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=[ ...]
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
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
```
what fields can I query?

# gnt-node list-fields

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ctime</td>
<td>Timestamp</td>
<td>CTime</td>
<td>Creation timestamp</td>
</tr>
<tr>
<td>ctotal</td>
<td>Number</td>
<td>CTotal</td>
<td>Number of logical processors</td>
</tr>
<tr>
<td>custom_ndparams</td>
<td>Custom</td>
<td>CustomNodeParameters</td>
<td>Custom node parameters</td>
</tr>
<tr>
<td>dfree</td>
<td>Storage size</td>
<td>DFree</td>
<td>Available storage space</td>
</tr>
<tr>
<td>disk_state</td>
<td>Custom</td>
<td>DiskState</td>
<td>Disk state</td>
</tr>
<tr>
<td>drained</td>
<td>Boolean</td>
<td>Drained</td>
<td>Whether node is drained</td>
</tr>
<tr>
<td>group</td>
<td>Text</td>
<td>Group</td>
<td>Node group</td>
</tr>
<tr>
<td>group.uuid</td>
<td>Text</td>
<td>GroupUUID</td>
<td>UUID of node group</td>
</tr>
<tr>
<td>hv_state</td>
<td>Custom</td>
<td>HypervisorState</td>
<td>Hypervisor state</td>
</tr>
</tbody>
</table>
fields are fixed

operators can be

\[
\begin{align*}
&== \quad < \quad <= \quad *= \quad =~ \quad \text{and} \quad \text{in} \\
&!= \quad > \quad >= \quad !* \quad !~ \quad \text{or} \quad \text{not}
\end{align*}
\]

values can be

\[
\begin{align*}
&127 \quad \text{(number)} \quad \text{"host1"} \quad \text{(string)} \quad \text{m/.*google.com/} \quad \text{(regex)}
\end{align*}
\]

Some filters have no values:

\[
gnt\text{-node list --filter 'master_candidate and not master'}
\]
All query filters have REST Http API equivalents.

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

```bash
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](https://example.com/issue958), fixed in Ganeti 2.13

make sure your shell doesn't understand '>' as output redirection
- quoting helps
Commands like

\textit{spawn me 10 Debian images}

\textit{live-migrate those instances to that node}

are jobs and live in a scheduler queue (\texttt{gnt-job list}).

Ganeti 2.13: \textit{filter rules} that can be matched against each job and execute an action

\{\texttt{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"]]]
```
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)
<table>
<thead>
<tr>
<th>Predicate</th>
<th>Fields</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>all names in the JSON representation of the opcode</td>
<td>“strings”, 14, 15</td>
</tr>
<tr>
<td>jobid gives access to</td>
<td>id only</td>
<td>14 or “watermark”</td>
</tr>
<tr>
<td>reason extensible.</td>
<td>source only</td>
<td>“strings”</td>
</tr>
<tr>
<td></td>
<td>reason only</td>
<td>“strings”</td>
</tr>
<tr>
<td></td>
<td>timestamp</td>
<td>1415664678</td>
</tr>
</tbody>
</table>
Job filters can be queried/added/deleted with the REST Http API.

```bash
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.
Filter language: man 7 ganeti

Job filters: man gnt-filter

Job filter
behaviour examples: qa_filters.py
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

http://docs.ganeti.org/ganeti/master/html/design-optables.html
bonnus: filter language in Haskell

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

    -- 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
-- 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

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
  LTEFilter field val -> opFun (Comp Le) field val
  GTFilter field val -> opFun (Comp Gt) field val
  GTEFilter 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

opFun function that decides whether / how a leaf matches the filter to evaluate result
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
specilises to both query filters and job filtering, and more if you like.