



Überwachung von MySQL-Datenbanken mit check_mysql_health

Gerhard Laußer
ConSol* Software GmbH, München

07.09.08

www.consol.de



Wer bin ich?

Gerhard Laußer
aus München
arbeite bei der Firma ConSol*

„lausser“ im Nagios-Portal

07.09.08

www.consol.de

- › Kundenanforderung mit etlichen Punkten.
- › Es gab ein paar MySQL-Plugins.
- › Die deckten nur wenige der Anforderungen ab.
- › Die unterschieden sich stark bei den Aufrufparametern.
- › Daraus entstand `check_mysql_perf` (in C geschrieben)

- › Anfang 2009 umgeschrieben in Perl als `check_mysql_health`

- › Ein Plugin, das viele Parameter prüft
- › mit leicht erweiterbarer Struktur

<http://www.consol.de/opensource/nagios/check-mysql-health>

```
tar xzvf check_mysql_health-2.0.4.tar.gz
```

```
cd check_mysql_health-2.0.4
./configure --help
```

```
.....
--libexecdir=DIR          program executables [PREFIX/libexec]
.....
--with-nagios-user=USER   set user name to run nagios
--with-nagios-group=GROUP set group name to run nagios
--with-statefiles-dir=PATH sets directory for the state files
                          (default=/var/tmp/check_mysql_health)
--with-perl=PATH          sets path to perl executable
                          (default= perl im $PATH)
.....
```

Ein funktionierendes DBD::mysql oder zumindest eine funktionierende MySQL-Client-Installation (mit dem Kommando mysql) muss natürlich auch vorhanden sein.

Grundsätzliche Kommandozeilenparameter, die zum Kontaktieren der Datenbank nötig sind:

1a. Lokaler Aufruf auf dem DB-Server

```
$ check_mysql_health --hostname localhost
$ check_mysql_health --hostname localhost --socket <nicht-standard Socket>
```

1b. Remote-Aufruf vom Nagios-Server aus

```
$ check_mysql_health --hostname <hostname oder ip>
$ check_mysql_health --hostname <hostname oder ip> --port <port ungleich 3306>
```

2. Authentifizierung

```
$ check_mysql_health ... --username <user> --password <pass>
```

3. Verbindungsmethode

```
$ check_mysql_health ... --method dbi (Default)
$ check_mysql_health ... --method mysql
$ check_mysql_health ... --method sqlrelay
```

Anstelle dieser Kommandozeilenparameter kann man die Connect/Login-Daten auch per Environmentvariablen übergeben.

- `$NAGIOS__SERVICEMYSQL_HOST` entspricht `--hostname`.
- `$NAGIOS__SERVICEMYSQL_PORT` entspricht `--port`.
- `$NAGIOS__SERVICEMYSQL_SOCKET` entspricht `--socket`.
- `$NAGIOS__SERVICEMYSQL_USER` entspricht `--username`.
- `$NAGIOS__SERVICEMYSQL_PASS` entspricht `--password`.
- `$NAGIOS__SERVICEMYSQL_METH` entspricht `--method` (s.u.).

Custom Macros !

```
define service {
  register          0
  name              app_mysql_NPXTST
  host_name         mydb3
  servicegroups    mysql_NPXTST
  _mysql_host      mydb3.naprax.de
  _mysql_user      nagios
  _mysql_pass      nagios
}
```

Jeder Service, der zu einer bestimmten MySQL-Datenbank gehört (diese heisst z.B. NPXTST), benutzt dieses Template.

```
define service {
  service_description app_mysql_default_NPXTST_check_login
  use                  app_mysql_default_NPXTST
  check_command        check_mysql_health!connection-time
}
```

Dadurch stehen beim Plugin-Aufruf die Environmentvariablen zur Verfügung.

Idealerweise richtet man einen eigenen Benutzer für Monitoringzwecke ein:

```
GRANT USAGE ON *.* to 'nagios'@'nagsrv.naprax.de' IDENTIFIED BY 'nagios';  
FLUSH PRIVILEGES;
```

Was genau das Plugin messen/bewerten soll, gibt man mit dem Kommandozeilenparameter --mode an:

--mode connection-time

--mode qcache-hitrate

...

Ab hier wird davon ausgegangen, dass die Environmentparameter gesetzt wurden.

• Ist ein Login möglich?

```
nagrsrv$ check_mysql_health --mode connection-time
OK - 0.009 seconds to connect as nagios|connection_time=0.01;2;10
```

Eignet sich für eine Parentbeziehung, denn ohne Login funktioniert der Rest auch nicht.

• Wie lange bzw. wie kurz läuft die Datenbank schon?

```
nagrsrv$ check_mysql_health --mode uptime --critical 10:
CRITICAL - database up since 6 minutes|uptime=6;10;;10:
```

Damit erkennt man DB-Crashes mit sekundenschnellem Wiederanlauf.

Wenn man Thresholds andersrum setzt, kann man auch alarmiert werden, wenn die DB schon ewig läuft und ihr ein Reboot guttut.

Es gibt eine Menge Tipps&Tricks, die aber z.T. widersprüchlich sind. Einen allgemeingültigen Performance-Leitfaden habe ich nicht gefunden. Die Default-Schwellwerte sind daher nicht in jedem Fall der Weisheit letzter Schluss. Das Einsatzszenario entscheidet.

• Index Usage

```
nagrsrv$ check_mysql_health --mode index-usage
OK - 93.08% of all reads use indices|index_usage=93.08%;90;;80:
```

Ein schlechter Wert kann auf schlechte Programmierung hindeuten, ist aber manchmal unvermeidlich. Der Entwickler entscheidet das. In so einem Fall checkt man, ob sich der Index verschlechtert.

• Query-Cache Hitrate

```
nagrsrv$ check_mysql_health --mode qcache-hitrate
OK - query cache hitrate at 96.12%| qcache_hitrate=96.12%;90;;80: qcache_hitrate_now=82.39%
selects_per_sec=130.72
```

Der erste Wert umfasst die gesamte Laufzeit der Datenbank

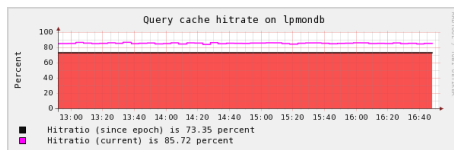
qcache_hitrate=96.12%;90::;80:

Er glättet Schwankungen und zeigt die langfristige Qualität der Applikation

Der zweite Wert umfasst das check_interval

qcache_hitrate_now=82.39%

Damit lassen sich kurzfristige Verschlechterungen feststellen (z.B. zu Spitzenzeiten, beim Backup,...)



Der dritte Wert "qualifiziert" den zweiten Wert

selects_per_sec=130.72

Je weniger Selects, desto weniger aussagekräftig ist die momentane Hitrate

MyISAM Keycache

Bereich im Hauptspeicher, in dem Indexblöcke gehalten werden. (Datenblöcke werden im Filesystem-Cache gehalten)

nagsrv\$ check_mysql_health --mode myisam-keycache-hitrates

CRITICAL - myisam key cache hitrate at 90.00% |

keycache_hitrates=99.97%;99::;95: keycache_hitrates_now=100.00%;99::;95:

InnoDB Buffer Pool

Die InnoDB-Engine verwendet Buffer für Daten und Indices, um Festplattenzugriffe zu minimieren. Eine hohe Hit-Rate bedeutet hier wenige Plattenzugriffe.

nagsrv\$ check_mysql_health --mode innodb-bufferpool-hitrates

CRITICAL - innodb buffer pool hitrate at 100.00% | bufferpool_hitrates=100.00%;99::;95::;0:100

bufferpool_hitrates_now=100.00%;99::;95::;0:100

Table-Cache

Das Öffnen von Tabellen verbraucht Ressourcen, deshalb werden einmal geöffnete Tabellen im Table-Cache gehalten. Ist dieser Cache zu klein, dann werden bereits geöffnete Tabellen wieder geschlossen, nur um weitere Zugriffe auf Tabellen abarbeiten zu können.

nagsrv\$ check_mysql_health --mode tablecache-hitrates

OK - table cache hitrate 86.54%, 16.17% filled | tablecache_hitrates=86.54%;99::;95: tablecache_fillrate=16.17%

Temporäre Tabellen

Bei Sortieroperationen (SELECT ... ORDER BY, GROUP BY) legt MySQL intern temporäre Tabellen an. Wenn der Hauptspeicher dafür nicht ausreicht, werden sie ins Filesystem ausgelagert. (laaaangsam)

```
nagsrv$ check_mysql_health --mode tmp-disk-tables
OK - 24.80% of 76 tables were created on disk | pct_tmp_table_on_disk=24.80%;25;50
pct_tmp_table_on_disk_now=10.53%
```

Slow Queries

SELECTs, die besonders lange dauern (Parameter `long_query_time=5`), sollten selten bzw. nie vorkommen. Bei Auffälligkeiten sollte man Logging mit `log_slow_queries` einschalten.

```
nagsrv$ check_mysql_health --mode slow-queries --warning 0.01 --critical 0.1
WARNING - 13 slow queries in 316 seconds (0.0411/sec) | slow_queries_rate=0.0411;1;10
```

Long running processes

Momentane Anzahl der DB-Prozesse, die bereits länger als eine Minute laufen. Wenn so etwas häufig vorkommen, mit `SHOW PROCESSLIST (< 5.1)` oder `information_schema.processlist` kontrollieren.

```
nagsrv$ check_mysql_health --mode long-running-procs
WARNING - 2 long running processes|long_running_procs=2;1;10
```

Connection Threads

Anzahl gleichzeitig verbundener User-Sessions. Liefert Hinweise auf Unterdimensionierung, Lastspitzen und aussergewöhnliche Zustände. Ist für Langzeitbeobachtung interessant. Connection-Pooling verwenden!

```
nagrsrv$ check_mysql_health --mode threads-connected  
OK - 8 client connection threads | threads_connected=8;10;20
```

Master

- startet einen Thread, der datenverändernde Statements ins Binlog schreibt.

Slave

- startet einen Thread, der Binlog-Daten vom Master holt. (IO-Thread)
- startet einen Thread, der die Statements aus dem Binlog ausführt. (SQL-Thread)

```
nagrsrv$ check_mysql_health --mode slave-lag  
WARNING - slave is 12sec behind master | slave_lag=12;10;20
```

```
nagrsrv$ check_mysql_health --mode slave-io-running  
OK - slave io is running | slave_io_running=1
```

```
nagrsrv$ check_mysql_health --mode slave-sql-running  
OK - slave sql is running | slave_sql_running=1
```

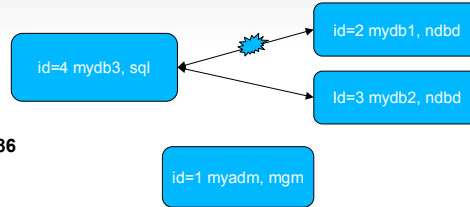
Wer den Angaben von SHOW SLAVE STATUS nicht traut, kann `check_repl_mysql_heartbeat.pl` von Oli Sennhauser verwenden. Es basiert auf einer kleinen Tabelle, in die der Master die aktuelle Uhrzeit schreibt. Auf dem Slave wird diese gelesen und sollte möglichst nahe bei der Systemzeit liegen.

```

ndb_mgm> show
Connected to Management Server at: 192.168.100.4:1186
Cluster Configuration
-----
[ndbd(NDB)] 2 node(s)
id=2 (not connected, accepting connect from 192.168.100.1)
id=3 @192.168.100.2 (mysql-5.1.27 ndb-6.3.17, Nodegroup: 0, Master)

[ndb_mgmd(MGM)] 1 node(s)
id=1 @192.168.100.4 (mysql-5.1.27 ndb-6.3.17)

[mysqld(API)] 1 node(s)
id=4 @192.168.100.3 (mysql-5.1.27 ndb-6.3.17)
    
```



```

myadm$ check_mysql_health --mode cluster-ndbd-running
CRITICAL - ndb node 2 is not connected | ndbd_nodes=1 ndb_mgmd_nodes=1 mysqld_nodes=1
    
```

In diesem Modus greift `check_mysql_health` auf das Kommando `ndb_mgm` zurück. Normalerweise gibt es dieses nur auf dem Managementknoten des MySQL-Clusters. Man kann allerdings auch auf dem Nagios-Server das entsprechende Paket installieren, z.B. `MySQL-Cluster-gpl-tools`

In dem Fall muss man noch den Hostnamen des Managementknotens angeben.

```

nagrsrv$ check_mysql_health --mode cluster-ndbd-running --hostname myadm
OK - all ndb nodes are connected | ndbd_nodes=2 ndb_mgmd_nodes=1 mysqld_nodes=1
    
```

Mit `--mode sql` kann man eigene SQL-Statements ausführen lassen. Diese müssen eine Zahl zurückliefern, die mit den Thresholds verglichen werden kann.

Beispiel: Anzahl der Zeilen in einer Tabelle QUEUE

```
$ check_mysql_health --mode sql --name 'SELECT COUNT(*) FROM QUEUE' \  
  --warning 100 --critical 300 --name2 queuelen  
OK - queuelen: 55 | queuelen=55
```

`--name2` wird für die Ausgabe und die Performancedaten verwendet. Ohne diesen Parameter erscheint sonst das SQL-Statement.

`--units` kann man verwenden, um in Ausgabe und Performancedaten eine Einheit anzugeben.

Vorsicht bei Sonderzeichen `*` & etc im Statement, `check_by_ssh` ist da sehr empfindlich.

Am Besten man encodiert SQL-Statements, dann braucht man sich um Sonderzeichen nicht zu kümmern.

```
check_mysql_health --mode encode <<EOSQL  
> SELECT COUNT(*) FROM QUEUE  
> EOSQL  
SELECT%20COUNT%28%2A%29%20FROM%20QUEUE
```

```
$ check_mysql_health --mode sql --name SELECT%20COUNT%28%2A%29%20FROM%20QUEUE
```

Eigene Erweiterungen packt man in eine Datei namens CheckMySQLHealth1.pm (1..n)

```

package MyQueue;

sub init {
    # Gewinnen von Informationen
    my $self = shift;
    my %params = @_;
    if ($params{mode} =~ /my::queue::length/) {
        $self->(length) = $self->(handle)->fetchrow_array(q{
            SELECT COUNT(*) FROM queues
        });
    }
}

sub nagios {
    # Auswerten von Informationen
    my $self = shift;
    my %params = @_;
    if ($params{mode} =~ /my::queue::length/) {
        $self->add_nagios(
            $self->check_thresholds($self->(length), 100, 500),
            sprintf "queue length is %d", $self->(length));
        $self->add_perfdata(sprintf "queuelen=%d;%d;%d",
            $self->(length), $self->(warningrange),
            $self->(criticalrange));
    }
}

$ check_mysql_health --mode my-queue-length
OK - queue length is 12 | queuelen=12;100;500
    
```

--mode my-queue-length

1. my-queue bedeutet, dass package MyQueue in einer der Erweiterungsdateien CheckMySQLHealth[1..n].pm adressiert wird.
2. my-queue-length wird intern in my::queue::length umgewandelt und steht in \$params{mode} zur Verfügung.

Wie wird CheckMySQLHealth1.pm in check_mysql_health integriert?

1. Statische Methode

Beim "Compilieren" des Plugins legt man die Erweiterungsdateien in ein Verzeichnis /tmp/xy

```
./configure --with-mymodules-dir=/tmp/xy
```

CheckMySQLHealth1.pm ist dann in check_mysql_health enthalten

2. Dynamische Methode

Die Erweiterungsdateien werden erst zur Laufzeit von check_mysql_health geladen.

```
./configure --with-mymodules-dyn-dir=/usr/local/nagios/etc/extensions
```

Vorteil: man kann "basteln"

Nachteil: man muss die Erweiterungen evt. auf alle DB-Server kopieren.

SQLRelay ist ein Datenbankproxy, der die gängigsten DB-Produkte unterstützt.

Er etabliert eine Verbindung zum Server mydb3 und lässt sie die ganze Zeit offen. Clients verbinden sich dann mit DBD::SQLRelay zum Proxy. Bei vielen Services bedeutet das kein ständiges Rein und Raus auf der Datenbank.

```
<instance id="mydb3" port="10000" socket="" dbase="mysql" connections="3" maxconnections="5" maxqueuelength="0"
growby="1" ttl="60" endofsession="commit" sessiontimeout="600" runasuser="nobody" runasgroup="nobody" cursors="5"
authtier="listener" handoff="pass">
  <users>
    <user user="nagios" password="nagios"/>
  </users>
  <connections>
    <connection connectionid="mydb31"
string="host=mydb3;user=nagios;password=nagios;db=information_schema" metric="1"/>
  </connections>
</instance>
```






```
check_mysql_health --hostname 127.0.0.1 --port 10000 --method sqlrelay
```

```
define command {
  command_name    check_mysql_health
  command_line    $USER3$/check_mysql_health \
  --hostname $ _SERVICEMYSQL_HOST$ \
  --username $ _SERVICEMYSQL_USER$ \
  --password $ _SERVICEMYSQL_PASS$ \
  --mode $ARG1$ \
  $ARG2$
}

define service {
  register        0
  name            app_mysql_NPXTST
  host_name       mydb3
  servicegroups   mysql_NPXTST
  _mysql_host     mydb3.naprax.de
  _mysql_user     nagios
  _mysql_pass     nagios
}

define service {
  service_description app_mysql_default_NPXTST_check_login
  use                app_mysql_default_NPXTST
  check_command      check_mysql_health!connection-time
}

define service {
  service_description app_mysql_default_NPXTST_check_uptime
  use                app_mysql_default_NPXTST
  check_command      check_mysql_health!uptime!--warning 10 --critical 100
}
```

Service 	Status 	Last Check 	Duration 	Attempt 	Status Information
app_mysql_default_NDO_check_login	OK	06-21-2009 16:48:22	0d 22h 42m 9s	1/4	OK - 0.46 seconds to connect as ndo
app_mysql_default_NDO_check_uptime	OK	06-21-2009 16:48:22	0d 22h 41m 54s	1/4	OK - database is up since 109514 minutes
app_mysql_perf_NDO_check_index_usage	CRITICAL	06-21-2009 16:48:22	107d 0h 49m 14s	4/4	CRITICAL - index usage 13.74%
app_mysql_perf_NDO_check_inno_bpool_hits	OK	06-21-2009 16:48:22	0d 22h 41m 39s	1/4	OK - innodb buffer pool hitrate at 100.00%
app_mysql_perf_NDO_check_inno_bpool_waits	OK	06-21-2009 16:48:22	0d 22h 41m 54s	1/4	OK - 0 innodb buffer pool waits in 299 seconds (0.0000/sec)
app_mysql_perf_NDO_check_inno_log_waits	OK	06-21-2009 16:48:20	0d 22h 41m 54s	1/4	OK - 0 innodb log waits in 300 seconds (0.0000/sec)
app_mysql_perf_NDO_check_longrunners	OK	06-21-2009 16:48:21	0d 22h 41m 29s	1/4	OK - 0 long running processes
app_mysql_perf_NDO_check_mysql_keycache_hits	OK	06-21-2009 16:48:20	0d 22h 41m 54s	1/4	OK - mysam keycache hitrate at 100.00%
app_mysql_perf_NDO_check_query_cache_hits	CRITICAL	06-21-2009 16:48:21	106d 17h 39m 17s	4/4	CRITICAL - query cache hitrate 73.35%
app_mysql_perf_NDO_check_query_cache_prunes	OK	06-21-2009 16:48:20	61d 4h 8m 12s	1/4	OK - 0 query cache lowmem prunes in 299 seconds (0.00/sec)
app_mysql_perf_NDO_check_slow_queries	OK	06-21-2009 16:48:21	0d 22h 41m 0s	1/4	OK - 0 slow queries in 300 seconds (0.00/sec)
app_mysql_perf_NDO_check_table_lock_cont	CRITICAL	06-21-2009 16:48:20	107d 19h 52m 10s	4/4	CRITICAL - table lock contention 4.89%
app_mysql_perf_NDO_check_table_cache_hits	OK	06-21-2009 16:48:20	0d 22h 41m 25s	1/4	OK - table cache hitrate 71.11%, 16.21% filled
app_mysql_perf_NDO_check_temp_tables	WARNING	06-21-2009 16:48:20	0d 22h 41m 1s	4/4	WARNING - 48.83% of 73 tables were created on disk
app_mysql_perf_NDO_check_threadcache_hits	OK	06-21-2009 16:48:19	61d 4h 8m 12s	1/4	OK - thread cache hitrate 99.99%
app_mysql_perf_NDO_check_threads_conn	OK	06-21-2009 16:48:20	0d 22h 41m 54s	1/4	OK - 17 client connection threads

Das Plugin `check_mysql_health` ist die Nr.2 einer Familie von Datenbank-Plugins. Daneben gibt es `check_oracle_health`, `check_mssql_health` und `check_db2_health`.

Allen gemeinsam sind:
--mode connection-time
--mode sql

Die Erweiterungen mit `Check***1.pm`
--method sqrelay