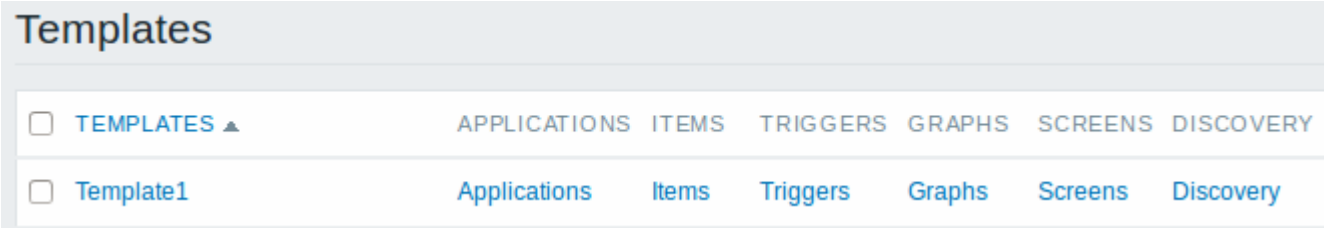
Ограничения для JSON данных низкоуровневого правила обнаружения отсутствуют, если эти данные получены напрямую Zabbix сервером, так как полученные значения обрабатываются без сохранения в базу данных. Также ограничения отсутствуют и для пользовательских правил низкоуровневого обнаружения, однако, если 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.

3.1 Обнаружение файловых систем

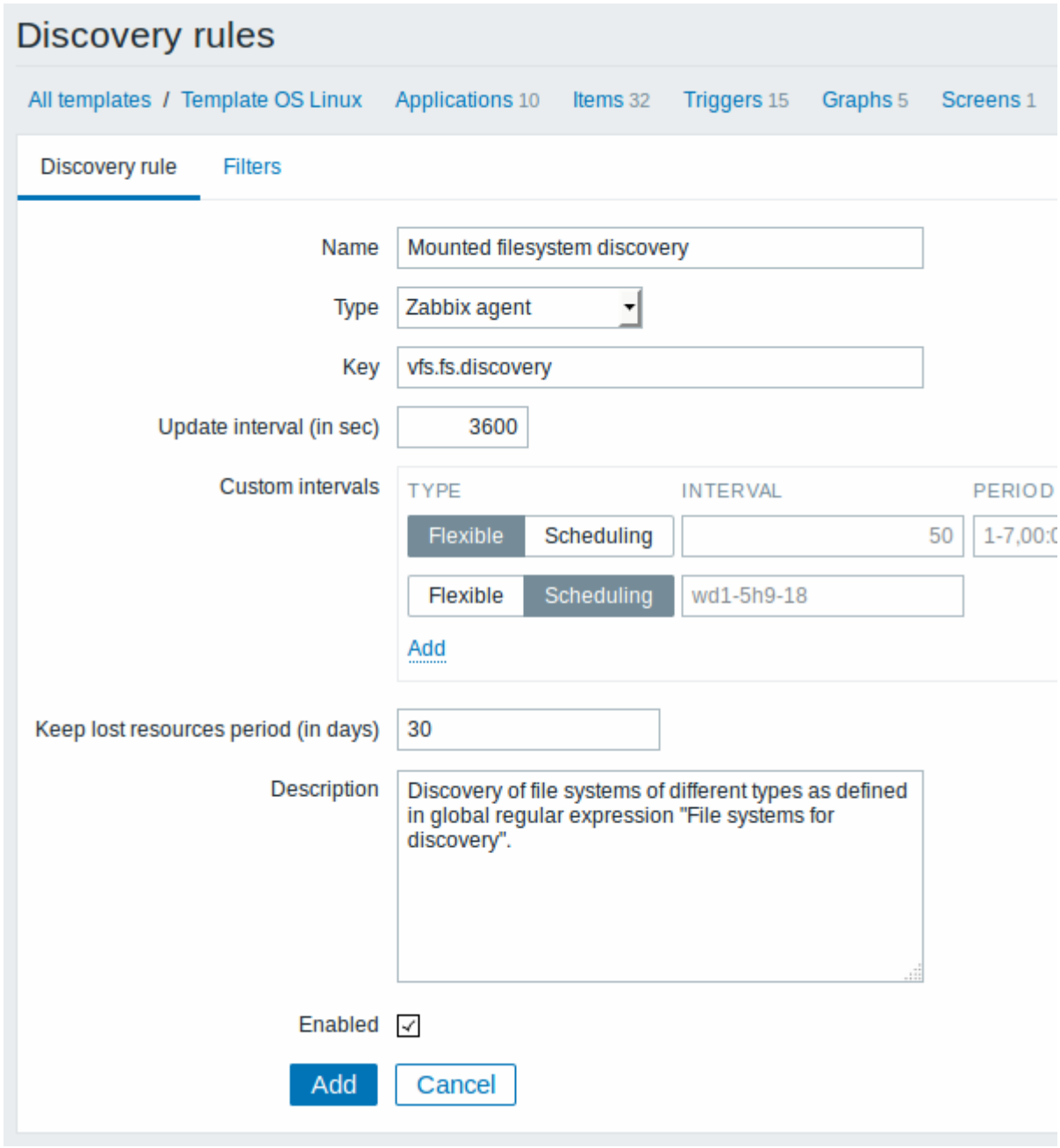
Для настройки обнаружения файловых систем, сделайте следующее:

- Перейдите в: *Настройки* → *Шаблоны*
- Нажмите на *Обнаружение* в строке соответствующего шаблона



- Нажмите на *Создать правило обнаружения* в верхнем правом углу экрана
- Заполните диалог следующими деталями

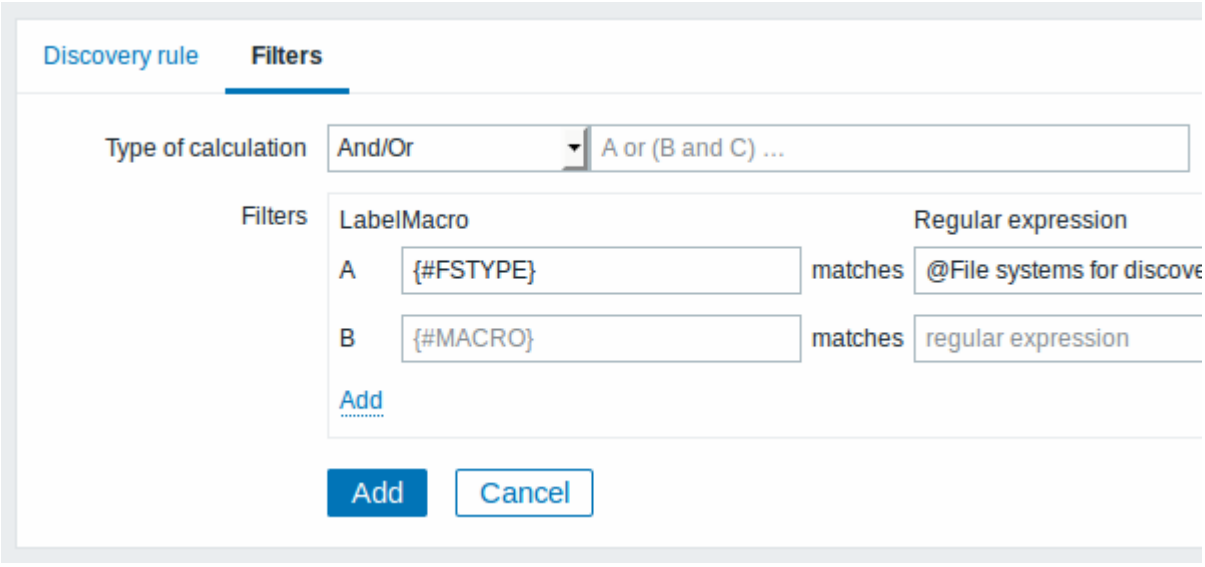
Вкладка **Правило обнаружения** содержит общие атрибуты правила обнаружения:



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 на 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.

Параметр	Описание
Интервал обновления (в сек)	Этот фильтр 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. *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.
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 (дней)	Это поле позволяет вам указать как много дней обнаруженный объект будет храниться (не будет удален), как только его состояние обнаружения станет “Не обнаруживается более” (макс 3650 дней). *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.

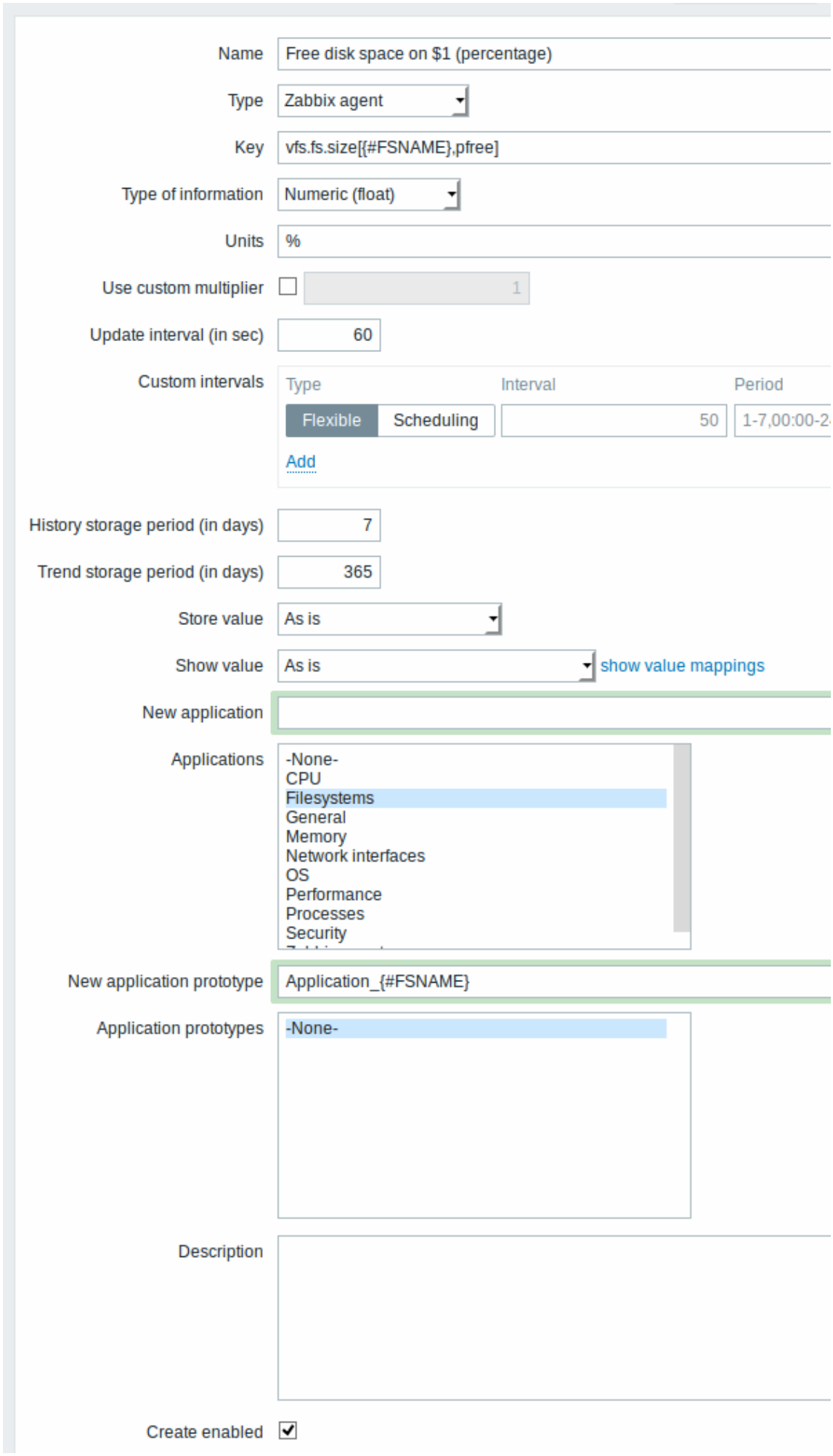
Вкладка **Фильтры** содержит определения фильтрации правила обнаружения:



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 Расширенные регулярные выражения POSIX. 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.

База данных Zabbix в MySQL должна быть создана чувствительной к регистру, если имена файловых систем различаются только по регистру, чтобы обнаружение сработало корректно.
Ошибка или опечатка в регулярном выражении, которое используется в LLD правиле элементов конфигурации, значений истории и событий по большому количеству 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.

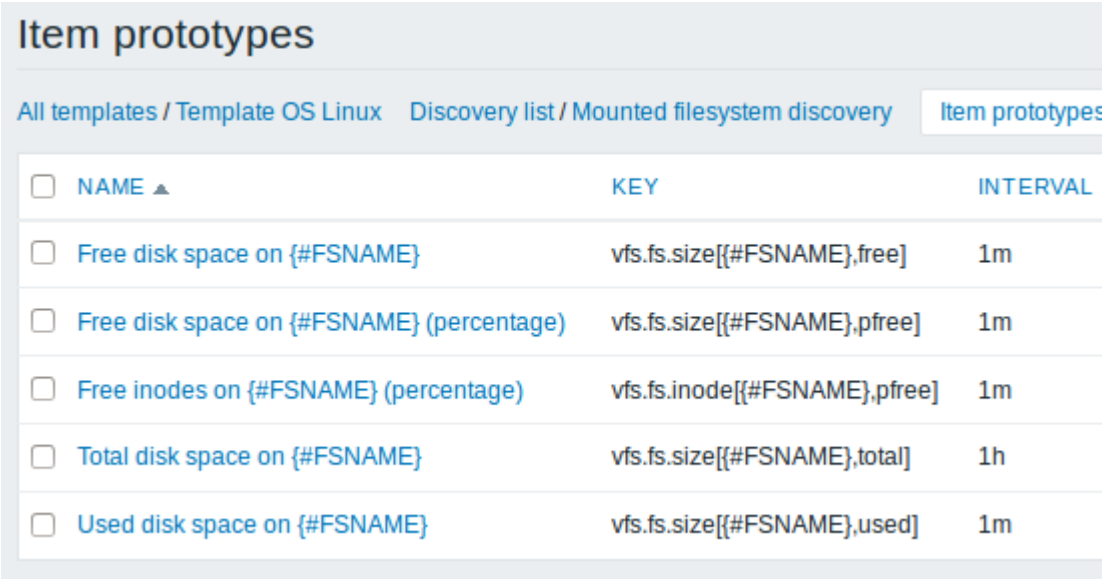
Как только правило будет создано, перейдем к элементам данных этого правила и нажмем “Создать прототип”, чтобы создать прототип элементов данных. Обратите внимание на то, как используется макрос {#FSNAME}, где требуется указать имя файловой системы. Когда правило будет обрабатываться, этот макрос будет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


Name
Free disk space on $1 (percentage)
Type
Zabbix agent
Key
vfs.fs.size[{#FSNAME},pfree]
Type of information
Numeric (float)
Units
%
Use custom multiplier
1
Update interval (in sec)
60
Custom intervals
Type
Interval
Period
Flexible
Scheduling
50
1-7,00:00-2
Add
History storage period (in days)
7
Trend storage period (in days)
365
Store value
As is
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

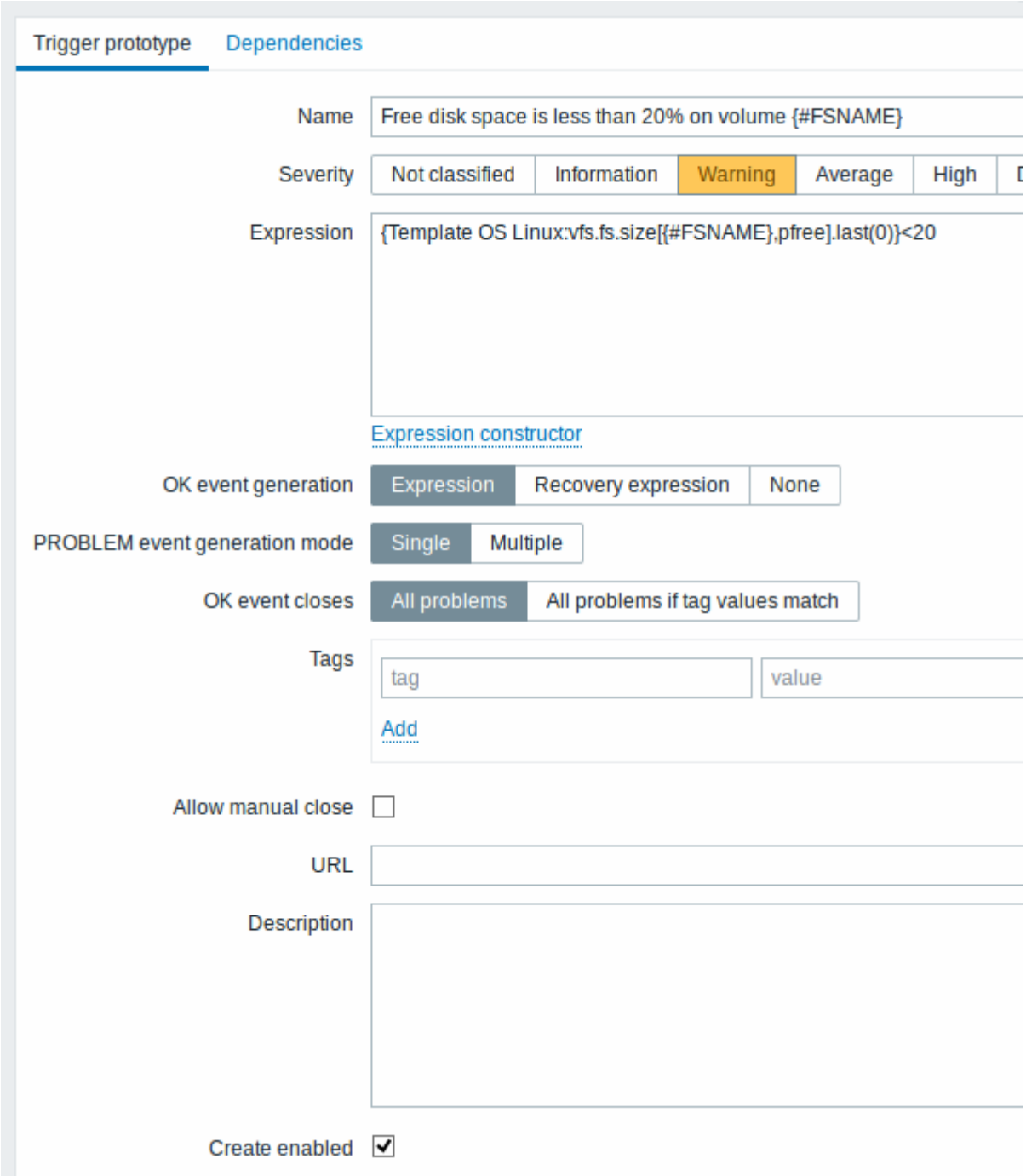
Атрибуты, которые специфичны для прототипов элементов данных:

Параметр	**Описание**
Новый прототип группы элементов данных	Вы можете добавить новый прототип группы элементов данных. В прототипах групп 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:



Теперь похожим способом мы создадим прототипы триггеров:



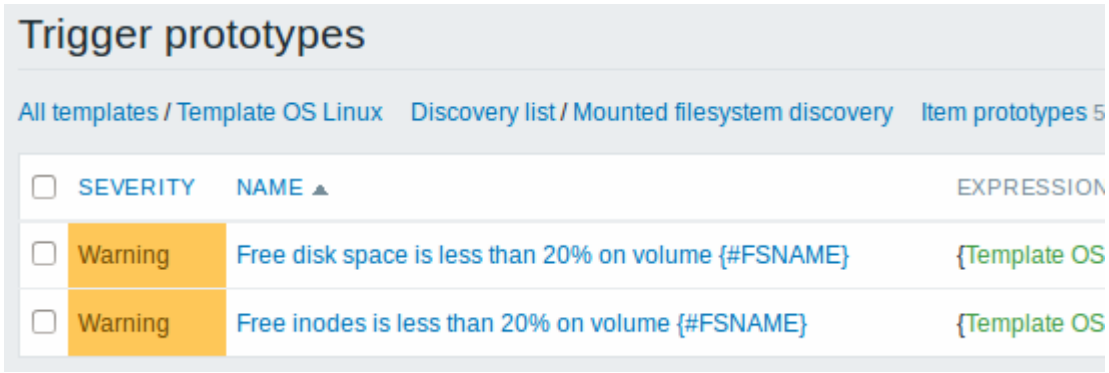
Атрибуты, которые специфичны для прототипов триггеров:

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.

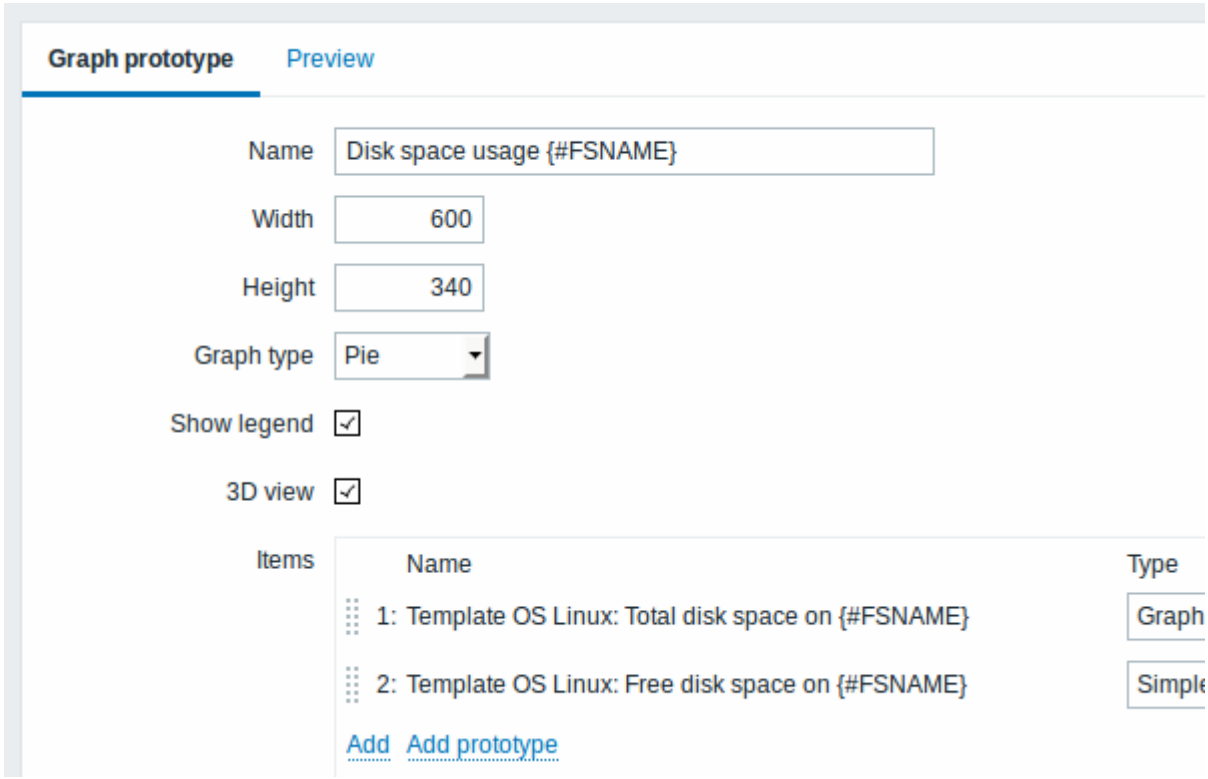
Когда реальные триггера создаются из прототипов, возможно потребуется некая гибкость в отношении того какая константа ('20' в нашем примере) в выражении используется для сравнения. Смотрите как пользовательские макросы с контекстом могут быть полезны для получения подобной гибкости.

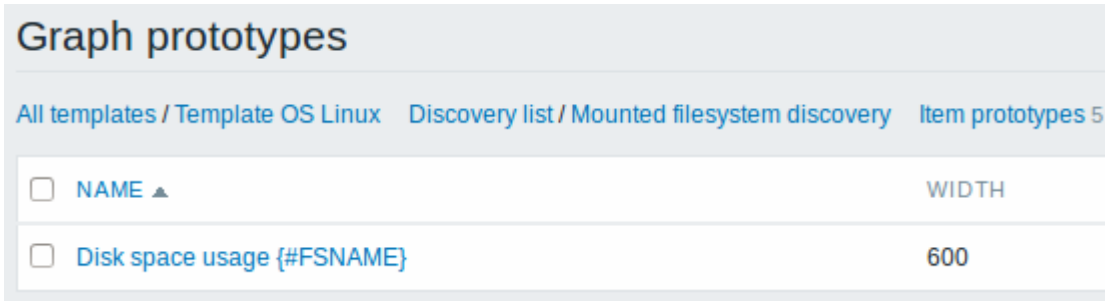
Также вы можете задать зависимости между прототипами триггеров (поддерживается начиная с Zabbix 3.0). Чтобы это сделать, перейдите на вкладку *Зависимости*. Прототип триггеров может зависеть от другого прототипа триггеров из этого же правила

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.

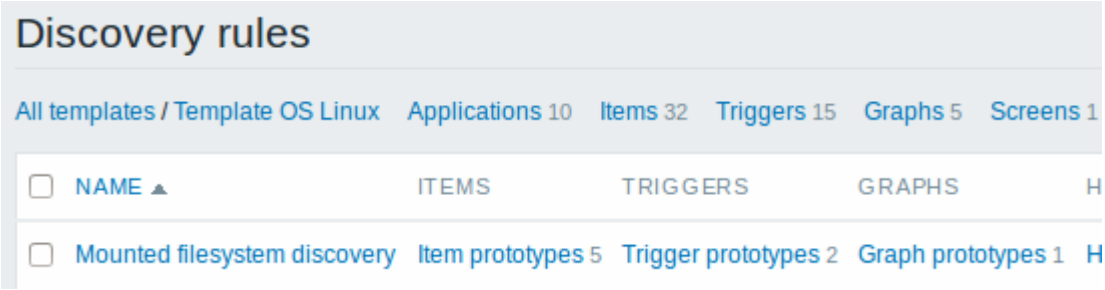


И также прототипы графиков:



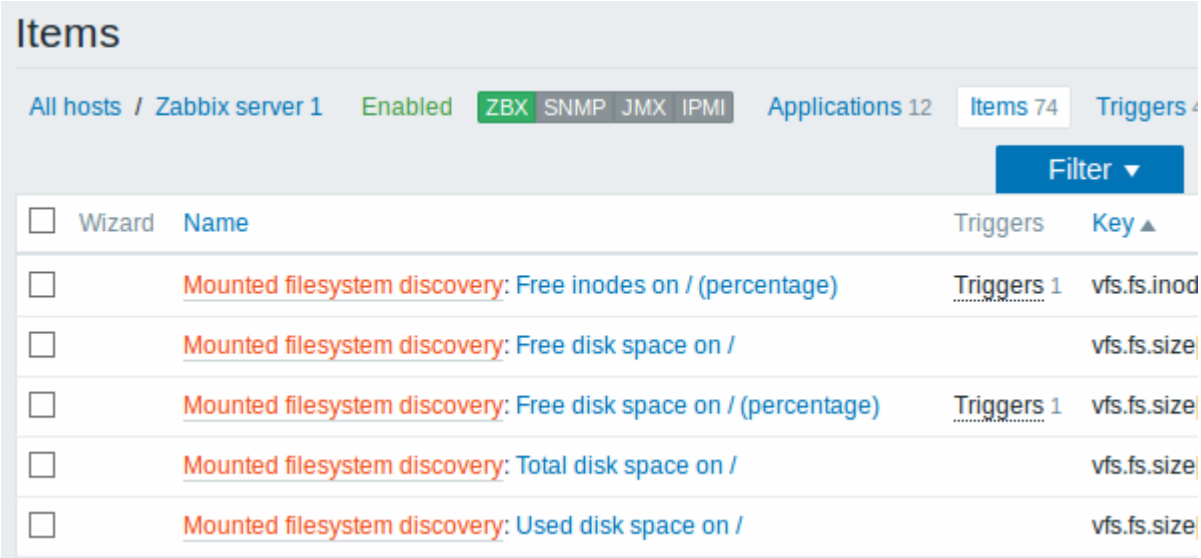


В конце концов, мы создали правило обнаружения, которое выглядит как видно ниже. Оно имеет пять прототипов элементов данных, два прототипа триггеров и один прототип графика.



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

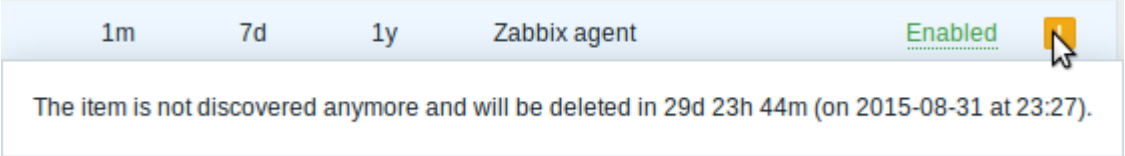
Представленные снимки экрана ниже иллюстрируют как выглядят уже обнаруженные элементы данных, триггера и графики в настройке узла сети. Обнаруженные объекты имеют префикс ссылку золотистого цвета, которая ведет к правилу обнаружения, создавшего эти объекты.



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) созданные с помощью низкоуровневого правила обнаружения 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*.

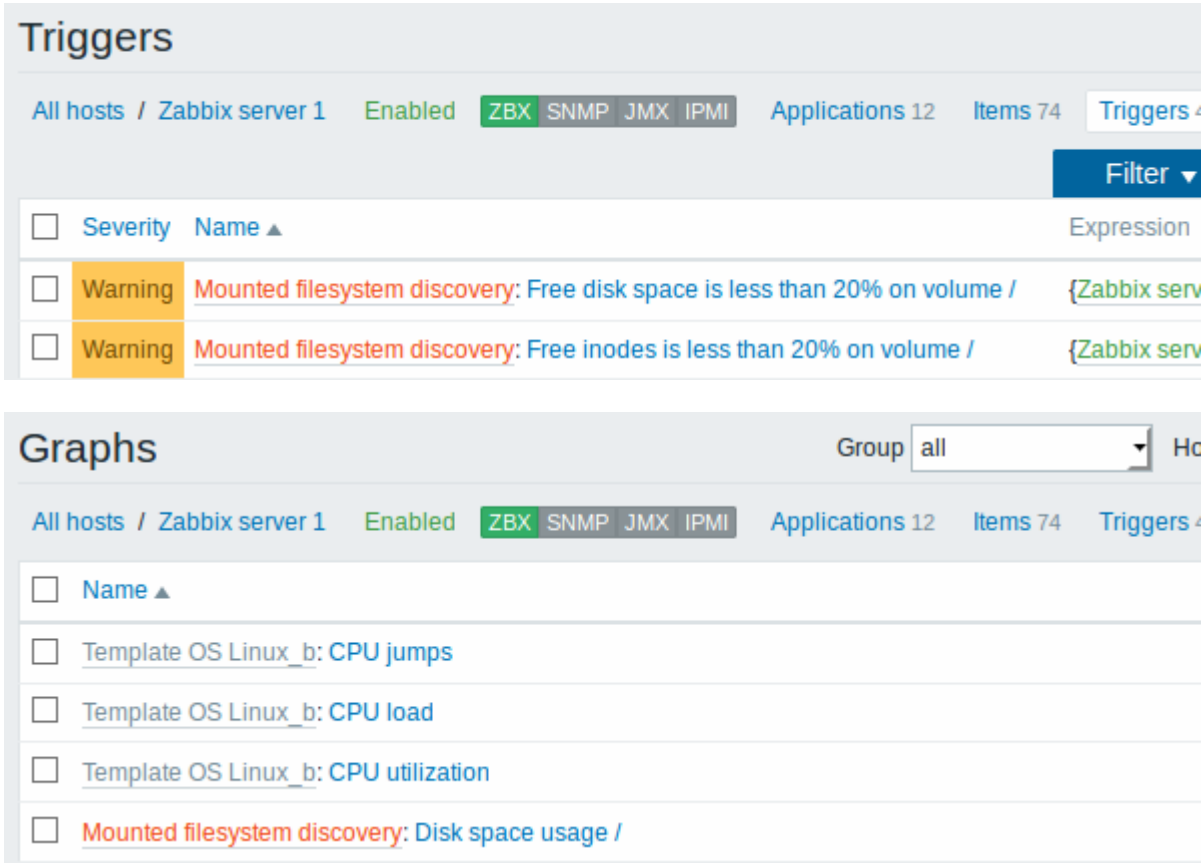
Когда обнаруженный объект становится 'Более не обнаруживается', в списке элементов данных будет отображаться оранжевый индикатор времени жизни. Переместите курсор мыши на этот индикатор и вы увидите сообщение с количеством дней до момента удаления элемента данных.



Если объекты отмечены 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 содержащие другие объекты, которые отмечены к удалению, не будут обновляться при изменениях на уровне правила обнаружения. Например, триггеры обнаруженные при помощи низкоуровневого обнаружения не будут обновляться, если они содержат элементы данных, которые помечены к удалению.



3.2 Обнаружение сетевых интерфейсов

Обнаружение сетевых интерфейсов осуществляется таким же образом, как и обнаружение файловых систем, за исключением того, что мы используем ключ правила обнаружения “net.if.discovery” вместо “vfs.fs.discovery” и макрос {#IFNAME} вместо {#FSNAME} в фильтре и в прототипах элементов данных/триггеров/графиков.

Примеры прототипов элементов данных, которые вы можете захотеть создать основываются на “net.if.discovery”: “net.if.in[{#IFNAME},bytes]”, “net.if.out[{#IFNAME},bytes]”.

Смотрите выше для получения информации по поводу фильтра.

3.3 Обнаружение CPU и ядер CPU

Обнаружение CPU и ядер CPU выполняется аналогично обнаружению сетевых интерфейсов за исключением того, что ключем правила обнаружения является “system.cpu.discovery”. Этот ключ обнаружения возвращаетс два макроса - {#CPU.NUMBER} и {#CPU.STATUS},

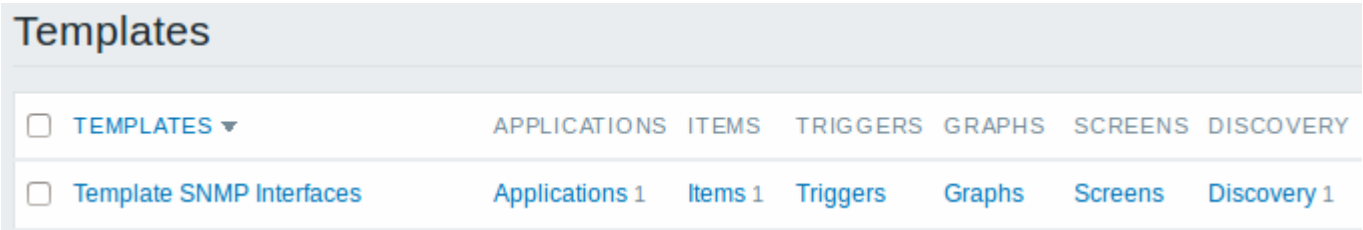
идентифицирующие порядковый номер CPU и состояние соответственно. Отметим, нельзя сделать четкого различия между действительными, физическими процессорами, ядрами и hyperthread. {#CPU.STATUS} на Linux, UNIX и BSD системах возвращают состояние процессора, которое может быть как “online”, так и “offline”. На Windows системах, этот же макрос может представлять собой третье значение - “unknown” - которое указывает на то, что процессор был обнаружен, но информация по нему еще не собрана.

Обнаружение CPU основано на процессе коллектора агента, чтобы поддерживать соответствие с данными, которые поставляются коллектором и сохранить ресурсы на получение данных. Такое поведение дает эффект, что этот ключ элемента данных не работает с флагом командой строки тестирования (-t) бинарного файла, который возвращает состояние NOT_SUPPORTED и сопутствующее сообщение о том, что процесс коллектора не запущен.

Прототипы элементов данных, которые можно создать на основе обнаружения CPU включают в себя, например, “system.cpu.util[{#CPU.NUMBER}, <тип>, <режим>]” и “system.hw.cpu[{#CPU.NUMBER}, <информация>]”.

3.4 Обнаружение SNMP OID'ов

В этом примере мы осуществим обнаружение SNMP на коммутаторе. Сначала перейдем в “Настройка” → “Шаблоны”.



Для изменения правил обнаружения шаблона, нажмите на ссылку в колонке “Обнаружение”.

Затем нажмите “Создать правило” и заполните форму, как отображено на снимке экрана ниже.

В отличие от обнаружения файловых систем и сетевых интерфейсов - этот элемент данных не требует наличия ключа “snmp.discovery”, достаточно указать, что типом элемента данных является SNMP агент.

OID'ы для обнаружения добавляются в поле SNMP OID в следующем формате:
`discovery[{#MAKPOC1}, oid1, {#MAKPOC2}, oid2, …,]`

где *{#MAKPOC1}*, *{#MAKPOC2}* … допустимые имена низкоуровневых макросов и *oid1*, *oid2*… являются OID'ами способными сгенерировать осмысленные значения для этих макросов. Встроенный макрос *{#SNMPINDEX}* содержит индекс обнаруженного OID, который применяется к обнаруженным объектам. Обнаруженные объекты группируются по значению макроса *{#SNMPINDEX}*.

Для понимания того, что мы имеем в виду, давайте выполним несколько раз snmpwalk на нашем коммутаторе:

```
$ 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

$ snmpwalk -v 2c -c public 192.168.1.1 IF-MIB::ifPhysAddress
IF-MIB::ifPhysAddress.1 = STRING: 8:0:27:90:7a:75
IF-MIB::ifPhysAddress.2 = STRING: 8:0:27:90:7a:76
IF-MIB::ifPhysAddress.3 = STRING: 8:0:27:2b:af:9e
```

И зададим SNMP OID равным: `discovery[{#IFDESCR}, ifDescr, {#IFPHYSADDRESS}, ifPhysAddress]`

Теперь это правило будет обнаруживать объекты с макросом {#IFDESCR} равным **WAN**, **LAN1** и **LAN2**, макросом {#IFPHYSADDRESS} равным **8:0:27:90:7a:75**, **8:0:27:90:7a:76**, и **8:0:27:2b:af:9e**, макросом {#SNMPINDEX} равным индексам обнаруженных OID **1**, **2** и **3**:

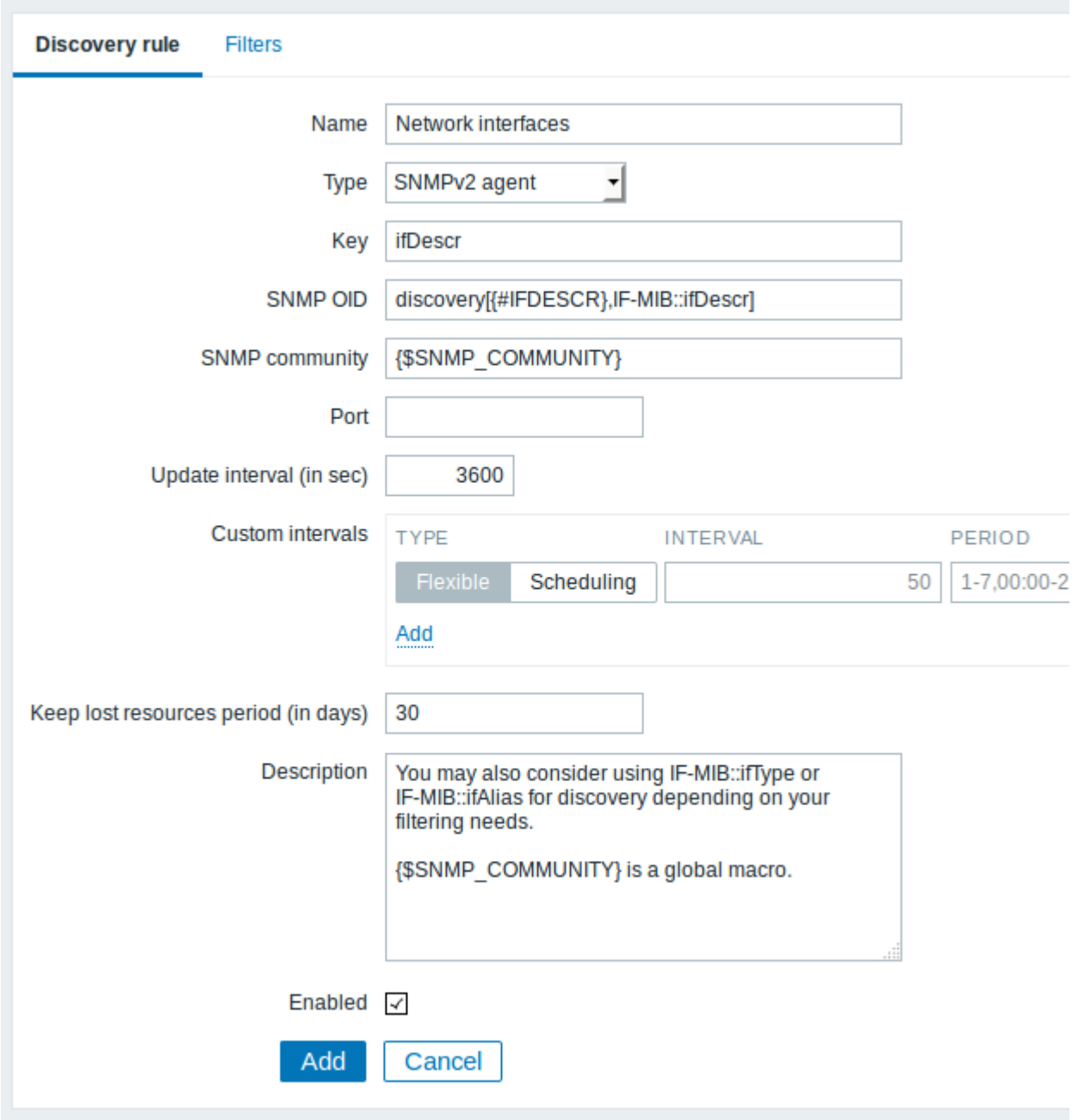
```
{
    "data": [
        {
            "{#SNMPINDEX}": "1",
            "{#IFDESCR}": "WAN",
            "{#IFPHYSADDRESS}": "8:0:27:90:7a:75"
        },
        {
            "{#SNMPINDEX}": "2",
            "{#IFDESCR}": "LAN1",
            "{#IFPHYSADDRESS}": "8:0:27:90:7a:76"
        },
        {
            "{#SNMPINDEX}": "3",
            "{#IFDESCR}": "LAN2",
            "{#IFPHYSADDRESS}": "8:0:27:2b:af:9e"
        }
    ]
}
```

Если обнаруженный объект не имеет указанный OID, тогда по этому объекту соответстующий макрос пропускается. Например, если у нас есть следующие данные:

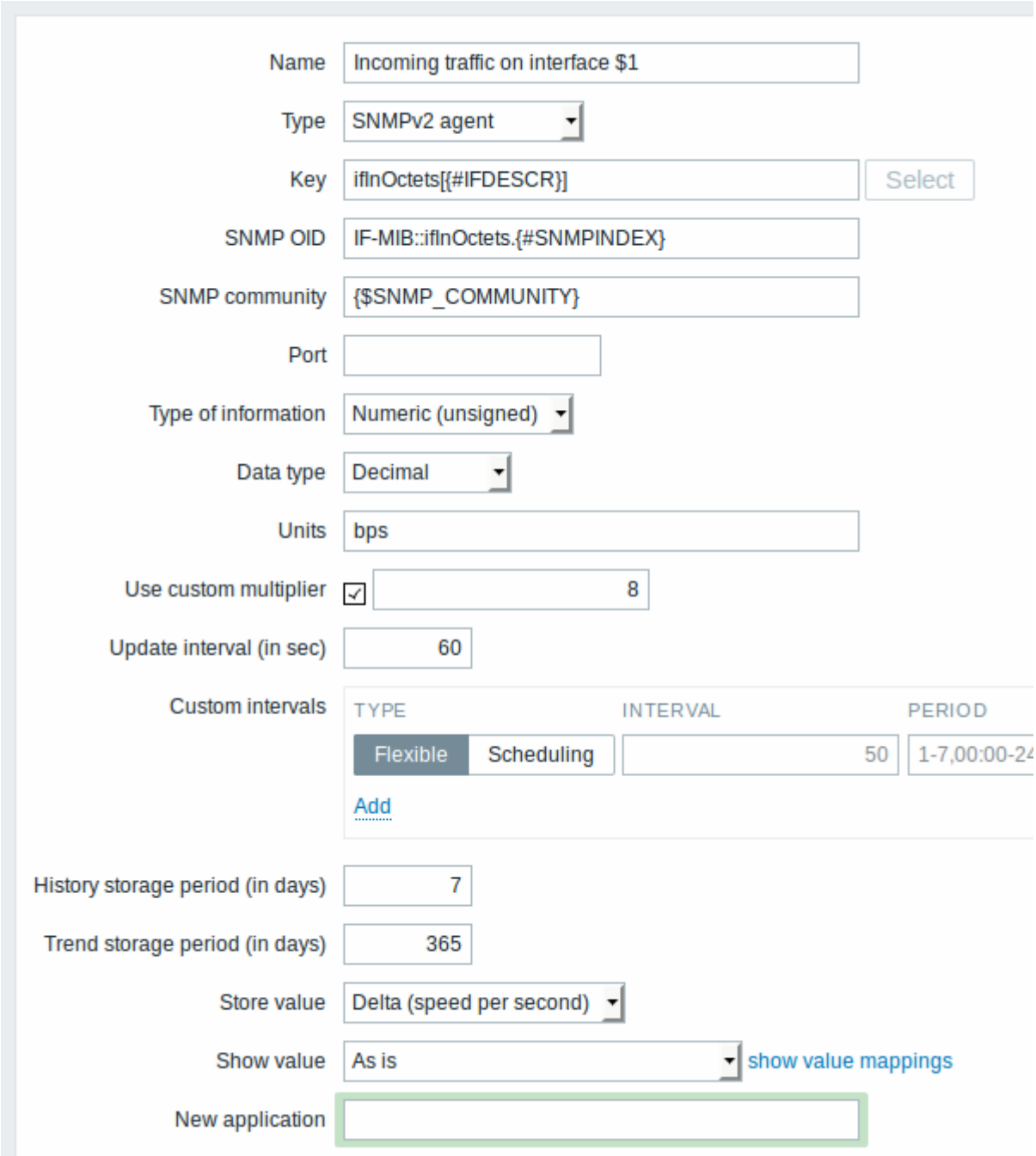
```
ifDescr.1 "Interface #1"
ifDescr.2 "Interface #2"
ifDescr.4 "Interface #4"

ifAlias.1 "eth0"
ifAlias.2 "eth1"
ifAlias.3 "eth2"
ifAlias.5 "eth4"
```

Тогда, в случае SNMP обнаружения `discovery[{#IFDESCR}, ifDescr, {#IFALIAS}, ifAlias]` вернется следующая структура:

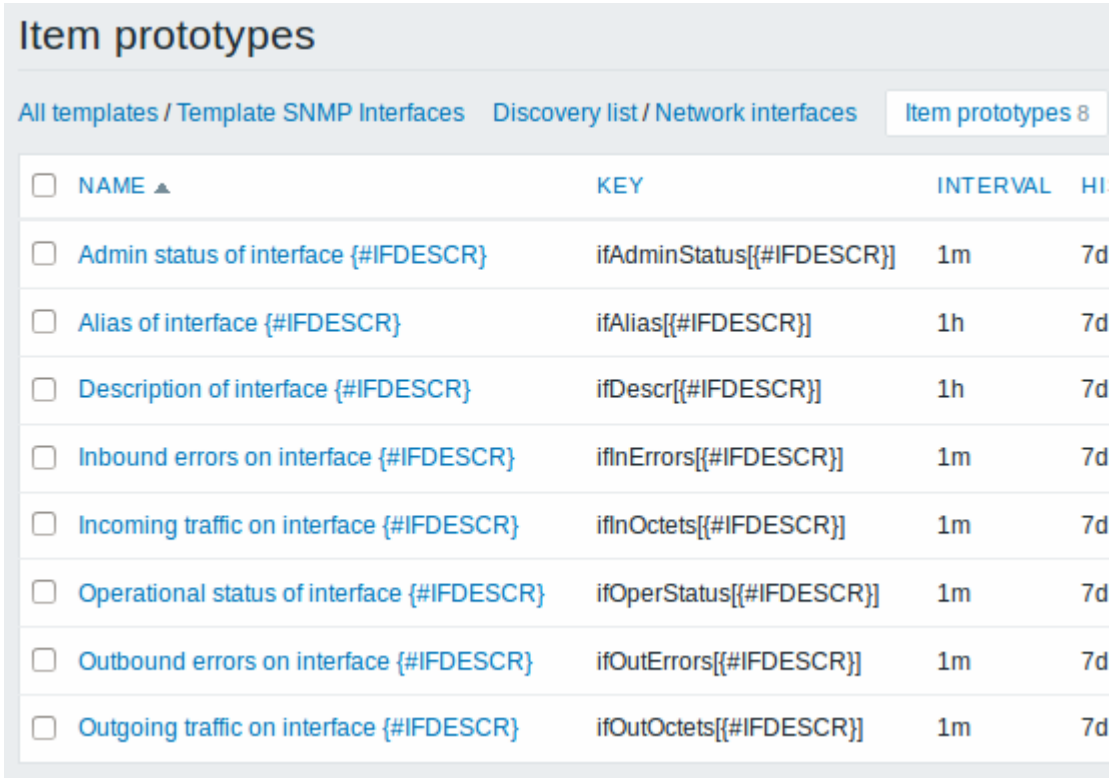
```
{
    "data": [
        {
            "{#SNMPINDEX}": 1,
            "{#IFDESCR}": "Interface #1",
            "{#IFALIAS}": "eth0"
        },
        {
            "{#SNMPINDEX}": 2,
            "{#IFDESCR}": "Interface #2",
            "{#IFALIAS}": "eth1"
        },
        {
            "{#SNMPINDEX}": 3,
            "{#IFALIAS}": "eth2"
        },
        {
            "{#SNMPINDEX}": 4,
            "{#IFDESCR}": "Interface #4"
        },
        {
            "{#SNMPINDEX}": 5,
            "{#IFALIAS}": "eth4"
        }
    ]
}
```



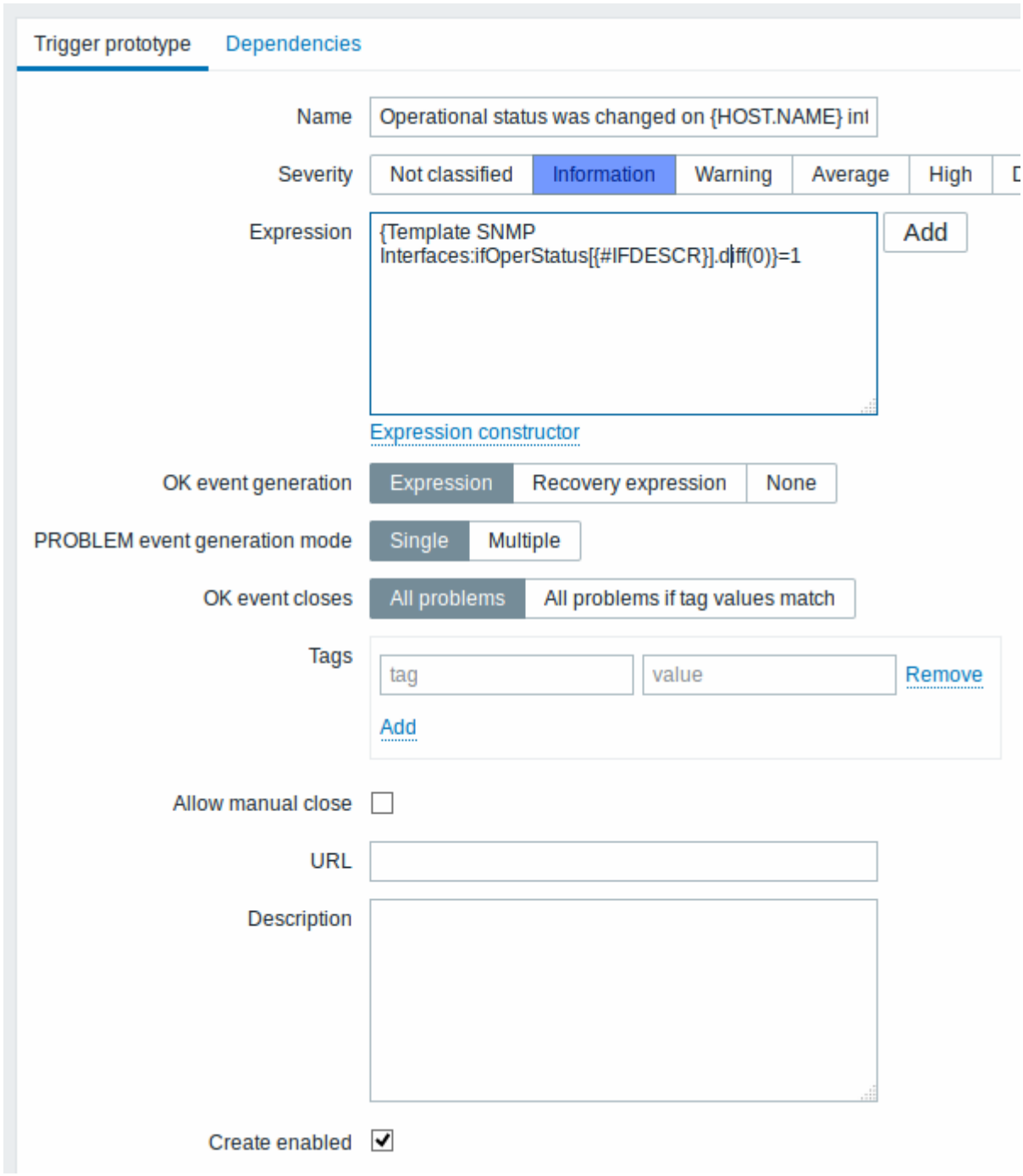
Представленный снимок экрана иллюстрирует как мы можем использовать эти макросы в прототипах элементов данных:

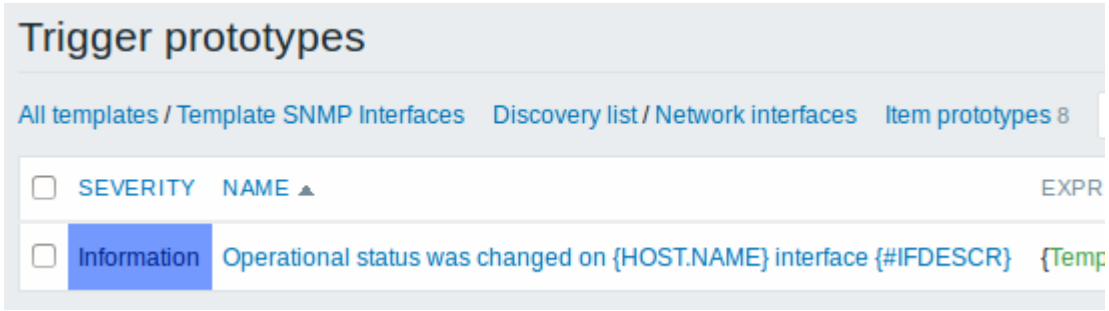


Опять же, создадим столько прототипов элементов данных, сколько необходимо:

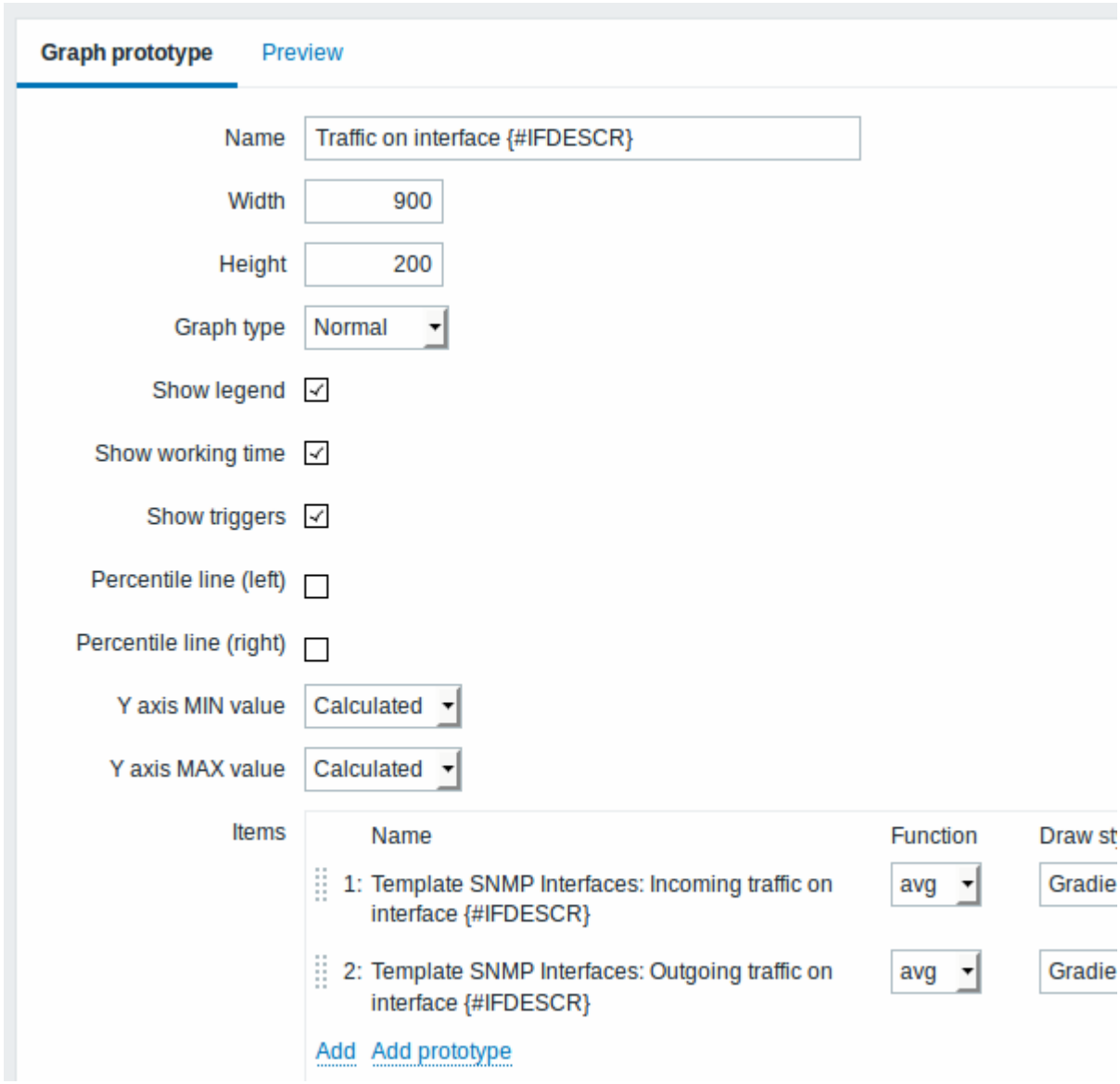


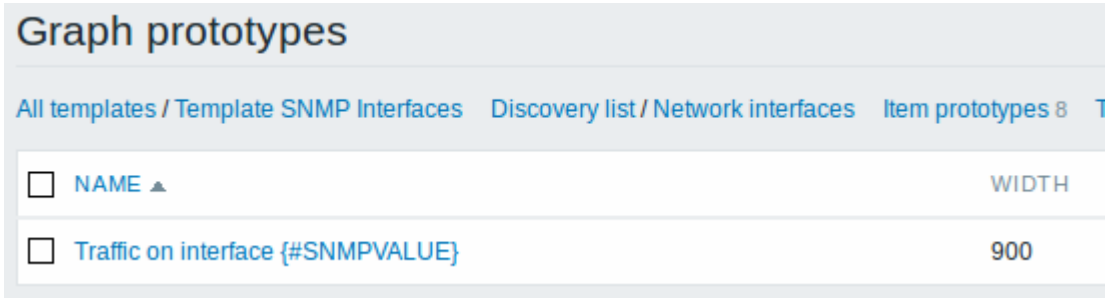
Также как и прототипы триггеров:



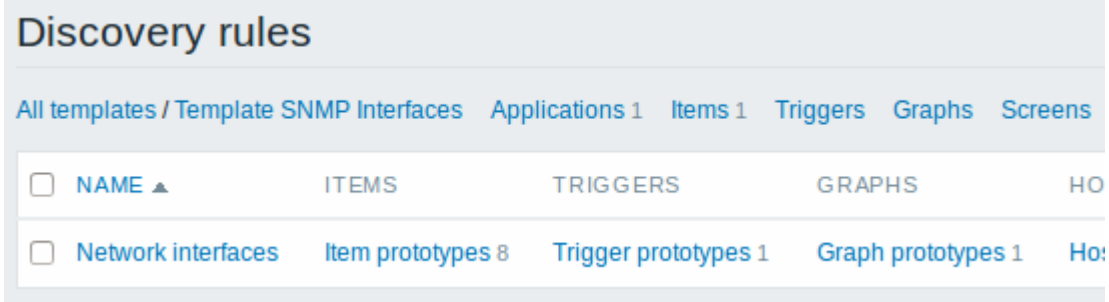


И прототипы графиков:

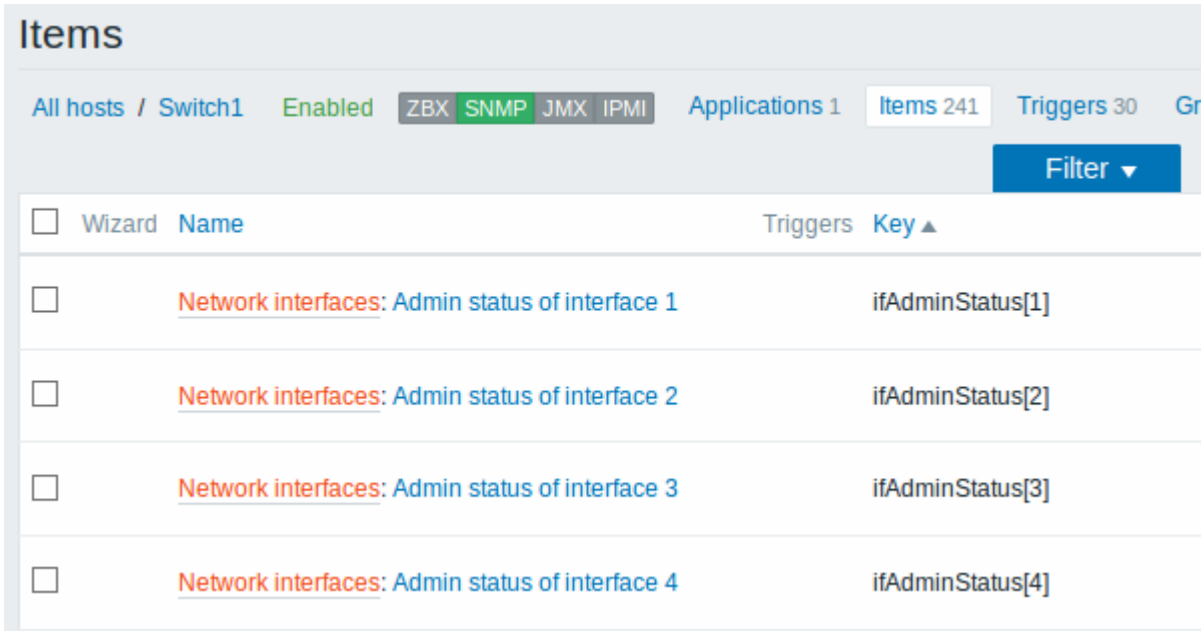


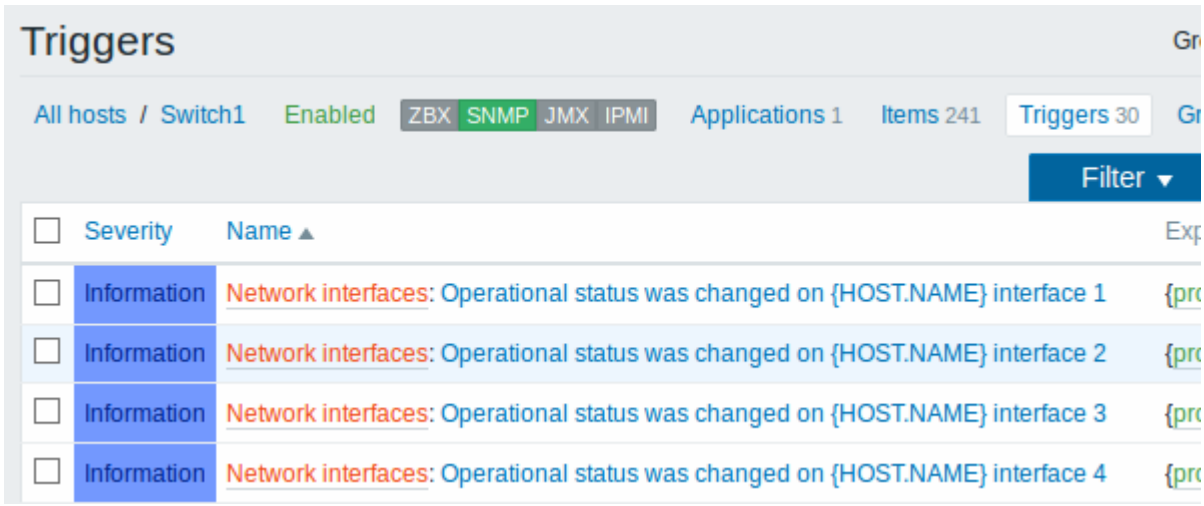


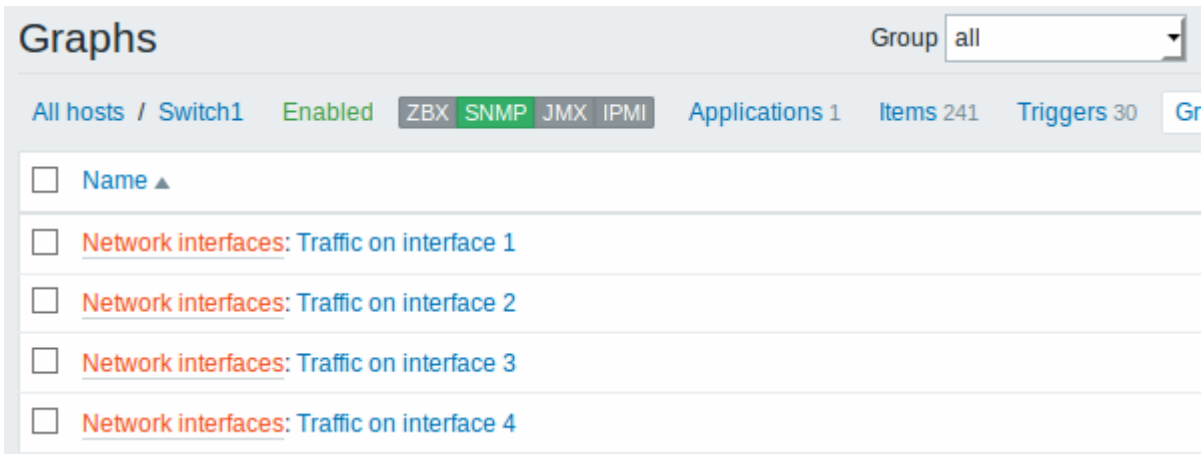
Результат нашего правила обнаружения:



Когда сервер работает, он создаст реальные элементы данных, триггеры и графики на основе значений, полученных от SNMP правила обнаружения. В настройках узлов сети они будут иметь префикс ссылку оранжевого цвета, которая ведет к правилу обнаружения, их создавшего.





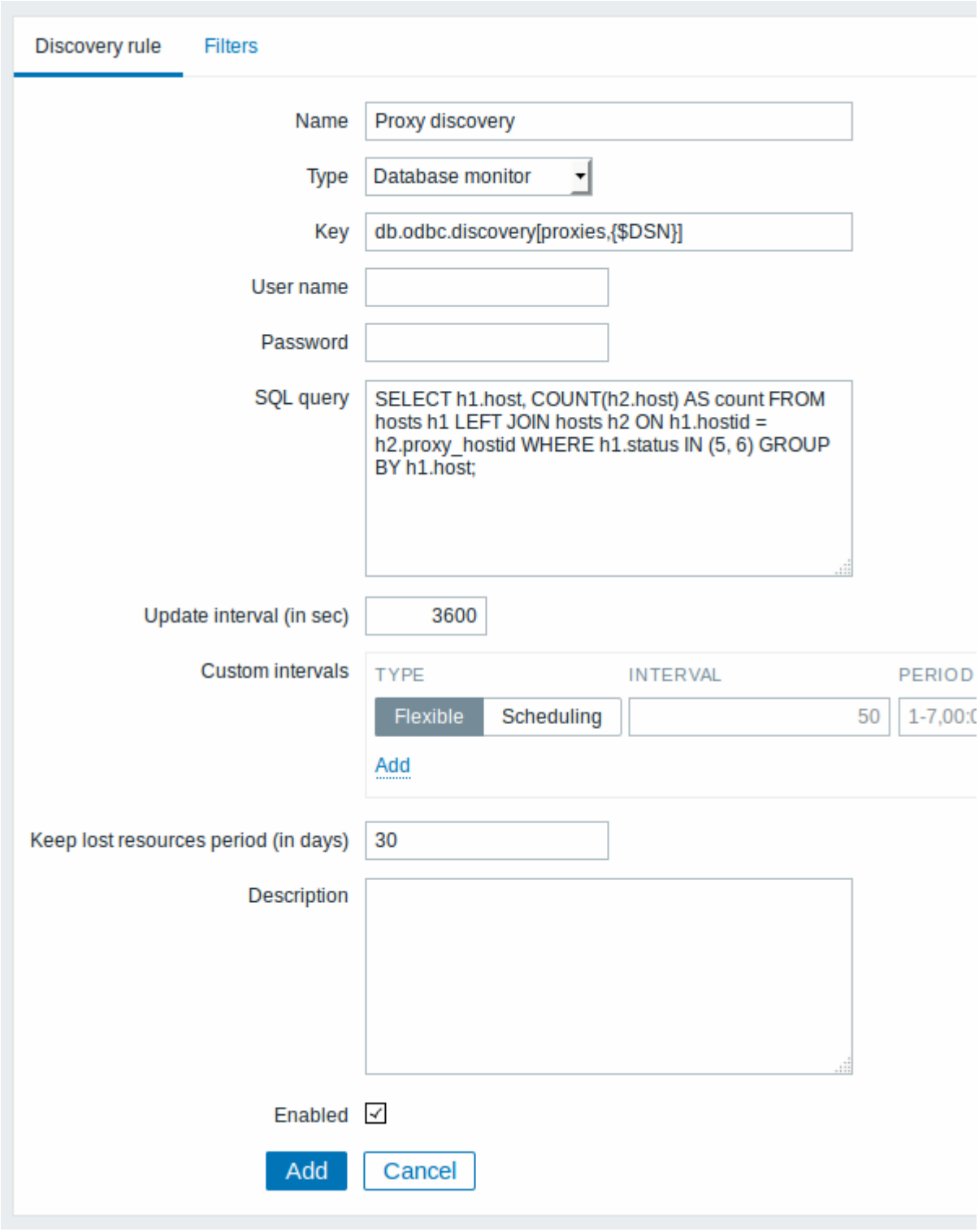


3.5 Обнаружение с использованием SQL запросов ODBC

Этот тип обнаружения осуществляется с использованием SQL запросов, полученные результаты которых автоматически преобразуются в объект JSON, пригодный для низкоуровневого обнаружения. SQL запросы выполняются при помощи элементов данных типа “Монитор баз данных”. Так что, большая часть указаний со страницы ODBC мониторинга применима к получению работающего “Монитора баз данных” правила обнаружения, единственная разница лишь в том, что необходимо использовать ключ “db.odbc.discovery[<описание>,<dsn>]” вместо “db.odbc.select[<описание>,<dsn>]”.

В качестве практического примера, иллюстрирующего как SQL запрос трансформируется в JSON, рассмотрим низкоуровневое обнаружения Zabbix прокси, выполнив ODBC запрос в Zabbix базу данных. Он может быть полезен для автоматического создания внутренних элементов данных “zabbix[proxy,<имя>,lastaccess]”, чтобы наблюдать какие прокси живы.

Давайте начнем с настройки правила обнаружения:



Здесь используется следующий прямой запрос в базу данных Zabbix для выборки всех Zabbix прокси вместе с количеством узлов сети, за которыми эти прокси наблюдают. Количество узлов сети можно использовать, например, для фильтрации пустых прокси:

```
mysql> SELECT h1.host, COUNT(h2.host) AS count FROM hosts h1 LEFT JOIN hosts
h2 ON h1.hostid = h2.proxy_hostid WHERE h1.status IN (5, 6) GROUP BY
h1.host;
+---------+-------+
| host    | count |
+---------+-------+
```

```
| Japan 1 |     5 |
| Japan 2 |    12 |
| Latvia  |     3 |
+---------+-------+
3 rows in set (0.01 sec)
```

Благодаря внутреннему механизму обработки элемента данных “db.odbc.discovery[]”, результат этого запроса автоматически преобразуется в следующий JSON:

```
{
    "data": [
        {
            "{#HOST}": "Japan 1",
            "{#COUNT}": "5"
        },
        {
            "{#HOST}": "Japan 2",
            "{#COUNT}": "12"
        },
        {
            "{#HOST}": "Latvia",
            "{#COUNT}": "3"
        }
    ]
}
```

Видно, что имена колонок становятся именами макросов и выбранные строки становятся значениями этих макросов.

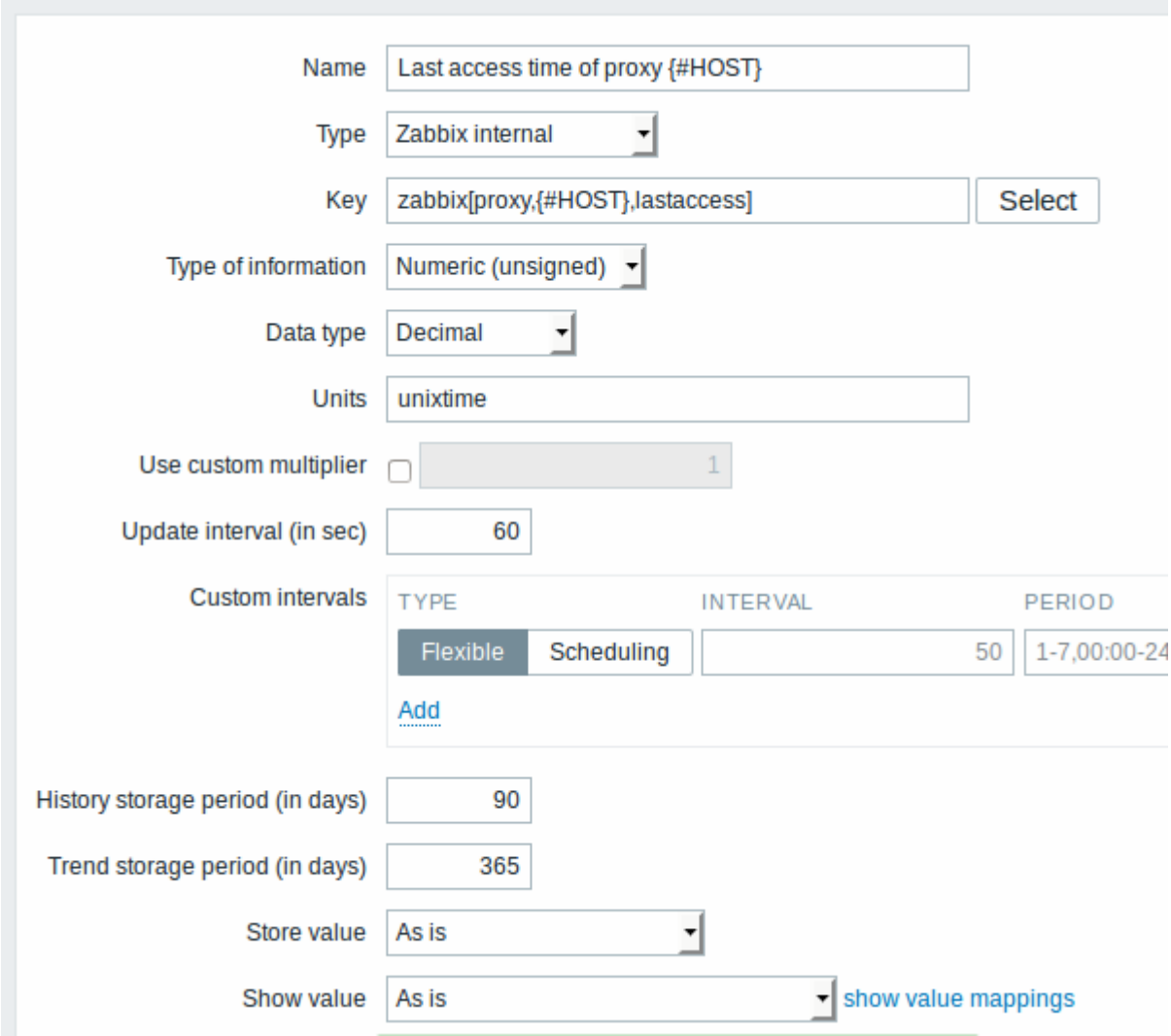
Если результат преобразования имени колонки в имя макроса неочевиден, предлагается использовать алиасы к именам колонок, так же как “COUNT(h2.host) AS count” в примере выше.

В случае, если имя колонки не удается сконвертировать в допустимое имя макроса, правило обнаружения становится неподдерживаемым с детальным сообщением об ошибке какой номер колонки не удалось преобразовать. Если желательна дополнительная помощь, полученные имена колонки отражаются при DebugLevel=4 в файле журнала Zabbix сервера:

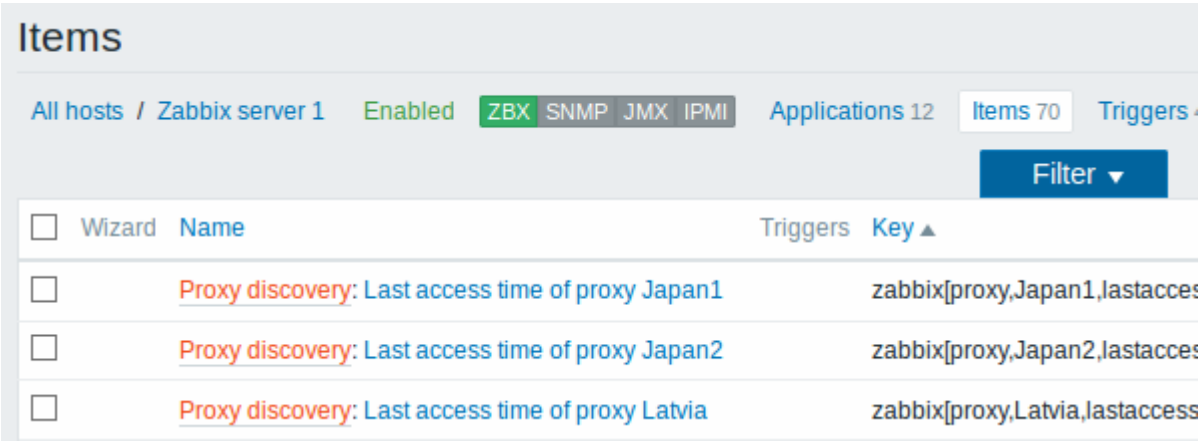
```
$ grep db.odbc.discovery /tmp/zabbix_server.log
 ...
 23876:20150114:153410.856 In db_odbc_discovery() query:'SELECT h1.host,
COUNT(h2.host) FROM hosts h1 LEFT JOIN hosts h2 ON h1.hostid =
h2.proxy_hostid WHERE h1.status IN (5, 6) GROUP BY h1.host;'
 23876:20150114:153410.860 db_odbc_discovery() column[1]:'host'
 23876:20150114:153410.860 db_odbc_discovery() column[2]:'COUNT(h2.host)'
 23876:20150114:153410.860 End of db_odbc_discovery():NOTSUPPORTED
 23876:20150114:153410.860 Item [Zabbix
server:db.odbc.discovery[proxies,{$DSN}]] error: Cannot convert column #2
```

```
name to macro.
```

Теперь, когда мы понимаем как SQL запрос трансформируется в JSON объект, мы можем использовать макрос {#HOST} в прототипах элементов данных:



Как только обнаружение будет выполнено, элемент данных будет создан по каждому прокси:



3.6 Обнаружение служб Windows

Обнаружение служб Windows осуществляется тем же самым способом, что и обнаружение файловых систем. Используемым ключем в правиле обнаружения является “service.discovery” и поддерживаются следующие макросы для применения их в фильтре и прототипах элементов данных/триггерах/графиков:

```
{#SERVICE.NAME}
{#SERVICE.DISPLAYNAME}
{#SERVICE.DESCRIPTION}
{#SERVICE.STATE}
{#SERVICE.STATENAME}
{#SERVICE.PATH}
{#SERVICE.USER}
{#SERVICE.STARTUP}
{#SERVICE.STARTUPNAME}
```

На основе обнаружения служб Windows вы можете создать прототип элементов данных вроде “service.info[{#SERVICE.NAME},<парам>]”, где *парам* принимает следующие значения: *state*, *displayname*, *path*, *user*, *startup* или *description*. Например, чтобы получить отображаемое имя службы вам необходимо использовать элемент данных “service.info[{#SERVICE.NAME},displayname]”. Если значение *парам* не указано (“service.info[{#SERVICE.NAME}]”), будет использоваться параметр *state* по умолчанию.

Макросы {#SERVICE.STATE} и {#SERVICE.STATENAME} возвращают одинаковое содержимое, однако, {#SERVICE.STATE} возвращает числовое значение (0-7), тогда как {#SERVICE.STATENAME} возвращает текст (*running*, *paused*, *start pending*, *pause pending*, *continue pending*, *stop pending*, *stopped* или *unknown*). То же самое относится и к {#SERVICE.STARTUP} и {#SERVICE.STARTUPNAME}, где первый возвращает числовое значение (0-4), тогда как второй - текст (*automatic*, *automatic delayed*, *manual*, *disabled*, *unknown*).

3.7 Настройка нескольких LLD правил по одному элементу данных

Начиная с Zabbix агента версии 3.2 имеется возможность изменения ключей элементов данных низкоуровневого обнаружения при помощи “Alias” параметра в zabbix_agentd.conf файле, чтобы была возможность настройки нескольких LLD правил по одному элементу данных.

3.8 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.

3.9 Использование макросов LLD в контекстах пользовательских макросов

Пользовательские макросы с контекстом можно использовать для получения более гибких порогов в выражениях триггеров. Можно задать разные пороги на уровне пользовательского макроса и затем использовать в константах триггеров в зависимости от обнаруженного контекста. Обнаруженный контекст появляется тогда, когда макрос низкоуровневого обнаружения, используемый в макросах, раскрывается в реальные значения.

Для иллюстрации мы модем взять данные из примера выше и предположить, что будут обнаружены следующие файловые системы: `/`, `/home`, `/tmp`, `/usr`, `/var`.

Мы можем добавить прототип триггера, который проверяет свободное место на диске, на узел сети, где порог задается 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**

Теперь, когда файловые системы будут обнаружены, события будут сгенерированы, если файловые системы `/`, `/usr` и `/var` имеют менее **10**% свободного места на диске, файловая система `/home` - менее **20**% свободного места на диске или файловая система `/tmp` - менее **50**% свободного места на диске.

LLD макросы не поддерживаются внутри контекстов пользовательских макросов в параметрах функций триггеров.