

Ganeti



query filters & job filters

Niklas Hambüchen

niklash@google.com

niklas@nh2.me

overview

query filters

filter your output – similar to `select * WHERE ...`
`gnt-something list --filter`

job filters

say what jobs can run – similar to `iptables`
`gnt-filter add --action=PAUSE --predicates=[...]`

overview

query filters

filter your output – similar to `select * WHERE ...`

```
gnt-something list --filter
```



can use now

job filters

say what jobs can run – similar to iptables

```
gnt-filter add --action=PAUSE --predicates=[ ...]
```



ganeti 2.13

query filters

work with

```
gnt-node      gnt-instance  gnt-job
gnt-group     gnt-backup    gnt-filter
```

examples

```
gnt-instance list -o name,be/memory
```

Instance	ConfigMaxMem
host1.google.com	128M
host2.google.com	256M

```
gnt-instance list --filter 'be/memory > 200'
```

host2.google.com	256M
------------------	------

query filters

Which nodes aren't "mynode"?

```
gnt-node list --filter "not(name == 'myhost.google.com')"
```

Which instances are using more than 3 virtual CPUs?

```
gnt-instance list -F 'oper_vcpus > 3'
```

Which instances have node "fred" as their primary?

```
gnt-instance list --no-header -o name -F ' pnode == "fred" '
```

More examples at:

<http://everythingsysadmin.com/2013/02/ganeti-list-filters.html>

query filters

Which nodes aren't "mynode"?

```
gnt-node list --filter "not(name == 'myhost.google.com')"
```

Which instances are using more than 3 virtual CPUs?

```
gnt-instance list -F 'oper_vcpus > 3'
```

Which instances have node "fred" as their primary?

```
gnt-instance list --no-header -o name -F ' pnode == "fred" '
```

More examples at:

<http://everythingsysadmin.com/2013/02/ganeti-list-filters.html>

**copying examples from blogs?
your software must be popular**



what fields can I query?

```
# gnt-node list-fields
```

Name	Type	Title	Description
ctime	Timestamp	CTime	Creation timestamp
ctotal	Number	CTotal	Number of logical processors
custom_ndparams	Custom	CustomNodeParameters	Custom node parameters
dfree	Storage size	DFree	Available storage space
disk_state	Custom	DiskState	Disk state
drained	Boolean	Drained	Whether node is drained
group	Text	Group	Node group
group.uuid	Text	GroupUUID	UUID of node group
hv_state	Custom	HypervisorState	Hypervisor state

```
# man gnt-node
```

syntax

man 7 ganeti

```
gnt-instance list --filter 'be/memory > 127'
```

operator

field

value

fields are fixed

operators can be

==	<	<=	=*	=~	and	in
!=	>	>=	!*	!~	or	not

values can be

127	(number)	"host1"	(string)	m/.*google.com/	(regex)
-----	----------	---------	----------	-----------------	---------

Some filters have no values: `gnt-node list --filter 'master_candidate and not master'`

RAPI

All query filters have REST Http API equivalents.

```
curl -G --insecure  
https://user:pw@localhost:5080/2/query/instance/ fields
```

```
curl -G --insecure  
https://user:pw@localhost:5080 /2/query/instance  
--data-urlencode 'fields=name,uuid'  
--data-urlencode 'filter=["=*", "name", "*.google.com"]'
```

```
Output: {"fields": [{"doc": "Instance name", "kind": "text", "name": "name", "title": "Instance"},  
{"doc": "Instance UUID", "kind": "text", "name": "uuid", "title": "UUID"}],  
"data": [[[0, "host1.google.com"], [0, "c1305d40-692b-11e4-9803-0800200c9a66"]]]  
}
```

task for Ganeti users

Please `replace custom filtering workarounds
by query filters.`

things to watch out for

fields are case-sensitive

`gnt-job list --filter 'not(...)'` always returns empty output

- Issue [958](#), fixed in Ganeti 2.13

make sure your shell doesn't understand '>' as output redirection

- quoting helps

job filters

Commands like

spawn me 10 Debian images

live-migrate those instances to that node

are jobs and live in a scheduler queue (`gnt-job list`).

Ganeti 2.13: *filter rules* that can be matched
against each job and execute an action

{ ACCEPT
PAUSE
REJECT
CONTINUE
RATE_LIMIT n

job filters

Drain the queue:

```
gnt-filter add '--predicates=[["jobid", [ ">", "id", "watermark" ] ] ]'  
--action=REJECT
```

Pause all new jobs not belonging to a specific maintenance:

```
gnt-filter add --priority=0 --action=ACCEPT  
  '--predicates=[["reason", [ "=~", "reason", "maintenance" ] ] ]'  
gnt-filter add --priority=1 --action=PAUSE  
  '--predicates=[["jobid", [ ">", "id", "watermark" ] ] ]'
```

Limit the number of simultaneous instance disk replacements to 10 in order to throttle replication traffic:

```
gnt-filter add '--action=RATE_LIMIT 10'  
  '--predicates=[["opcode", [ "=", "OP_ID", "OP_INSTANCE_REPLACE_DISKS" ] ] ]'
```

```
gnt-filter add list
--predicates=[["reason", ["=~", "reason", "maintenance"]], ...]
```

predicate

predicate name
defines behaviour

ganeti filter expression
same language as in query filters

The **predicate name** defines

- what job-related thing the predicate works on (**opcode**, **jobid**, **reason**)
- what **fields** are available and what they access
- whether special **values** are available (like "watermark" for **jobid**)

examples

Predicate

`opcode`

`jobid`

`reason`



extensible.

you can ask us to add more for you
with custom `fields` and `values` to solve your task

Fields

`all names` in the
JSON representation
of the opcode

`id` only

`source`

`reason` only

`timestamp`

Values

`"strings", 14, 15`

`14` or `"watermark"`



treated as highest job id
at creation time

`"strings"`

`"strings"`

`1415664678`

RAPI

Job filters can be queried/added/deleted with the REST Http API.

```
curl -X POST -H "Content-Type: application/json" --insecure
https://user:pw@localhost:5080 /2/filters
--data '{"priority": 0,
        "predicates": [{"jobid", [ ">", "id", "watermark" ]}],
        "action": "REJECT"}
```

```
curl -X GET/DELETE -H "Content-Type: application/json" --insecure
https://user:pw@localhost:5080 /2/filters/filter_uuid
```


task for Ganeti users

Please `replace custom drain logic /
out-of-ganeti job control
by job filters if possible.`

more info

Filter language: [man 7 ganeti](#)

Job filters: [man gnt-filter](#)

Job filter
behaviour examples: [qa_filters.py](#)

questions

Please ask them.

bonus: ad-hoc reason rate limiting

Use `'rate-limit:N:label'` reason for rate limiting


```
gnt-node evacuate --reason='rate-limit:7:mydrain' node1
```



**not more than this many jobs
with this reason** will run at any given time

bonus: filter language in Haskell

```
data Filter field
= EmptyFilter           -- No filter at all
| AndFilter    [ Filter field ] -- & [expression, ...]
| OrFilter     [ Filter field ] -- | [expression, ...]
| NotFilter    (Filter field)   -- ! expression
| TrueFilter   field           -- ?
| EQFilter     field FilterValue -- = !=
| LTFilter     field FilterValue -- <
| GTFilter     field FilterValue -- >
| LEFilter     field FilterValue -- <=
| GEFilter     field FilterValue -- >=
| RegexpFilter field FilterRegex -- =~
| ContainsFilter field FilterValue -- =[]

-- Ways we can compare things.
data Comparison = Eq | Lt | Le | Gt | Ge

== < <=  =*  =~  and  in
!= > >=  !*  !~  or  not

-- Operations in the leaves of the Ganeti filter
language.
data FilterOp field val where
  Truth    :: FilterOp field ()
  Comp     :: Comparison -> FilterOp field FilterValue
  Regex    :: FilterOp field FilterRegex
  Contains :: FilterOp field FilterValue
```

bonus: filter language in Haskell

```
-- Checks if a filter matches.
--
-- field: accessors like "name", "uuid", or "a.b.c"
-- val:   values to be matched, like "*.google.com" or 12
```

```
evaluateFilterM :: Monad m => (FilterOp field val -> field -> val -> m Bool)
  -> Filter field
  -> m Bool
```

opFun
function that decides
whether / how a leaf matches

the filter to evaluate
result

```
evaluateFilterM opFun fil = case fil of
  EmptyFilter      -> return True
  AndFilter flts   -> allM recurse flts
  OrFilter flts    -> anyM  recurse flts
  NotFilter flt    -> not <$> recurse flt
  TrueFilter field -> opFun Truth   field ()
  EQFilter field val -> opFun (Comp Eq) field val
  LTFilter field val -> opFun (Comp Lt) field val
  LEFilter field val -> opFun (Comp Le) field val
  GTFilter field val -> opFun (Comp Gt) field val
  GEFilter field val -> opFun (Comp Ge) field val
  RegexpFilter field re -> opFun Regex   field re
  ContainsFilter field val -> opFun Contains field val
where
  recurse = evaluateFilterM opFun
```

e.g. **predicate** → **field** / **value** meaning

as in **field**

or **a.b.c** JSON lookup as in **opcode**

can be pure or do some IO to fetch your result

specialises to both query filters and job filtering, and more if you like.