3 Descoberta de baixo nível (LLD)

Visão geral

O processo de LLD fornece uma forma automática de criar itens, triggers, gráficos para os diferentes objetos descobertos no dispositivo monitorado. Por exemplo, o Zabbix pode começar automaticamente a monitorar os sistemas de arquivo ou interfaces de rede de um servidor, sem precisar que sejam criados manualmente itens para cada um deles. Adicionalmente, o Zabbix também pode remove-los automaticamente também após determinado período consecutivo em que eles não sejam mais encontrados.

Nativamente no Zabbix, são suportados seis tipos de LLD:

- descoberta de sistemas de arquivo;
- descoberta de interfaces de rede;
- descoberta de CPUs seus núcleos;
- descoberta de árvores de OID SNMP;acesse
- descoberta usando consultas SQL/ODBC;
- descoberta de serviços Windows.

O usuário pode estender estas descobertas através de seus próprios scripts, contando que estes forneçam o dado em um formato em particular usando o protocolo JSON.

A arquitetura geral do processo de descoberto pode ser definido assim:

- Primeiro um usuário cria uma regra de descoberta em *Configuração → Templates*“ ou *Configuração → Hosts* e clica no link *Descoberta* da linha do host/template que se desejar. A configuração de um LLD pode ser separada em duas fases:
  - Definição de um item capaz de descobrir os elementos de configuração de interesse (por exemplo, sistemas de arquivo ou interfaces de rede);
  - Definição dos protótipos de itens, triggers e gráficos que poderão ser criados dinamicamente usando as informações descobertas.

Um item que descobre os elementos necessários é como qualquer outro item: O Zabbix solicita ao Zabbix Agent (ou através de qualquer outro tipo de item) um valor para o item, e o agente responde com um valor textual. A diferença aqui é que o valor contêm uma lista de elementos descobertos no formato JSON. Enquanto os detalhes deste formato são importantes somente para quem for implementar uma regra customizada de descoberta, é necessário que se saiba que o valor retornado conterá uma lista de pares no padrão *macro → valor*. Por exemplo, o item `net.if.discovery` pode retornar dois pares: ”{#IFNAME}“ → “lo” e ”{#IFNAME}“ → “eth0”.

Os itens de LLD “vfs.fs.discovery”, “net.if.discovery” e a descoberta de OIDs SNMP são suportados desde o Zabbix 2.0.
O item “system.cpu.discovery” desde o Zabbix 2.4.
A descoberta através de “querys SQL OBDC” desde o Zabbix 3.0.
Os valores de retorno de um LLD são limitados a 2048 bytes no Zabbix Proxy rodando com o BD IBM DB2. Este limite não se aplica ao Zabbix Server pois os valores são processados sem precisarem de armazenamento prévio.

As macros usadas nos nomes, chaves e outros campos de protótipos serão substituídas pelos seus valores descobertos quando as entidades reais (itens, triggers, gráficos, ou até mesmo outros hosts)

forem criados. Consulte a lista completa de opções para as macros LLD.

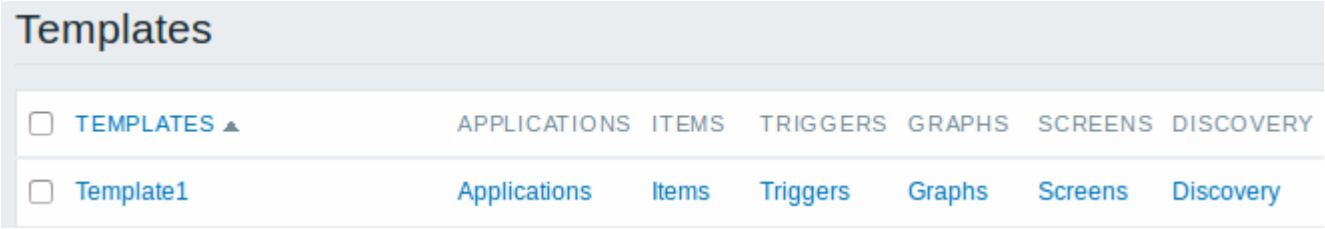
Quando o Zabbix Server recebe um valor de um item de descoberta, ele consulta os pares *macro* → *valor* para criar as entidades, conforme protótipos definidos na regra. No exemplo anterior, com o "net.if.discovery", o Zabbix Server irá criar um conjunto de itens, triggers e graphs para a interface de 'loopback' e outro para a interface "eth0".

As próximas seções vão demonstrar o processo em detalhes e servem como um "how-to" para configurar todos os tipos de descoberta antes citados. A última seção descreve o formato JSON para itens de descoberta e provê um exemplo de como implementar a sua própria regra de descoberta de sistema de arquivos usando um script Perl.

3.1 Descoberta de sistemas de arquivos

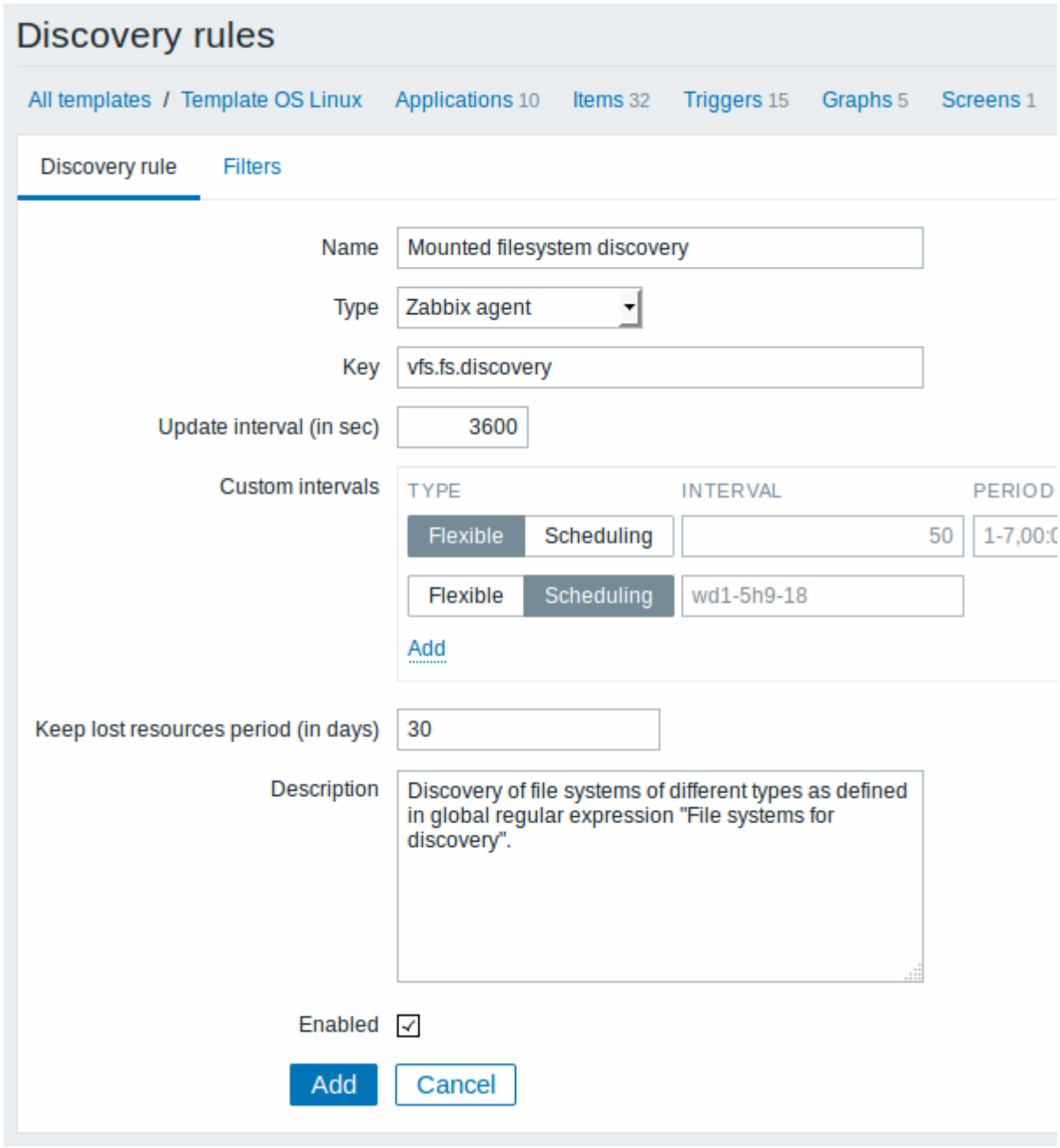
Para descobrir um sistema de arquivos:

- Acesse *Configuração* → *Templates*
- Clique no link *Descoberta* da linha apropriada



- Clique no botão *Criar regra de descoberta* situado no canto direito da barra de título
- Preencha o formulário com os detalhes necessários

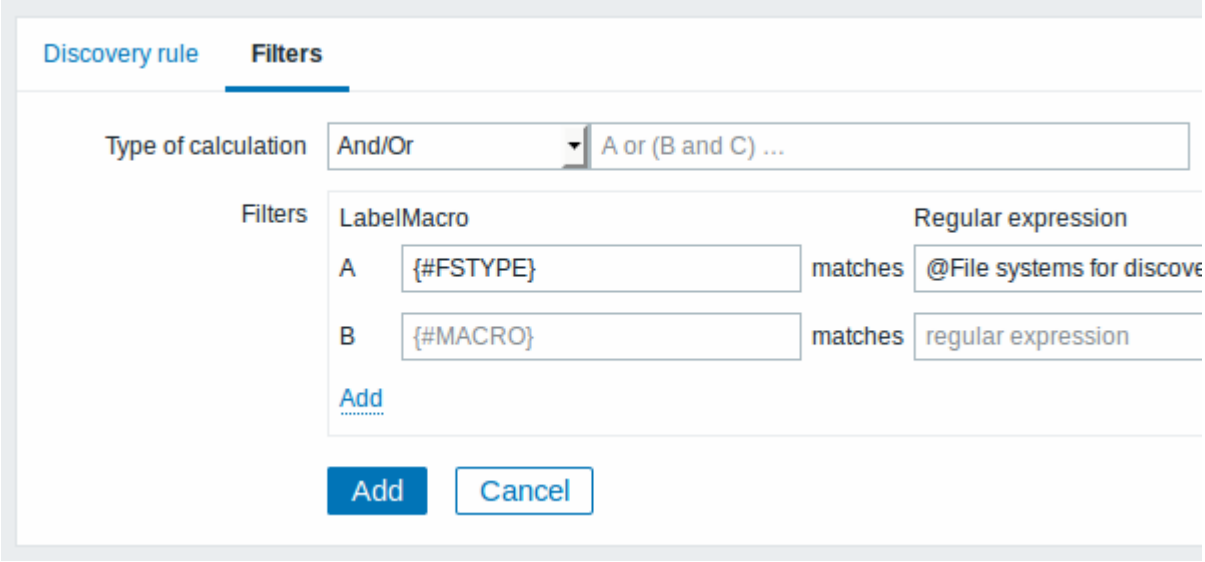
A aba **Regra de descoberta** contêm atributos gerais da regra de descoberta (do item de descoberta):



Parâmetro	**Descrição**
Nome	Nome da regra de descoberta.
Tipo	Tipo da verificação a ser executada; pode ser *Agente Zabbix (passivo)* ou *Agente Zabbix (ativo)* neste caso.
Chave	Um item com a chave “vfs.fs.discovery” foi desenvolvido no Zabbix Agent para permitir a descoberta de sistemas de arquivos em várias plataformas (consulte a lista de itens suportados), e retorna um JSON com a lista de sistemas de arquivos e seus tipos.
Intervalo de atualização (em seg)	Este campo define de quanto em quanto tempo o Zabbix irá executar a descoberta. No início, quando você está configurando a regra de descoberta, pode ser interessante definir um intervalo curto, mas uma vez que você tenha confiança que sua regra está buscando o que você precisa, modifique o tempo para, no mínimo, 30 minutos. Esta recomendação se deve ao fato que os sistemas de arquivo não mudam tanto assim. *Nota*: Se você definir o intervalo de atualização com o valor '0', a coleta do item não será agendada, entretanto, se você também definir um intervalo flexível, o item será coletado somente nos momentos definidos no intervalo flexível.

Parâmetro	Descrição
Intervalo customizado	Você pode criar regras customizadas de coleta para o item: **Flexível** - cria uma exceção ao *Intervalo de atualização* (intervalo com frequência diferenciada) **Agendamento** - cria um agendamento de coleta. Para maiores detalhes consulte o manual de Intervalos customizados. Agendamento é suportado desde o Zabix 3.0.0.
Manter dados de recursos perdidos por (em dias)	Este campo define por quantos dias serão mantidos os recursos que foram descobertos em algum momento e deixaram de ser percebidos (máximo 3650 dias). *Nota:* Se for definido como “0” as entidades criadas com base nestes recursos serão imediatamente excluídas. O uso do valor “0” não é recomendado pois erros de edição em filtros poderão fazer com que todas as entidades criadas dinamicamente e os dados históricos coletados sejam apagados.
Descrição	Descrição da regra.
Ativo	Se marcado, a regra será processada.

A aba **Filtros** contêm as definições de filtros a serem aplicados após a coleta do item de descoberta e antes da criação das entidades:

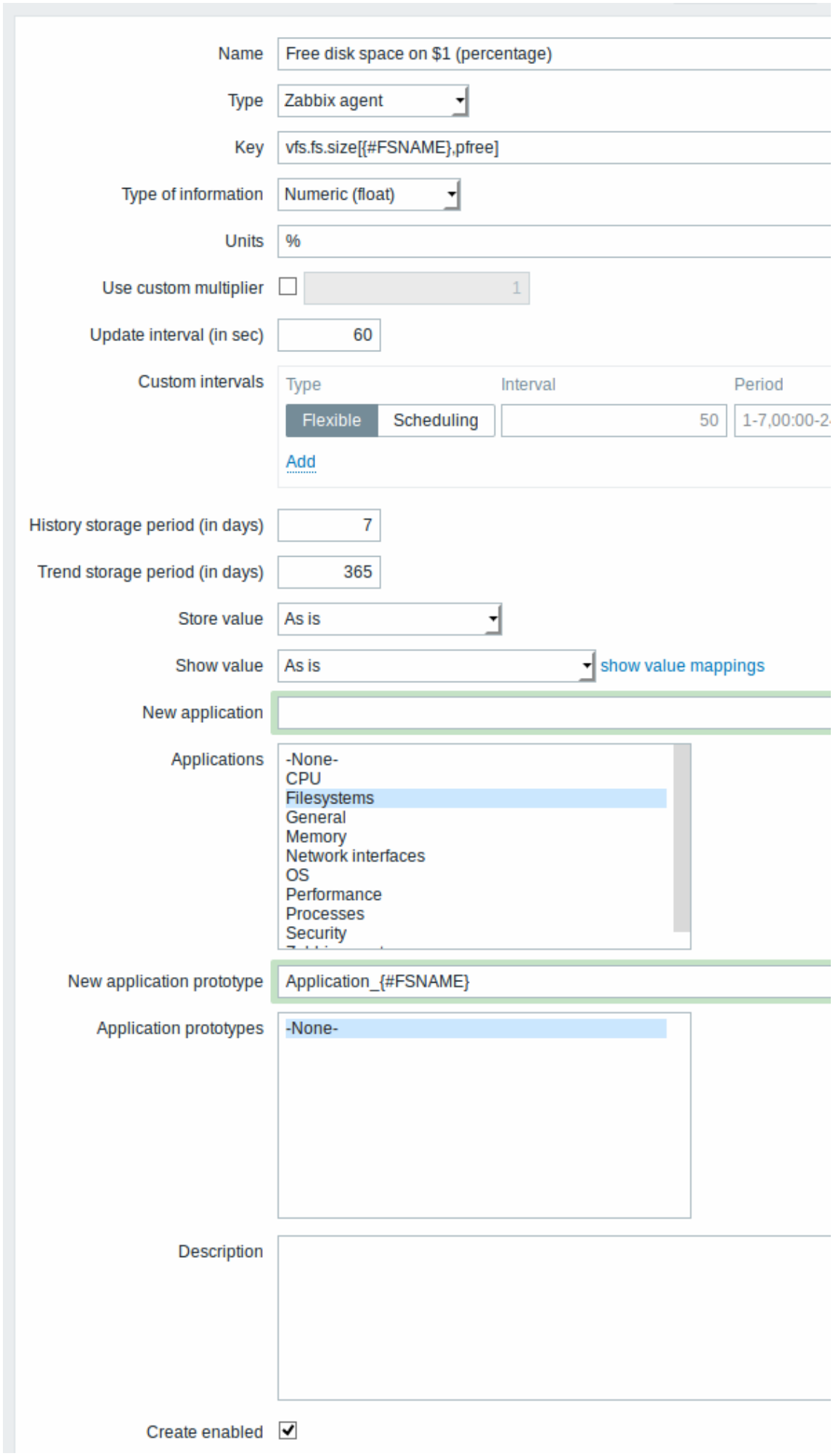


Parâmetro	Descrição
Tipo do cálculo	As seguintes opções podem ser utilizadas para aplicar os filtros em cada valor constante na lista descoberta: **E** - deve ser compatível com todos os filtros; **Ou** - suficiente se passar em um dos filtros; **E/Ou** - usa *E* com nomes de macro diferentes e *Ou* em macros com o mesmo nome; **Expressão personalizada** - oferece a possibilidade de definir forma customizada de cálculo dos filtros. A fórmula deverá conter todos os filtros na lista. Limitado à 255 símbolos.

Parâmetro	Descrição
Filtros	Um filtro pode ser usado para gerar itens, triggers e gráficos apenas para determinados sistemas de arquivos. O filtro deverá seguir o padrão POSIX estendido de expressões regulares. Por exemplo, se você estiver interessado somente nos drives: C:, D:, e E:, você pode preencher o campo *Macro* com o valor {#FSNAME} e o campo *Expressão regular* com o valor "^C|^D|^E". Também é possível filtrar usando outra macro, a {#FSTYPE} (ex. "^ext|^reiserfs") e tipos de drives (apenas no Windows) usando a macro {#FSDRIVETYPE} (ex. "fixed"). Você pode informar uma expressão regular diretamente ou usar uma expressão regular global. Para testar a expressão uma expressão regular você pode utilizar o comando “grep -E”, por exemplo: `for f in ext2 nfs reiserfs smbfs; do echo $f | grep -E '^ext|^reiserfs' || echo "SKIP: $f"; done` A macro {#FSDRIVETYPE} no Windows é suportada desde o Zabbix **3.0.0**. A definição de vários filtros é suportada desde o Zabbix **2.4.0**. Observe que se alguma macro definida no filtro estiver ausente, o recurso em questão será ignorado.

O BD do Zabbix em MySQL precisa ser criado de forma sensível ao caso se os nomes de sistemas de arquivo puderem ser diferenciados apenas pelo caso, senão a regra de descoberta poderá não funcionar.
O histórico das coletas da regra de descoberta não é preservada.

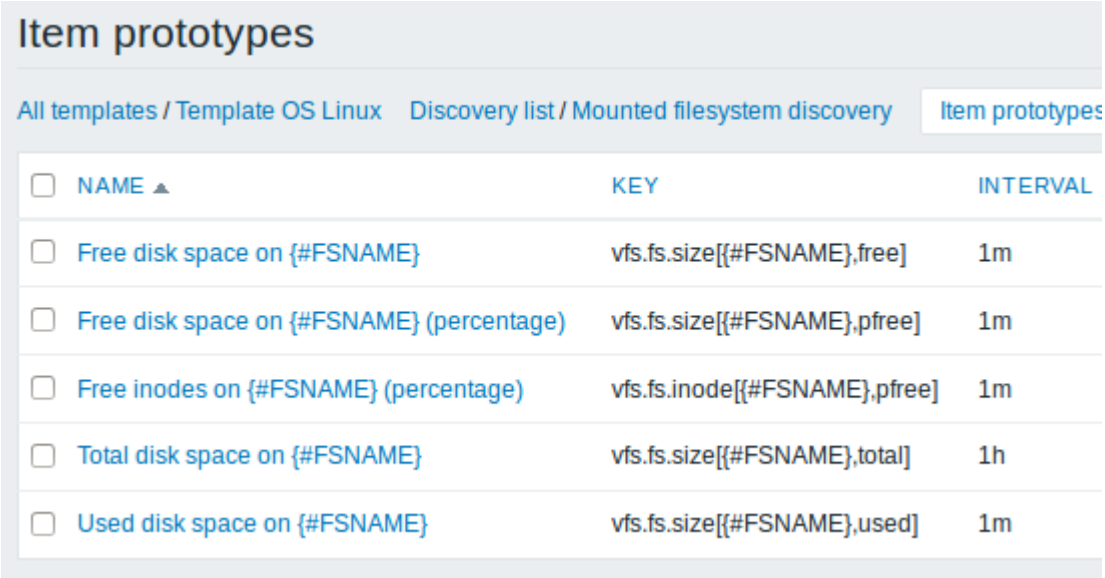
Uma vez que uma regra tenha sido criada, vá para use o link *Protótipos de item*, disponível na lista de regras de descoberta do host/template, e clique no botão *Criar protótipo de item*. Observe que a macro {#FSNAME} será utilizada aqui, seu uso neste é obrigatório, pois cada item a ser criado deverá ser diferente do item criado anteriormente.


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

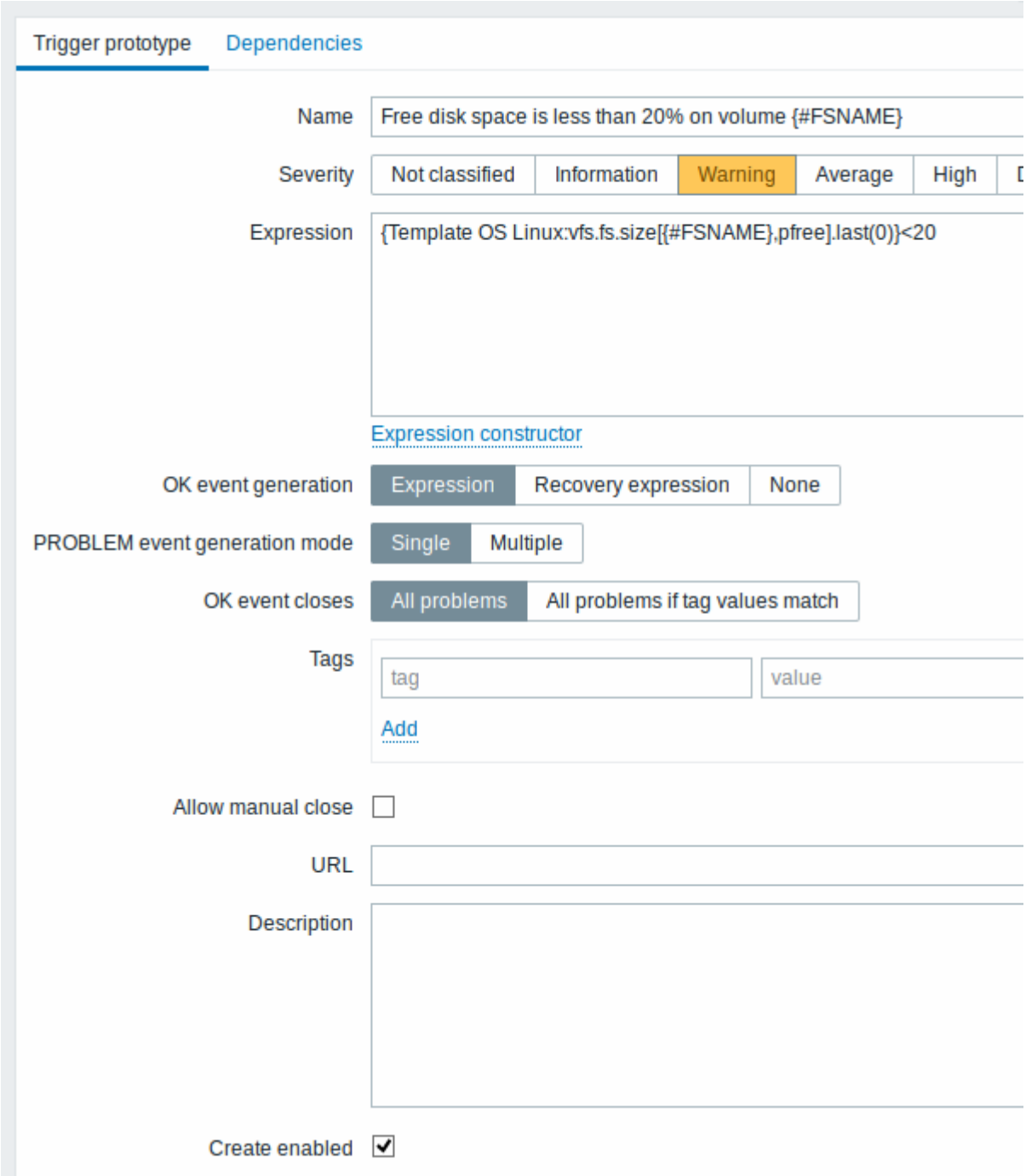
Se um protótipo de item for criado com o status *Inativo*, os itens continuarão a ser criados a partir dele mas estarão com o status *Inativo* também.

O campo *Protótipo de aplicação* é uma opção disponível especificamente nos protótipos de itens. Um protótipo de aplicação permite que você crie dinamicamente as aplicações e as associe aos itens. Consulte mais nas notas sobre descoberta de aplicações.

Nós podemos criar vários protótipos de item para cada métrica de sistema de arquivo que nos interesse:

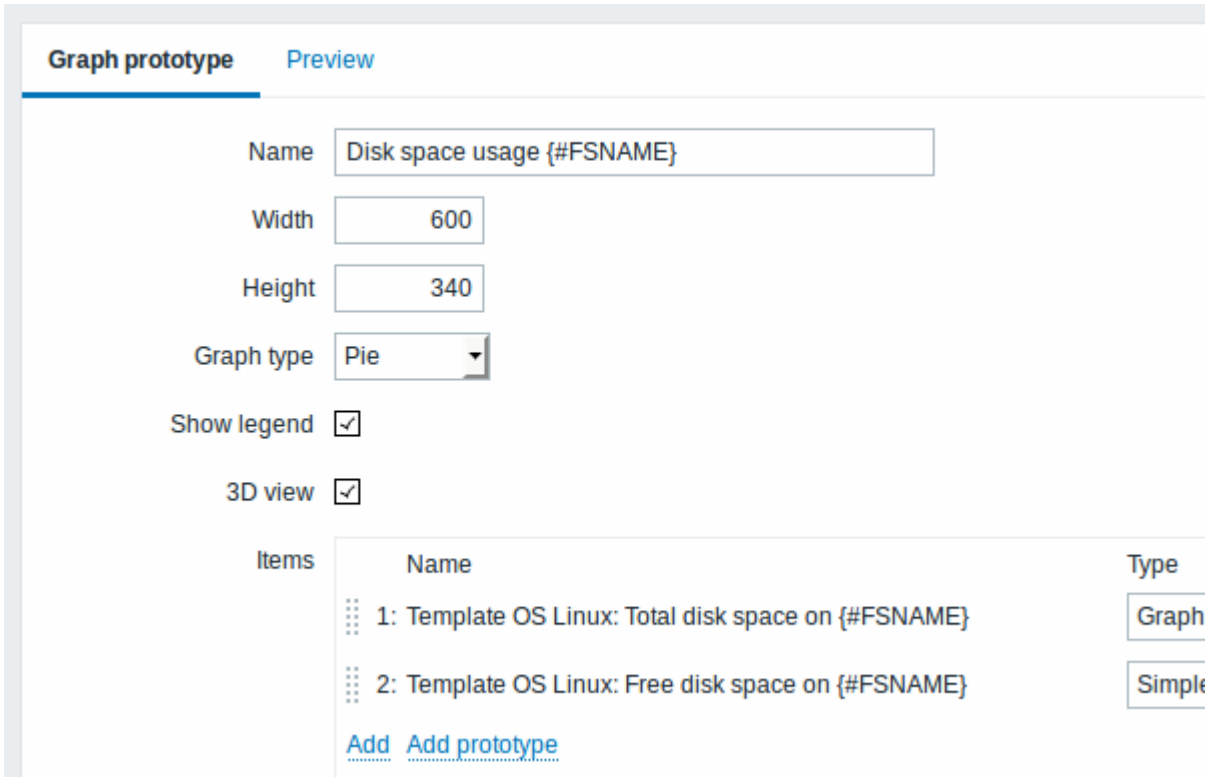


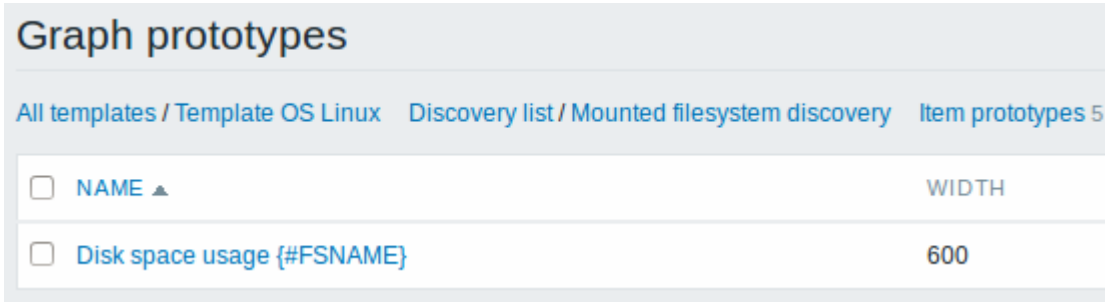
Então, criamos protótipos de trigger de forma similar:



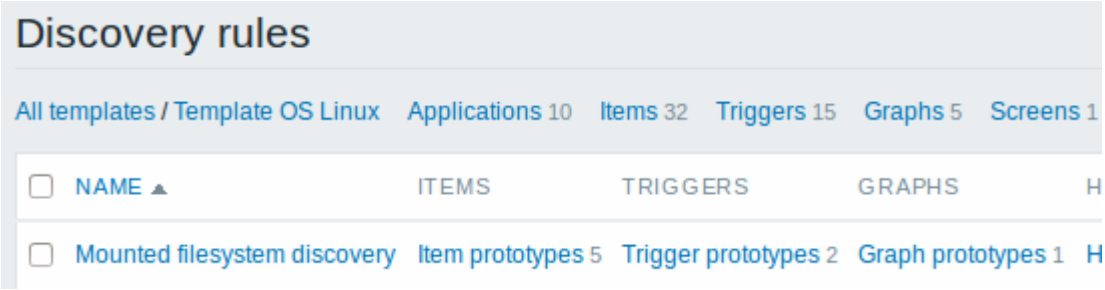
Você pode definir dependências entre protótipos de trigger também (suportado desde o Zabbix 3.0). Para fazer isso, acesse a **Dependências**. Um protótipo de trigger poderá depender de outro protótipo de trigger da mesma regra de descoberta ou de outra trigger normal. Não é possível criar relações de dependência entre protótipos de trigger de diferentes regras de descoberta. Um protótipo de trigger em nível de host não poderá depender de uma trigger em nível de template.

Nós podemos criar protótipos de gráfico também:



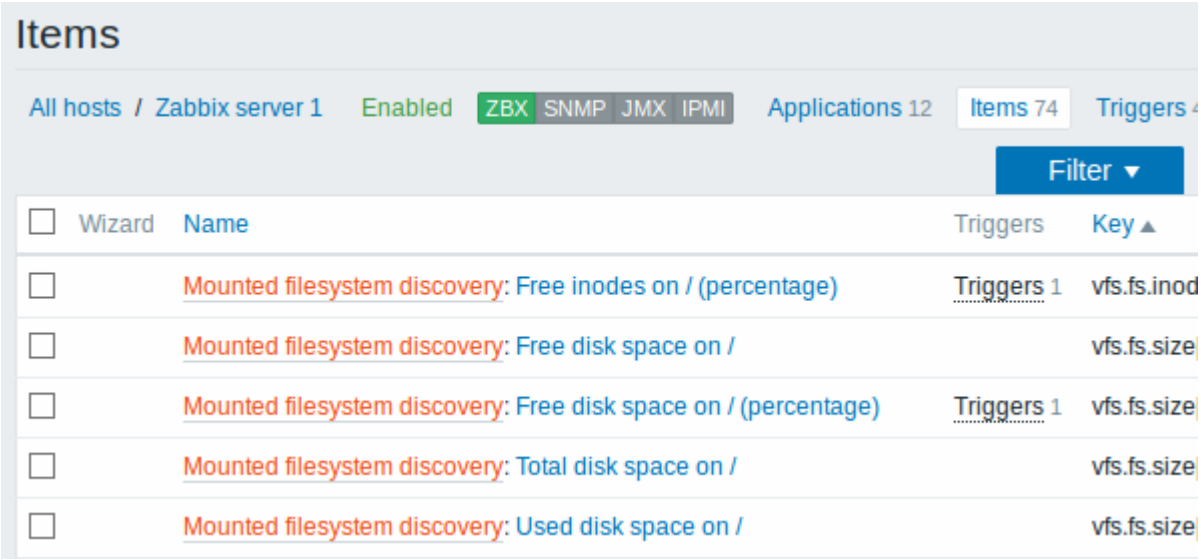


Finalmente, nós criamos a regra de descoberta e tudo parece que irá funcionar. Temos cinco protótipos de itens, dois protótipos de trigger e um protótipo de gráfico.



Nota: Para configurar um protótipo de host consulte o manual de protótipos de host na área de monitoração de máquinas virtuais.

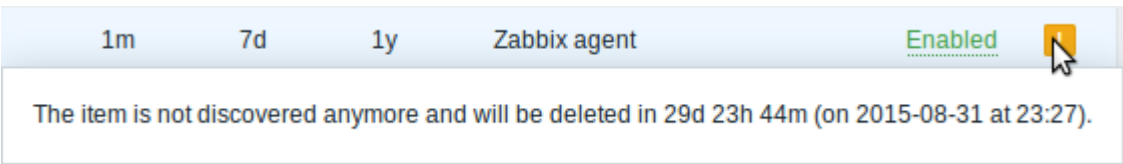
As telas a seguir ilustram como itens, triggers e gráficos que foram criados pelo processo de descoberta irão se parecer na tela de configuração de um host. As entidades descobertas serão prefixadas por um link laranja com o nome de sua regra de descoberta de origem.



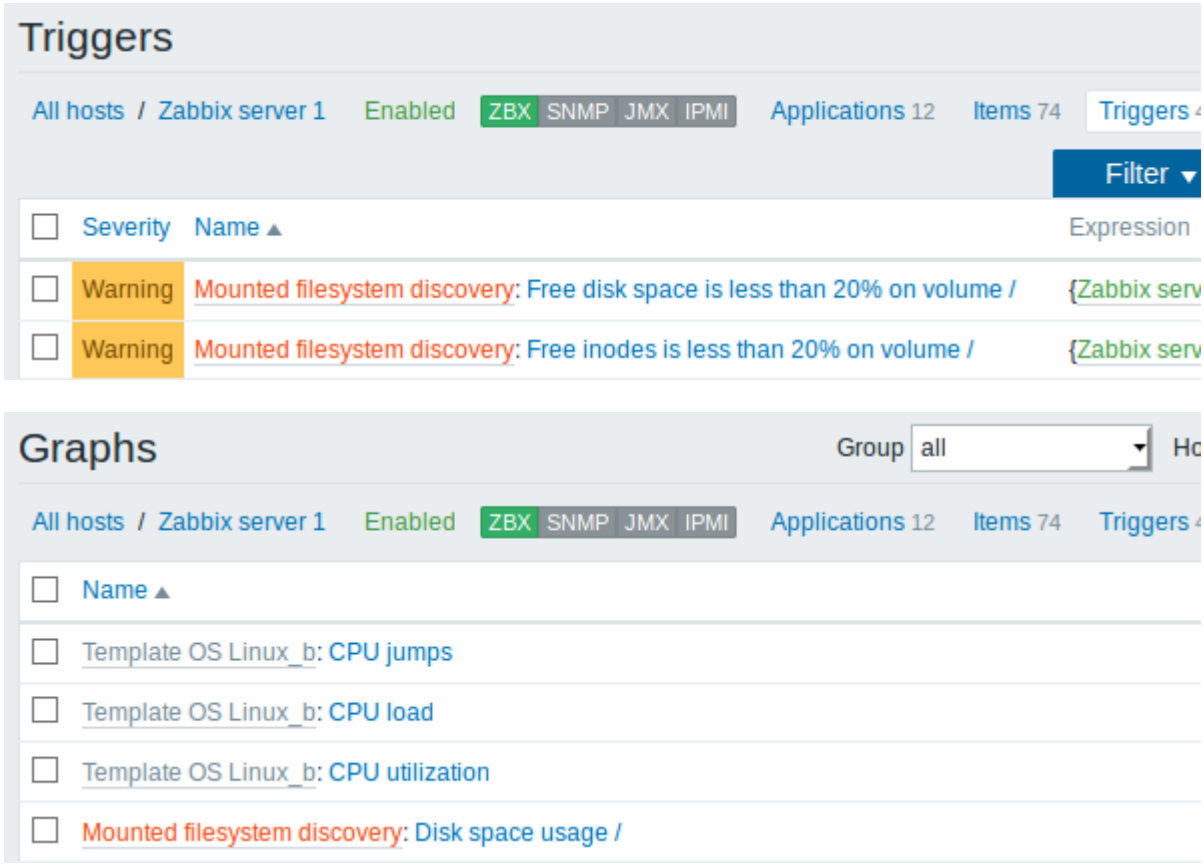
Observe que não serão criadas entidades para recursos caso já exista outra entidade com o mesmo critério de unicidade (por exemplo outro item com a mesma chave, ou gráfico com o mesmo nome).

Os itens (de forma similar às triggers e gráficos) criados por um LLD não poderão ser excluídos manualmente. Entretanto, poderão ser excluídos automaticamente se a regra de descoberta não conseguir mais localizar o recurso, ou se seu filtro excluir o recurso da lista. Neste caso os itens, triggers e gráficos serão apagados após o período definido no campo *Manter dados de recursos perdidos por (em dias)*.

Quando as entidades passam para o estado 'Não foi mais descoberto', um indicador do tempo restante de vida será apresentado para o item na lista. Ao posicionar o mouse sobre o ícone de alerta será apresentada uma mensagem indicando quanto tempo falta para o item e seu histórico serem excluídos.



Se as entidades forem marcadas para exclusão, mas não forem excluídas no momento previsto (por exemplo por causa da regra de descoberta ter sido inativada), eles serão excluídos no próximo momento que a regra for processada.



3.2 Descoberta de interfaces de rede

A descoberta de interfaces de rede é feita exatamente da mesma forma que a descoberta de sistemas de arquivos. O que muda é o nome da chave buscada (“net.if.discovery” ao invés de “vfs.fs.discovery”) e o nome das macros retornadas ({#IFNAME} ao invés de {#FSNAME}).

Exemplos de protótipos de item que você definir usando a chave “net.if.discovery”: “net.if.in[{#IFNAME},bytes]”, “net.if.out[{#IFNAME},bytes]”. Clique aqui para maiores detalhes.

3.3 Descoberta de CPUs e núcleos de CPU

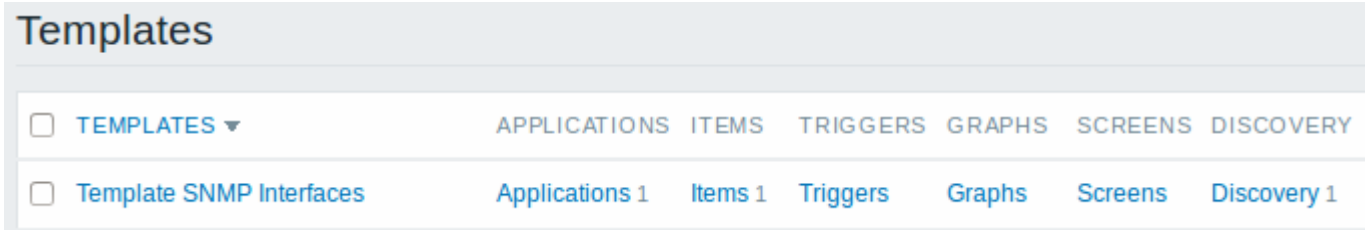
A descoberta de CPUs e núcleos de CPU da mesma forma que as anteriores, mudando novamente a chave de descoberta: “system.cpu.discovery”. Ela também retorna duas macros: `{#CPU.NUMBER}` e `{#CPU.STATUS}`, respectivamente o número de sequencia da CPU e seu status. Pode não ser possível uma distinção clara entre processadores físicos, núcleos e 'hyperthreads'. Em ambientes Linux, UNIX e BSD a macro {#CPU.STATUS} retorna o status do processador, que poderá ser “online” ou “offline”. Em ambientes Windows, esta macro poderá conter um terceiro valor - “unknown” - que indicará que o processador foi detectado mas nenhuma informação foi coletada ainda sobre ele.

A descoberta de CPU utiliza estatísticas dos processos de coleta do agente para manter a consistência dos dados providos e economizar recursos ao obter os dados. Isso afeta a coleta desta chave quando se tenta utilizar a opção '-t' na linha de comando com o binário do agente diretamente, esta chave será devolvida com o status de NOT_SUPPORTED, acompanhada da mensagem que o coletor não foi iniciado.

Os protótipos de item que podem ser criados com esta descoberta incluem "system.cpu.load[{#CPU.NUMBER}, <mode>]", "system.cpu.util[{#CPU.NUMBER}, <type>, <mode>]", dentre outros.

3.4 Descoberta de OIDs SNMP

Neste exemplo, nós vamos executar a descoberta SNMP em um switch. Acesse *Configuração* → *Templates*.



Para editar as regras de descoberta do template desejado, clique no link *Descoberta* de sua linha.

Para criar uma nova regra de descoberta, use o botão *Criar regra de descoberta* localizado no canto direito da barra de título.

Diferentemente das descobertas de sistemas de arquivos e de interfaces de rede, esta regra não necessita de uma chave "snmp.discovery", o uso do tipo SNMP `agent` é suficiente.

Os OIDs serão descobertos no seguinte formato: `discovery[{#MACRO1}, oid1, {#MACRO2}, oid2, …,]`

Onde *{#MACRO1}*, *{#MACRO2}* ... são nomes válidos de macro e *oid1*, *oid2*... são identificadores OIDs capazes de gerar valores para estas macros. Uma macro pré-definida *{#SNMPINDEX}* conterá o índice do OID descoberto que poderá ser aplicado aos recursos descobertos. Os recursos estarão agrupados através da macro *{#SNMPINDEX}*.

Para entender o que isso significa, vamos executar alguns `snmpwalks` em nosso switch:

```
$ 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
```

E definir o OID SNMP para: `discovery[{#IFDESCR}, ifDescr, {#IFPHYSADDRESS}, ifPhysAddress]`

Agora esta regra irá descobrir os recursos com as macros {#IFDESCR} definidas como **WAN**, **LAN1** e **LAN2**, as macros {#IFPHYSADDRESS} terá os valores **8:0:27:90:7a:75**, **8:0:27:90:7a:76**, e **8:0:27:2b:af:9e**, e as macros {#SNMPINDEX} com os índices **1**, **2** e **3**:

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

Se um recurso não existir no OID especificado então a macro correspondente para ele será omitida, por exemplo:

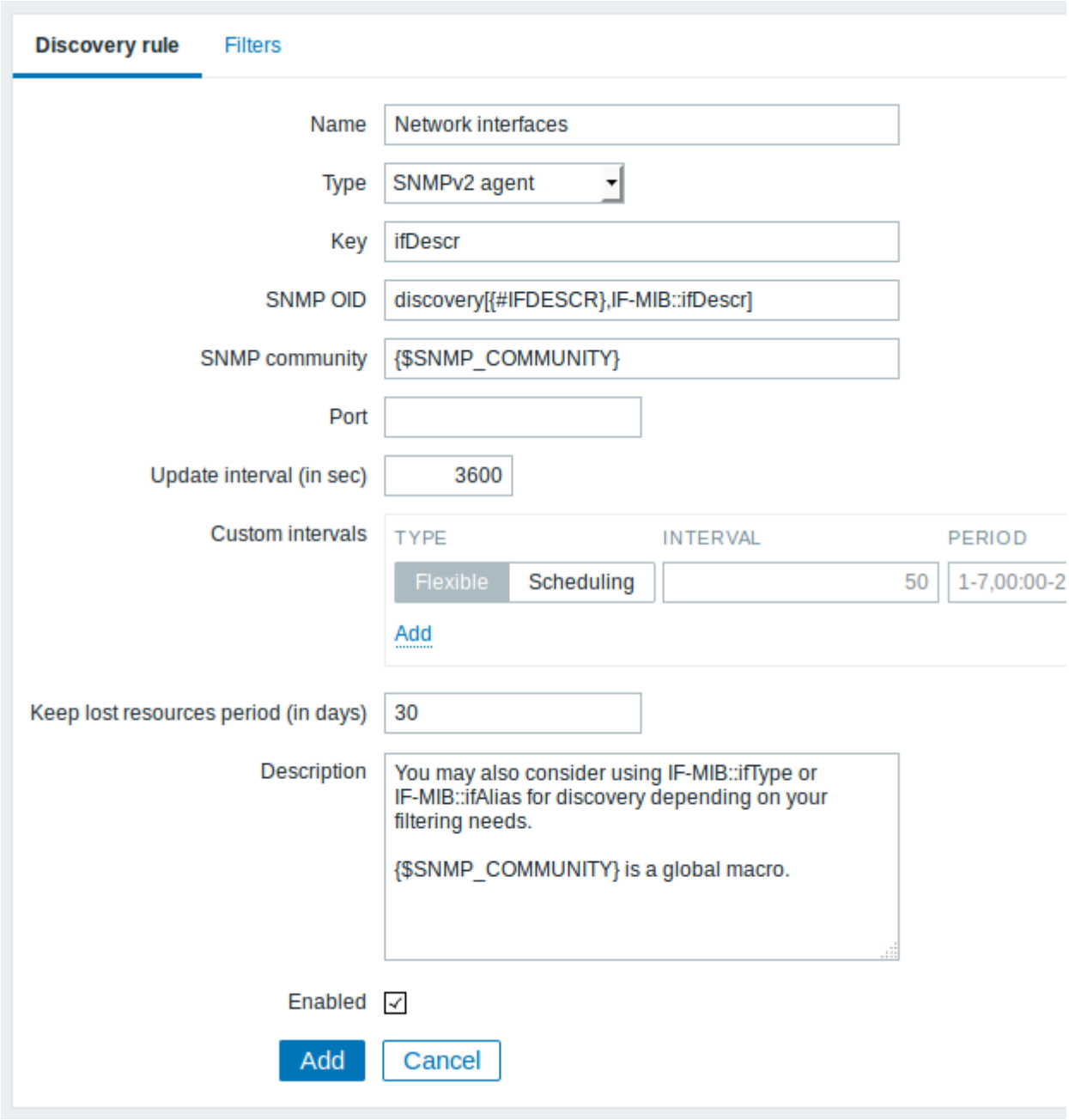
```
ifDescr.1 "Interface #1"
ifDescr.2 "Interface #2"
ifDescr.4 "Interface #4"

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

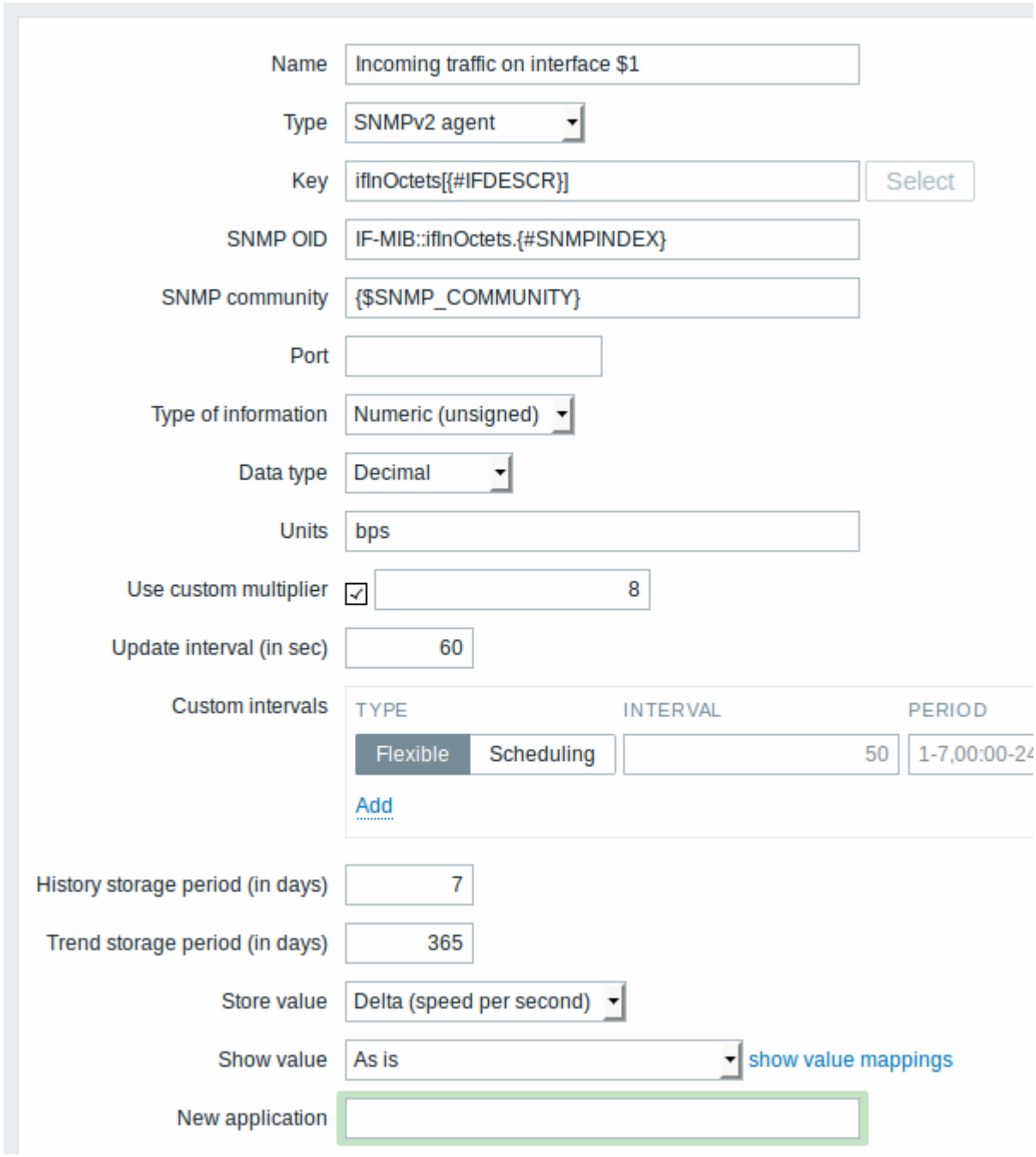
Então, neste caso, a descoberta SNMP `discovery[{#IFDESCR}, ifDescr, {#IFALIAS}, ifAlias]` retornará a seguinte estrutura JSON:

```
{
    "data": [
        {
            "{#SNMPINDEX}": 1,
            "{#IFDESCR}": "Interface #1",
            "{#IFALIAS}": "eth0"
        },
        {
            "{#SNMPINDEX}": 2,
            "{#IFDESCR}": "Interface #2",
            "{#IFALIAS}": "eth1"
        },
        {
            "{#SNMPINDEX}": 3,
```

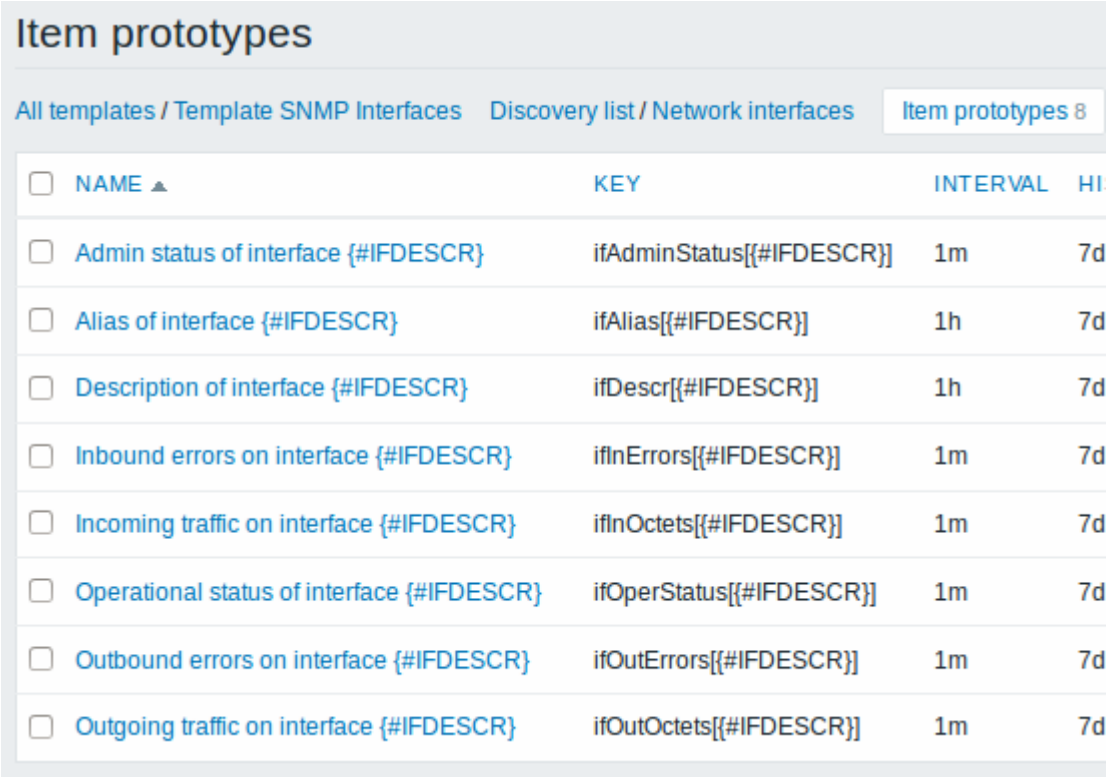
```
            "{#IFALIAS}": "eth2"
        },
        {
            "{#SNMPINDEX}": 4,
            "{#IFDESCR}": "Interface #4"
        },
        {
            "{#SNMPINDEX}": 5,
            "{#IFALIAS}": "eth4"
        }
    ]
}
```



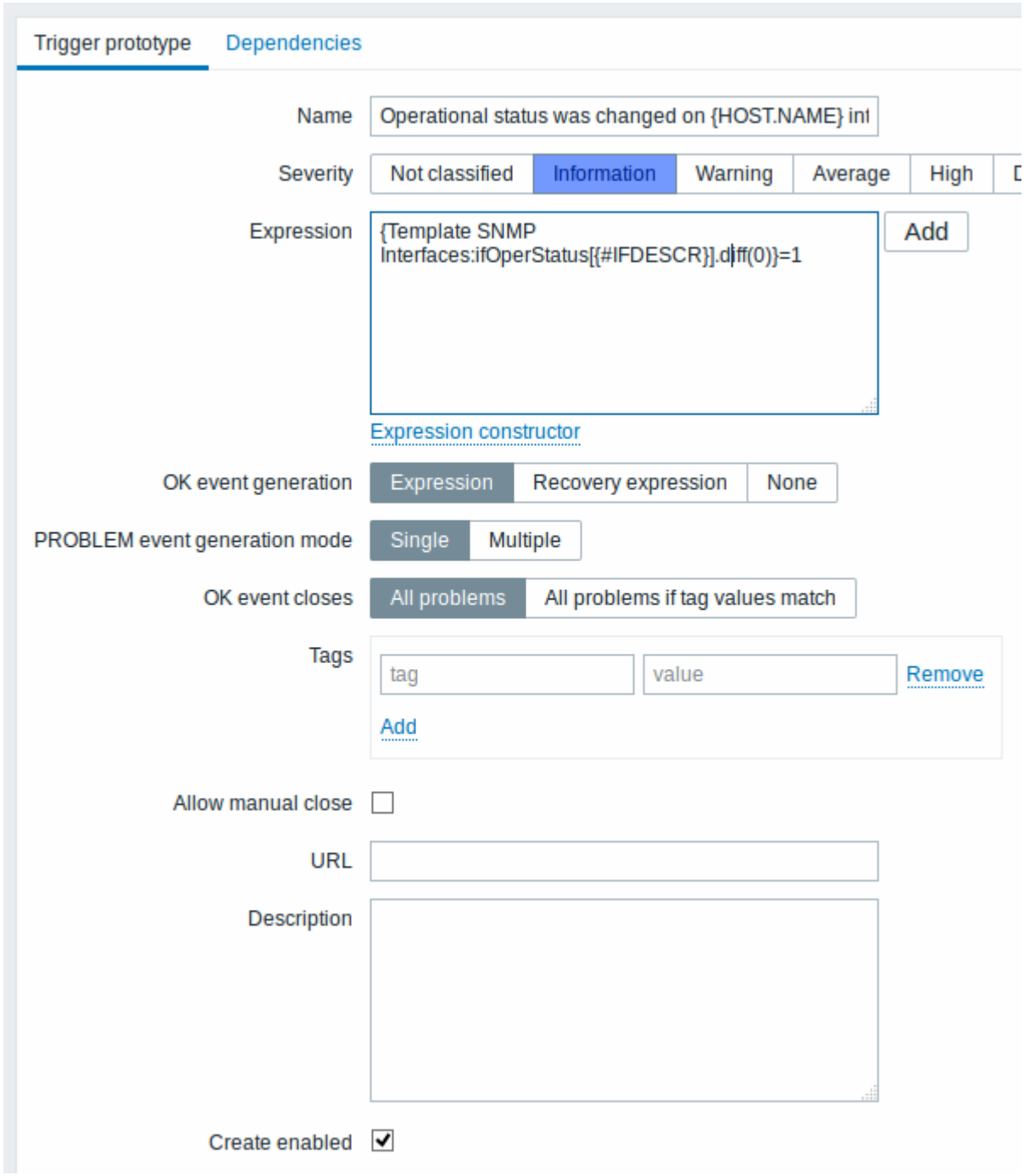
Estas telas ilustram como as macros podem ser utilizadas em protótipos de item:

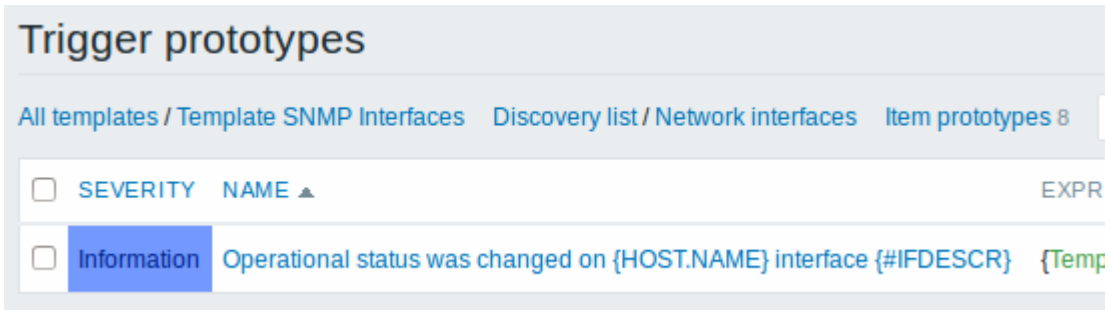


Da mesma forma, criando quantos itens quanto o necessário:

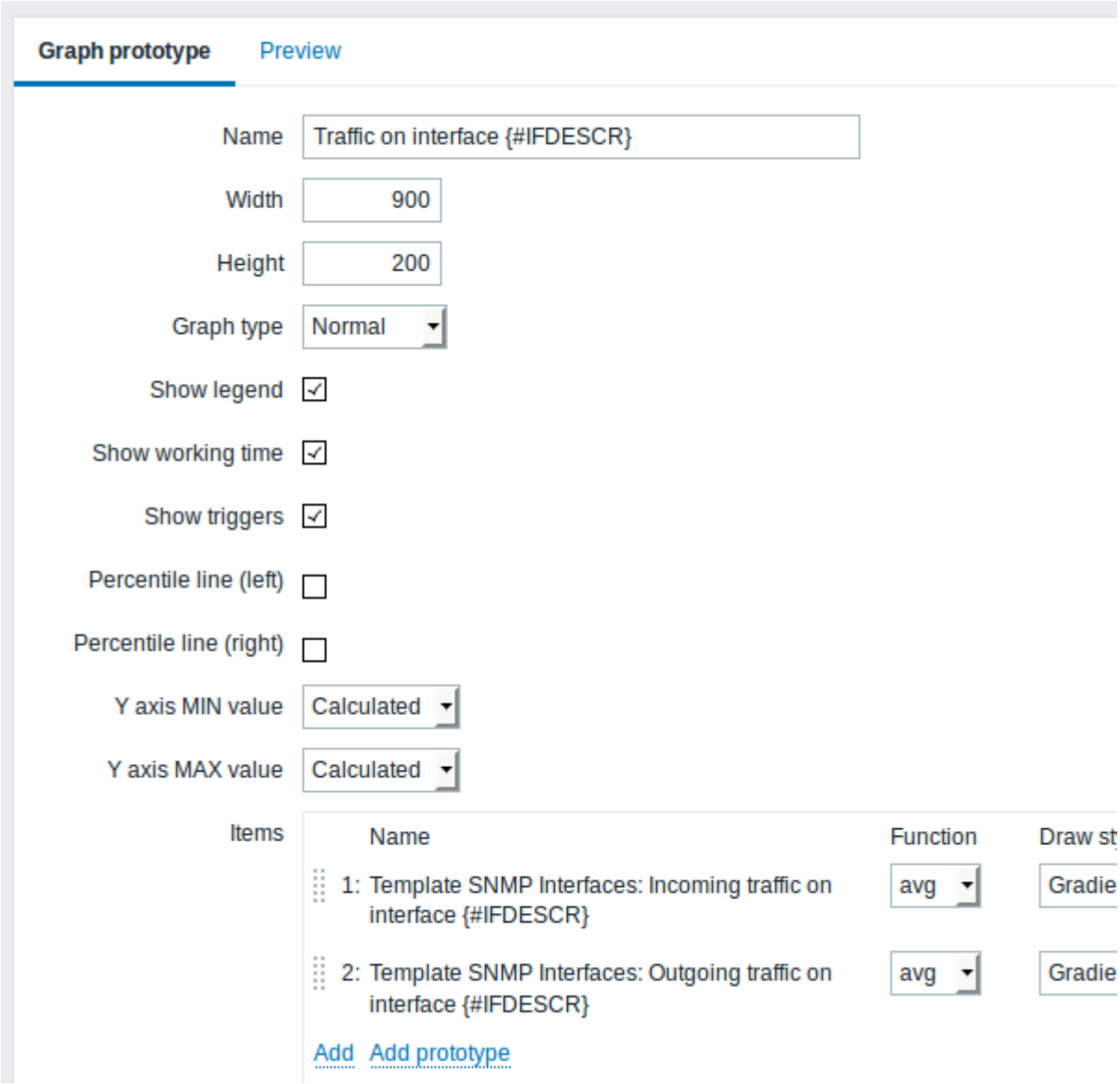


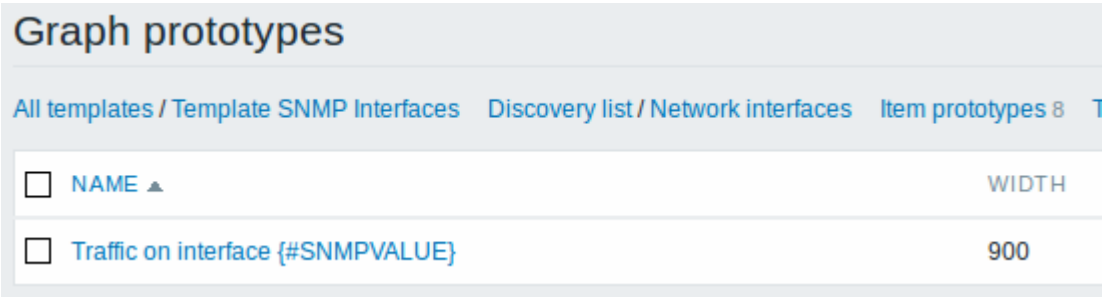
Protótipos de trigger:



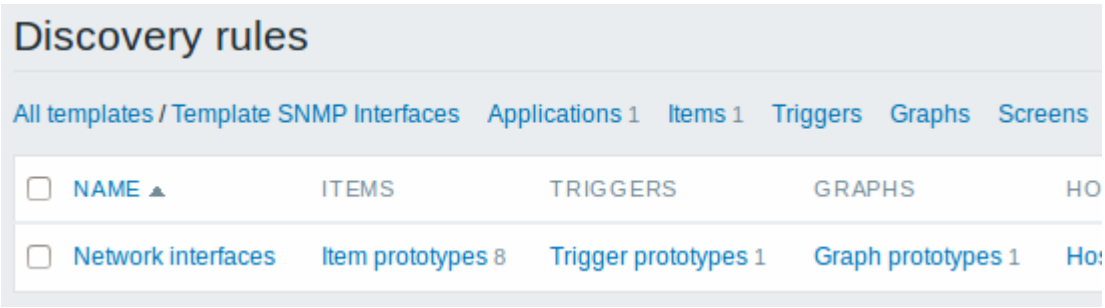


Protótipos de gráfico:



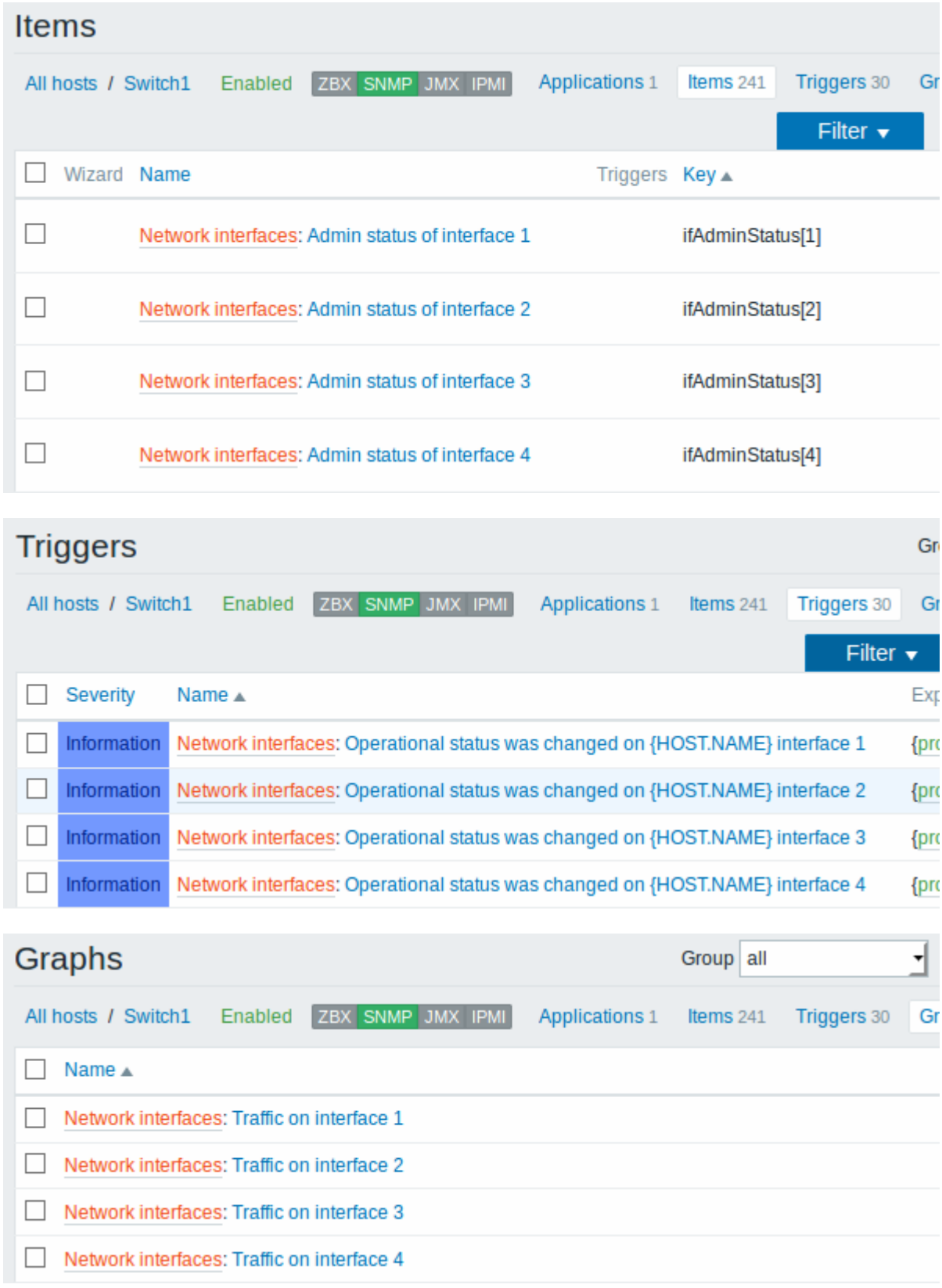


Um sumário de nossa regra de descoberta:



Quando o servidor executa, ele cria itens, triggers e gráficos reais a partir dos valores retornados pela

descoberta SNMP. Na configuração do host eles estarão prefixados com o nome da regra de descoberta que os originou e na cor laranja.



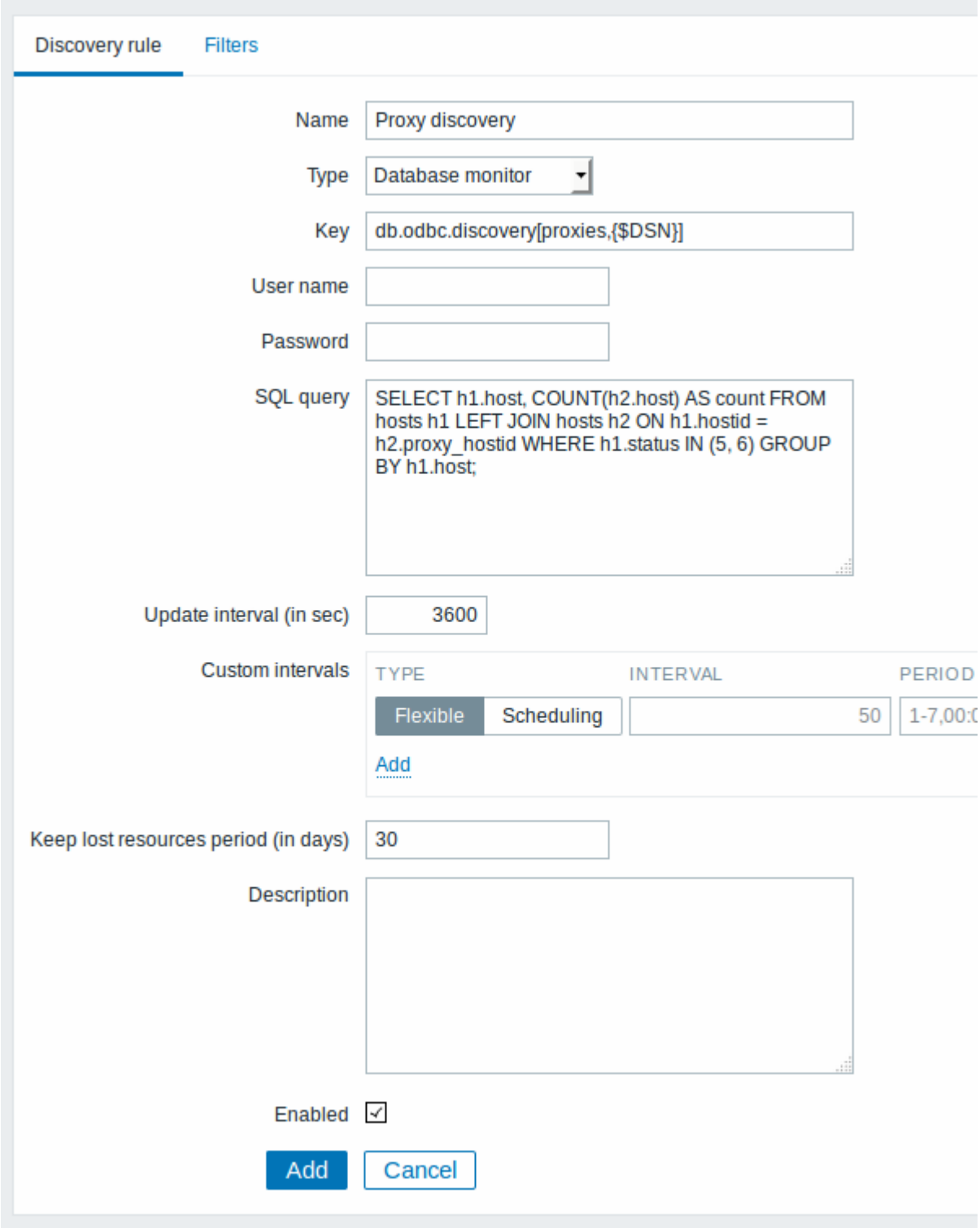
3.5 Descoberta usando consultas SQL/ODBC

Este tipo de descoberta é feito através de consultas SQL, cujos resultados são transformados

automaticamente em um objeto JSON aceitável pelo LLD. As consultas SQL são executadas com items do tipo “Monitoração de banco de dados”. Consulte o manual sobre monitoração ODBC para maiores detalhes de como trabalhar com este tipo de item, a única diferença é que a chave a ser utilizada será “db.odbc.discovery[<description>,<dsn>]” ao invés de “db.odbc.select[<description>,<dsn>]”.

Um exemplo prático de como as consultas SQL são transformadas em um JSON pode ser a descoberta através de um proxy com consultas OBDC no banco do Zabbix. Isso pode ser útil para a criação automática de itens internos para monitorar os proxies “zabbix[proxy,<name>,lastaccess]”, verificando se estão ativos.

Vamos começar com a configuração da regra de descoberta:



Aqui vamos usar conexão direta com o BD do Zabbix para localizar todos os proxies, junto com a quantidade de hosts que eles monitoram. A quantidade de hosts pode ser utilizada, por exemplo, para filtrar proxies que não tem carga de monitoração sobre sí:

```
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)
```

Internamente a chave “db.odbc.discovery[]” irá receber este valor e transformar em um código JSON:

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

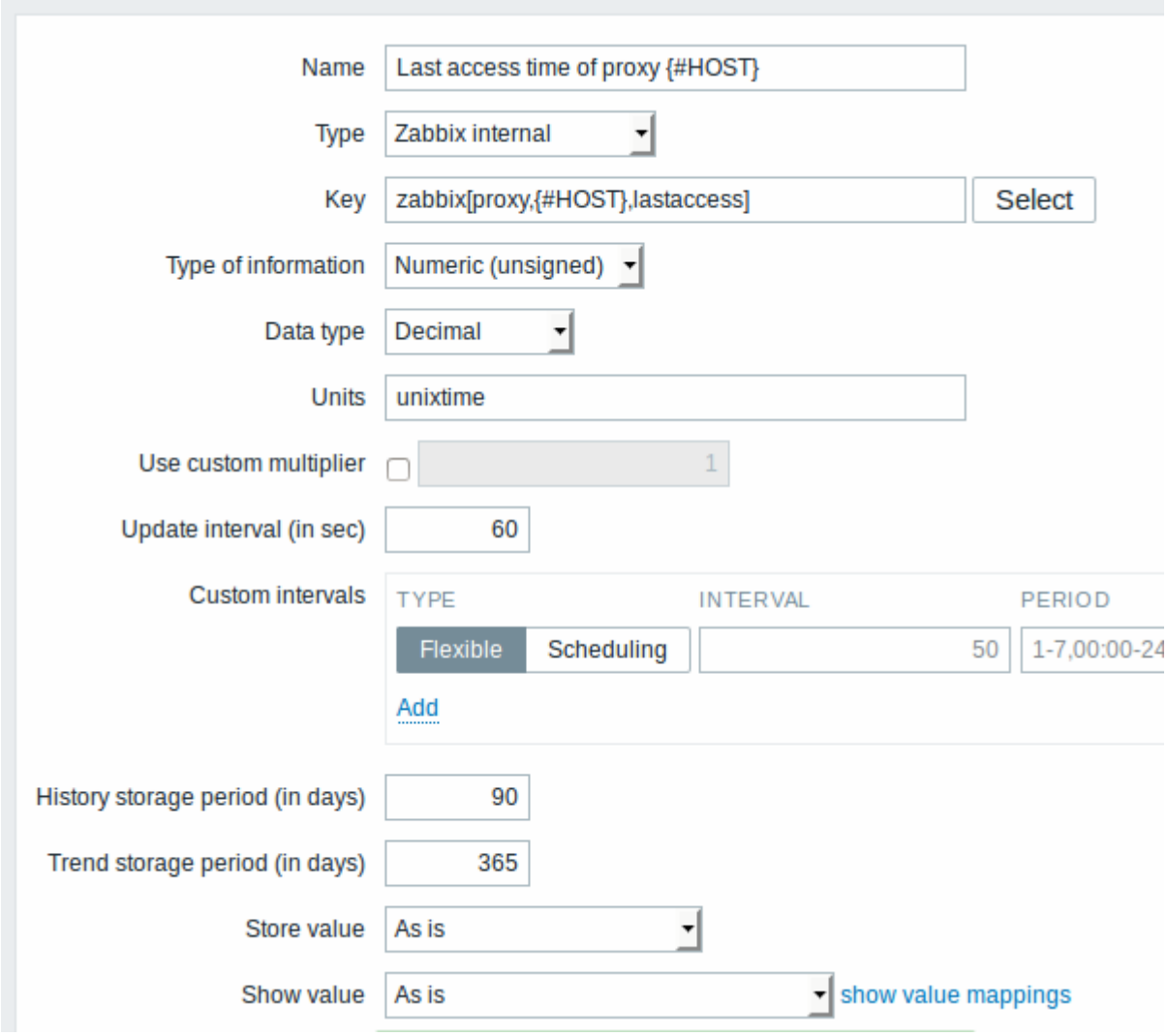
Os nomes das colunas da consulta foram automaticamente definidos como sendo os nomes de macro, seguidos de seus valores.

Se não estiver muito claro como um nome de coluna foi transformado em um nome de macro, sugerimos utilizar apelidos de colunas no exemplo acima (ex. “COUNT(h2.host) AS count”).

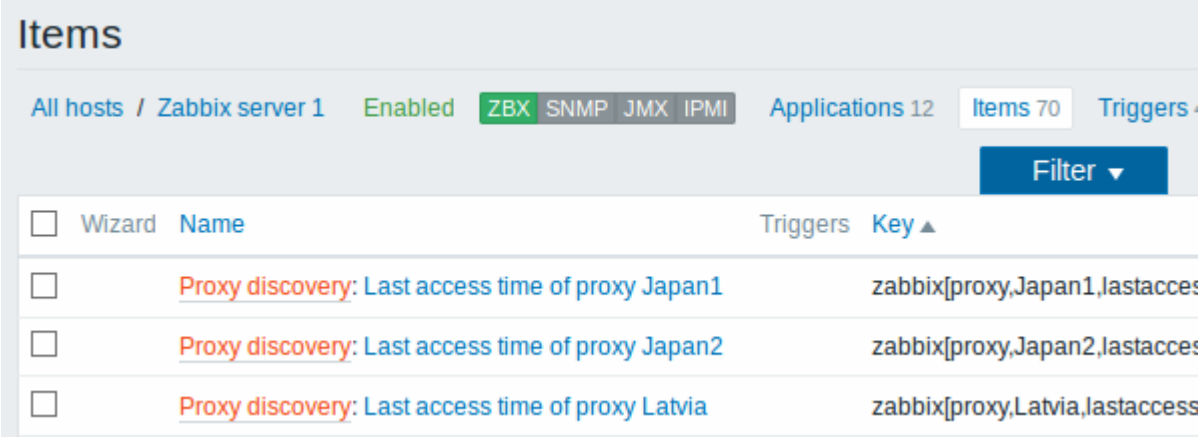
Caso o nome de alguma coluna não possa ser transformado em um nome válido de macro, a regra irá para o estado 'Não suportado' com mensagem de erro detalhando o problema com a coluna de número X. Se informações adicionais forem necessárias, poderão ser obtidas ao ativar o modo de debug 4 (**DebugLevel=4**) e consultar o log do Zabbix Server:

```
$ 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.
```

Agora conseguimos entender como uma consulta SQL é transformada em um objeto JSON e podemos utilizar a macro {#HOST} para criar os protótipos de itens:



Uma vez que a descoberta seja executada, um item será criado para cada proxy:



3.6 Descobertas de serviços no Windows

A descoberta de serviços no Windows é feita da mesma forma que a descoberta de sistemas de arquivos. A chave utilizada é a "service.discovery" as seguintes macros estarão disponíveis para uso nos filtros e protótipos de item/trigger/graph:

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

Através da descoberta de serviços do Windows você poderá criar protótipos de itens como "service.info[{#SERVICE.NAME},<param>]", onde *param* aceita os seguintes valores: *state*, *displayname*, *path*, *user*, *startup* ou *description*. Por exemplo, para obter o nome de exibição de um serviço você pode criar um item com a chave "service.info[{#SERVICE.NAME},displayname]". Se o parâmetro *param* não for definido ("service.info[{#SERVICE.NAME}]"), o valor padrão *state* será utilizado.

As macros {#SERVICE.STATE} e {#SERVICE.STATENAME} retornarão o mesmo conteúdo, entretanto, {#SERVICE.STATE} retornará a representação numérica (0-7), e {#SERVICE.STATENAME} a representação textual (*running*, *paused*, *start pending*, *pause pending*, *continue pending*, *stop pending*, *stopped* ou *unknown*). O mesmo se aplica às macros {#SERVICE.STARTUP} e {#SERVICE.STARTUPNAME}, onde uma retorna a representação numérica (0-4) e a outra textual (*automatic*, *automatic delayed*, *manual*, *disabled*, *unknown*).

3.7 Criando LLD customizado

Além de todos os tipos nativos de regra de descoberta, você também poderá criar suas próprias

regras de descoberta para retornar qualquer tipo de recurso - por exemplo, descobrir as bases de dados em um servidor de banco.

Para fazer isso precisamos de um item que retorne um JSON, definindo objetos e, opcionalmente, propriedades deles. A quantidade de macros por entidade não é limitada. Nas descobertas nativas normalmente são retornadas uma ou duas macros mas é possível retornar muito mais..

O formato JSON requerido é ilustrado neste exemplo, suponhamos que você esteja usando um Zabbix Agent 1.8 (bem antigo), ele não irá ter suporte nativo para a chave “vfs.fs.discovery”, mas ainda assim ele será capaz de executar a descoberta de sistemas de arquivo. Um simples script em Perl no Linux poderá descobrir os sistemas de arquivo montados e apresentar o resultado no formato JSON, indexando tanto os nomes quanto os seus tipos. Poderíamos criar, por exemplo, um **UserParameter** chamado “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+)/;
    $fsname =~ s!/!\\/!g;

    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";
```

Os caracteres permitidos em um nome de macro LLD são **0-9** , **A-Z** , **_** , **.**

Não são suportadas letras minúsculas em seus nomes.

Um exemplo de saída deste script (reformatado para melhor legibilidade) poderia ser algo como isso:

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

```
  { "{#FSNAME}":"\/",                                "{#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"  }
  ]
}
```

Assim, no *Filtro* da regra de descoberta você poderá, por exemplo, validar o conteúdo da macro "{#FSTYPE}" com uma expressão regular ("rootfs|ext3").

Você não precisa nomear as macros como FSNAME/FSTYPE em regras customizadas, use o nome que quiser contando que se adeque aos caracteres permitidos.