ParaStation V5

ParaStation HEALTHCHECKER

—

Administrator's Guide

Release 1.0.1-2
Published September 2010

ParaStation HEALTHCHECKER

ParaStation Healthchecker Administrator's Guide

Release 1.0.1-2

September 2010

Printed 10 November 2010, 15:38

ParTec Cluster Competence Center GmbH
Possartstr. 20
D-81679 München
Phone +49-89-99809-0
Fax +49-89-99809-555

http://www.par-tec.com
<info@par-tec.com>

This document provides detailed information about the *ParaStation Healthchecker*. Installation and configuration of the *ParaStation Healthchecker* as well as usage of the *ParaStation Healthchecker* commands are explained in-depth.

Though it may seem hard to believe, this manual might contain errors. We welcome any reports on errors or problems that are found. We also would appreciate suggestions on improving this book. Please direct all comments and problems to <support@par-tec.com>.

The most up-to-date version of this document is available at http://docs.par-tec.com.

> Share your knowledge with others. It's a way to achieve immortality.
>
> —Dalai Lama

Table of Contents

1. Preface 1
1.1. About this book 1
1.2. This book's audience 1
1.3. *Healthchecker* overview 1
2. *Healthchecker* description 3
2.1. Introduction 3
2.2. Terms 3
2.3. Framework 3
3. Installing the *Healthchecker* 5
3.1. Prerequisites 5
3.2. Installing the software package 5
4. Configuring the *Healthchecker* 7
4.1. General configuration 7
4.2. Configuring the test 9
4.3. Test set configuration 10
4.4. Alternate test configuration 10
4.5. Configuring actions 11
5. Running the *Healthchecker* 13
I. Reference Pages 15
healthcheck.conf 17
pshealthcheck 19
pshcgetconf 23
A. List of implemented checks 25
B. Extending the *Healthchecker* 27
B.1. Adding new tests 27
B.2. Adding new actions 27
B.3. Adding new test sets 27
C. How to determine a node's class 29
D. Sample action script 31
E. Including the *Healthchecker* in a resource management system 33
E.1. Using the *Healthchecker* within a job's prologue 33
E.2. Using the *Healthchecker* within a job's epilogue 34
Glossary 37

List of Figures

3.1. Installing package 5
4.1. Example `healthcheck.conf` file 8
4.2. Example `tests.conf` file 9
4.3. Example `testset.conf` file 10
4.4. Example test script 11
5.1. Example **pshealthcheck** output 13
5.2. Example verbose **pshealthcheck** output 13
5.3. Example **pshcgetconf** output 14
5.4. Example **pshcgetconf** `-l` output 14
C.1. **psconfig** calling example 29
D.1. Example action script output 31
D.2. Sample **pbsnodes** output 31
D.3. Example test script 32
E.1. Sample prologue file 34
E.2. Sample epilogue file 35

Chapter 1. Preface

1.1. About this book

This book discusses installation, configuration and usage of the *ParaStation Healthchecker*. In addition, all the *ParaStation Healthchecker* related utilities are described. Furthermore, this book also discusses all aspects of administring and maintaining a compute cluster using the *ParaStation Healthchecker*.

The information referenced in this book refers to version 1.0 of the *ParaStation Healthchecker*.

The most up-to-date version of this document is available at http://docs.par-tec.com.

1.2. This book's audience

The *ParaStation Healthchecker* and this book is targeted to Linux system and cluster administrators. The administrators should be familiar with the concepts of general Linux system installation, Linux server administration, HPC clustering, networking, remote boot of nodes and MPI environments.

As this document describes cluster infrastructure maintenance tasks, it is not intended for end users.

1.3. *Healthchecker* overview

The *ParaStation Healthchecker* offers tools and methods to easely verify a running cluster node against a pre-defined configuration. To do so, a pre-defined set of particular tests is run on the node to evaluate node-specific values and compare them against a set of pre-defined parameters.

If a node does not match those configuration, configurable actions may be performed to minimize the impact of this failed node on the entire cluster system. Those actions may be as easy as sending an email to the administrator, shutting down a node or may be as complex as setting a node off-line within the resource management system, opening a new trouble ticket, and forwarding this ticket to an external help desk. As a consequence, the node may be logically and even physically removed from the cluster without any intervention from the administrator. This enables unattended operation of the cluster system.

The *ParaStation Healthchecker* supports all major Linux distributions and system configurations. It is designed to be easily adjustable to the different needs. New checks may be added, existing checks may be omitted.

Chapter 2. *Healthchecker* description

2.1. Introduction

The *ParaStation Healthchecker* as part of the *ParaStationV5* cluster suite is an elaborated test suite to ensure the usability of compute and services nodes and the system in general of a compute cluster. It includes a framework to define and run a set of pre-defined tests. Each particular test analyzes a particular function on the node or within the system. It returns a single state, which might be OK, WARNING or FAILED. Each test is optimized to very fast analyze in a non-destructive way an aspect of the system. Tests may use all available information. This include in particular tools for reading the state of RAID controllers, reading operating system parameters, IPMI or SNMP queries, and more.

2.2. Terms

The following terms are used within this document to describe the different parts of the *Healthchecker*:

`Test`
: An execution unit including all needed configuration that returns one of the exit states 0 (SUCCESS), 1 (WARNING), or 2 (ERROR). In case of a warning or error a text string describing the problem can be returned.

`Check`
: A reusable execution unit doing a single kind of check.

  Checks are typically implemented using shell scripts or calling a system command like **grep**.

`Test set`
: A set of tests executed together (depending on connected classes) and returning the highest exit state of all included tests.

`Class`
: Each test can be connected to one or more classes. The test is only executed on nodes that belong to one of its connected classes.

2.3. Framework

The *Healthchecker* framework consists of a set of configuration files defining the particular tests. The command **pshcgetconf** analyzes this configuration and the node's role and returns all information suitable for the local node. This command is implicitly called by **pshealthcheck**, which performs the actual checks.

New tests may be added by extending an existing configuration file or by adding an additional configuration file. These tests may use existing checks with adapted parameters or make use of new checks. Each check within a test may be monitored using a timeout parameter. Therefore, the check itself does not have to deal with hanging commands. In case of hanging commands (timeout), they will be killed by the framework itself.

checks may be added by creating additional commands within the checks directory (see Section 4.2, “Configuring the test” for details). Every check has to return 0, 1 or 2, depending on the result.

Every test may be associated with a list of nodes or node classes. Refer to Appendix C, *How to determine a node's class* on how to determine a node's class. This allows to use an identical configuration on all nodes, as the tests actually run on a node are controlled by the node's name or class. A consistent configuration across all nodes simplifies configuration considerably.

Each test has to be a member of at least one test set. For each test set, action scripts may be defined within the test set's action subdirectory. All action scripts are called after running all tests for a test set, regardless of the result. The actual result is provided within the variable *TS_RESULT*.

Refer to Appendix B, *Extending the Healthchecker*, for details how to add a new test.

Chapter 3. Installing the *Healthchecker*

3.1. Prerequisites

The *ParaStation Healthchecker* currently supports the following distributions:

- SuSE Linux Enterprise Server 11 (x86_64)
- Redhat Enterprise Linux 5 (x86_64)

The software is provided as RPM package, therefore only RPM-based distributions are supported.

3.2. Installing the software package

As the software is deliverd as a RPM package, it may be installed using **rpm** or any other tools provided by the distribution, e.g. **zypper** or **yum**:

```
# rpm -U pshealthcheck-5.0.0-1.x86_64.rpm
```

Figure 3.1. Installing package

Before running the *Healthchecker*, a valid configuration for the local node is required.

Chapter 4. Configuring the *Healthchecker*

4.1. General configuration

The general configuration for the *Healthchecker* is read from the file `${PSconfDir}/general/healthcheck.conf`, where *`${PSconfDir}`* defaults to `/etc/parastation`. This file defines global paths and parameters.

Figure 4.1, “Example `healthcheck.conf` file”, shows the default configuration file `healthcheck.conf`, as provided by the package. For a standard installation, this file does not have to be modified. Refer also to healthcheck.conf(5).

```
#
# healthcheck.conf
#
# General configuration file for the Parastation healthcheck
#

#
# Directory containing the healthcheck configuration
#
HC_CONF_DIR=${PSconfDir-/etc/parastation}/healthcheck

#
# Path of directories to be searched for test configuration files
#
# Entries in the path have to be colon separated.
#
# This is used by the first way to configure tests for healthcheck
# using
# structured files.
#
HC_TESTCONF_PATH=${HC_CONF_DIR}/testconf.d

#
# Path of directories to be searched for testset configurations
#
# Entries in the path have to be colon separated.
#
# This is used by the second way to configure tests for healthcheck
# using
# init.d like concept with configuration scripts containing a magic
# line.
#
# Each directory in the path should have the following structure:
# ./all/tests/*               # All test configuration scripts
# ./all/actions/*             # All action scripts
# ./${TESTSET}/tests/*        # Symlinks to test scripts to be
# executed in
#                             #  ${TESTSET} in classes given by magic
#                             line.
# ./${TESTSET}/actions/*      # Symlinks to action scripts to be
# executed after
#                             #   testset ${TESTSET} failed.
# ./${TESTSET}/testset.conf   # Configuration file for testset
# ${TESTSET}
#
HC_TESTSET_PATH=${HC_CONF_DIR}/testsets

#
# Syslog facility to use for logging
#
HC_SYSLOG_FACILITY=user
```

Figure 4.1. Example `healthcheck.conf` file

4.2. Configuring the test

An example configuration may be found in the directory `/opt/parastation/share/doc/pshealtcheck`.

The tests are defined using one or more configuration files located in the directory `/etc/parastation/healthcheck/testconf.d`. Typically, there is a file called `tests.conf`, but any other name would be also accepted. All files found in this directory are analyzed.

For each test, a section like

```
[ethernet_eth0]
classes = cs:admin
testsets = reboot:prologue
test = /opt/parastation/lib/checks/ethernet eth0 100
timeout = 10
killwait = 2

[md5sum_parastation_cs]
classes = cs
testsets = reboot
test = /opt/parastation/lib/checks/md5sum
/etc/parastation/classes/cs/md5sums

[md5sum_parastation_admin]
classes = admin
testsets = reboot
test = /opt/parastation/lib/checks/md5sum
/etc/parastation/classes/admin/md5sums
```

Figure 4.2. Example `tests.conf` file

is required. The strings in brackets define a test name and start a new section decribing the named test.

The following list numerates all parameters. They may be given in any order within a test section of the configuration file. *`Parameter`* and value are separated by an equal sign.

`classes` (mandatory)
: A colon separated list of classes this test belongs to.

`testsets` (mandatory)
: A colon separated list of test sets this test belongs to.

`test` (mandatory)
: Command to be executed. Everything right of the equal sign is passed as-is to bash's eval.

`timeout` (default: 0)
: Timeout for the test in seconds. If the *`timeout`* is reached before the test returns, a signal `SIGTERM` is sent to the process and it is handled as if it returned an error.

`killwait` (default: 1)
: Time to wait between `SIGTERM` and `SIGKILL` in case the *`timeout`* of the test is reached.

The ordering of the tests within the configuration file defines the ordering the tests are run afterwards.

Within the previous example, three tests called *`ethernet_eth0`*, *`md5sum_parastation_cs`* and *`md5sum_parastation_admin`* are defined.

The test *ethernet_eth0* is related to the classes *cs* and *admin*. It is part of the test sets *reboot* and *prologue*. The check itself is performed using the command **/opt/parastation/lib/checks/ethernet** using the parameters *eth0* and *100*. For information about the required parameters, take a look at the check itself. The check will be terminated after *timeout* seconds using signal SIGTERM. After *killwait* seconds, it will be killed using SIGKILL.

See also pshealthcheck(1), the section called “CONFIGURATION” and Appendix C, *How to determine a node's class*.

4.3. Test set configuration

Each test set is defined by adding a subdirectory to `${HC_CONF_DIR}/testsets`, which is typically `/etc/parastation/healthcheck/testsets`. The name of the subdirectory is equal to the test set name. Typical test sets are *reboot*, *manual*, *prologue* or *epilogue*. It's useful to name the test set after the role when it's run.

Within the test set's subdirectory, at least the file `testset.conf` is required. An empty `testset.conf` is supported.

The file `testset.conf` may have the following parameters:

TS_TIMEOUT
: Timeout for the entire testset in seconds. Defaults to *0*, which means no timeout.

TS_BREAK
: test set break condition. Default is *never*. Available conditions are *never*, which means always run all the tests of a test set, and *first*, which means break on first error.

Listing Figure 4.3, “Example `testset.conf` file”, shows an example configuration file `testset.conf`:

```
# timeout for the entire testset in seconds
TS_TIMEOUT=240
# break on first failure
# TS_BREAK=first
```

Figure 4.3. Example `testset.conf` file

To add a test to a test set, append the test set name to the *testsets* entry of a particular test, see Section 4.2, “Configuring the test”, for details.

4.4. Alternate test configuration

Beside the previously described way to configure a test, there's an alternative for providing test scripts. A test script is a small script configuring and executing one particular test.

Test scripts are executed after the tests defined in the test configuration files. They are meant for easily creating specialized or temporary tests by just dropping in a file to the correct place.

The script has to be located in the directory `/etc/parastation/healthcheck/testsets/`*testset*`/tests`. The test parameters have to provided as comments, like:

```
#!/bin/bash
#classes cn:admin:node034
#testname Ethernet_Test
#timeout 10
#killwait 2
/opt/parastation/lib/checks/ethernet eth0 100
```

Figure 4.4. Example test script

The parameter *testname* names the test. For a definition of *classes*, *timeout* and *killwait* refer to Section 4.2, “Configuring the test”.

4.5. Configuring actions

action scripts are executed after a test set is run, regardless of the exit state of the test set. They decide what to do by themselves based on the value of the overall test result provided in *$TS_RESULT*. They have to be placed into the `actions` subdirectory of the test set (typically symlinked to `${HC_TESTSET_PATH}/all/actions/*`).

Scripts without the execution bit set as well as those with names matching the regular expression `(^[^a-zA-Z0-9])|(~$)|(\.bak$)|(\.orig$)|(\.rpmnew$)|(\.rpmorig$)|(\.rpmsave$)` are silently ignored.

A number of environment variables are passed to the action script:

TESTSET
: The name of the test set. *0*, which means no timeout.

TS_RESULT
: The result of the testset. (0 = OK, 1 = WARNING, 2 = ERROR)

TS_COUNT_TOTAL, *TS_COUNT_OK*, *TS_COUNT_WARN*, *TS_COUNT_ERR*
: Total number of tests run or number of tests which returned OK, WARNING or ERROR, respectively.

TS_LIST_OK, *TS_LIST_WARN*, *TS_LIST_ERR*
: Comma-separated lists of tests returning OK, WARNING or ERROR, respectively.

For a typical action script, refer to Appendix D, *Sample action script*.

Chapter 5. Running the *Healthchecker*

The *Healthchecker* may be run using the command **pshealthcheck**. At least a test set name must be provided as an argument. For options of the **pshealthcheck** refer to pshealthcheck(1), Synopsis.

```
# pshealthcheck manual
[++] Testset "manual": Total 37, Ok 37, Warn 0, Err 0 (Timeout 0)
```

Figure 5.1. Example **pshealthcheck** output

As the example shows, the *Healthchecker* prints a statistic about total number of test, the number of tests failed and succeded. By default, the *Healthchecker* will run all tests and actions defined for the given test set. To run the tests only, but not to run any action, use the option `--dry-run`. To list all tests actually performed provide the option `-v`.

```
# pshealthcheck -v manual
[++] ["bios_date compute"]
[++] Checking for bios date...
[++] ["bios_version compute"]
[++] Checking for bios version...
[++] ["cpu_count 8"]
[++] Checking for number of cpus...
[++] ["cpu_type"]
[++] Checking for cpu types...
[++] ["daemons common"]
[++] Checking for daemons...
[++] ["daemons parastation"]
[++] Checking for daemons...
[++] ["daemons pbs_mom"]
[++] Checking for daemons...
[++] ["disc_free"]
[++] Checking for available disc space...
[++] ["disc_smart"]
[++] Checking for smart...
[++] ["infiniband_counters compute"]
[++] Checking for infiniband error counters...
[++] ["infiniband_phy_state"]
[++] Checking for physical infiniband state...
[++] ["infiniband_speed"]
[++] Checking for infiniband speed...
[++] ["infiniband_state ACTIVE"]
[++] Checking for infiniband state...
[++] ["ipmi"]
[++] Checking for ipmi...
...
[++] Testset "manual": Total 37, Ok 37, Warn 0, Err 0 (Timeout 0)
```

Figure 5.2. Example verbose **pshealthcheck** output

To show all configured tests for a test set, run the command **pshcgetconf**.

```
# pshcgetconf manual

[<Test Name>] '<Test Command>' '<Test Timeout>' '<Kill Wait Time>'

================================================================

[bios_date admin:gpfs:login] \
'/opt/parastation/lib/checks/bios_date.sh "05/04/2009"' '0' '1'
[bios_version admin:gpfs:login] \
'/opt/parastation/lib/checks/bios_version.sh "R4232X20"' '0' '1'
[daemons common]        '/opt/parastation/lib/checks/daemons.sh \
"syslog-ng" "ntpd" "sshd"' '0' '1'
[daemons admin] '/opt/parastation/lib/checks/daemons.sh "pscollect" \
"dhcpd" "named"' '0' '1'
[daemons parastation]   '/opt/parastation/lib/checks/daemons.sh \
"xinetd" "psid"' '0' '1'
[kernel admin:sm]       '/opt/parastation/lib/checks/kernel.sh \
"2.6.27.19-5-default"' '0' '1'
[md5_sum admin] '/opt/parastation/lib/checks/md5_sum.sh \
${HC_CONF_DIR}/data/admin.md5sum' '0' '1'
[memory_size]   '/opt/parastation/lib/checks/memory_size.sh "24"' \
'0' '1'
[mounts common] '/opt/parastation/lib/checks/mounts.sh "boot" "tmp" \
"var"' '0' '1'
[nameserver]    '/opt/parastation/lib/checks/nameserver.sh' '0' '1'

[net_counters]  '/opt/parastation/lib/checks/net_counters.sh' '0' '1'

[net_ports_tcp common] '/opt/parastation/lib/checks/net_ports_tcp.sh \
"sshd=22"' '0' '1'
[net_ports_tcp admin] '/opt/parastation/lib/checks/net_ports_tcp.sh \
"pscollect=4000" "named=53"' '0' '1'
...
```

Figure 5.3. Example **pshcgetconf** output

To print all configured test sets, use the command **pshcgetconf -l**:

```
# pshcgetconf -l
epilogue
ib-test
manual
mce-stats
mem-test
prologue
reboot
stress-test
```

Figure 5.4. Example **pshcgetconf -l** output

Reference Pages

This appendix lists all available reference pages related to the *ParaStation Healthchecker*.

healthcheck.conf

healthcheck.conf — *ParaStation Healthchecker*: global configuration file

DESCRIPTION

This configuration file defines the overall behaviour of the *ParaStation Healthchecker*, especially the paths used to locate tests and checks.

This file is sourced by Bash shell scripts, therefore the syntax to define parameters is identical to Bourne shell, e.g. **HC_SYSLOG_FACILITY=*user***. Comments are indicated by **#**.

PARAMETERS

The following parameters define global values used for the *Healthchecker*:

HC_CONF_DIR

Defines the directory containing all *Healthchecker* configuration files. Defaults to *${PSconfDir}* if set, or */etc/parastation/healthcheck* otherwise.

HC_TESTCONF_PATH

Path of directories to be searched for test configuration files. Defaults to *${HC_CONF_DIR}/testsets*.

HC_TESTSET_PATH

Path of directories to be searched for testset configurations. Defaults to *${HC_CONF_DIR}/testsets*.

HC_SYSLOG_FACILITY

Syslog facility to use for logging. Defaults to *user*.

SEE ALSO

pshcgetconf(1) and pshealthcheck(1)

COPYRIGHT

pshealthcheck

pshealthcheck — ParaStation Healthchecker

Synopsis

pshealthcheck [-c *config_path*] [-p *testsets_path*] [-t *timeout*] [-l] [-n] [-v [-v]] [-x [-x]] [--no-update] [--help | -h] *testset*

DESCRIPTION

pshealthcheck is a program for running some tests to check the health state of the local node. With each call it runs a configured *testset* specifying which tests to do.

The configuration of the tests in a *testset* can be done in to ways:

- As sections in one of the test configurations files, mostly using one of the prepared check scripts coming with the pshealtcheck distribution.
- As stand alone test configuration script in one of the *testset*'s configuration scripts directory.

OPTIONS

`-c` *config_path*
: The path to look for test configurations files (override global configfile)

`-p` *testsets_path*
: The path to look for *testset* directories containing test configuration scripts. (override global configfile)

`-t` *timeout*
: Timeout for the run of the *testset* in seconds. If the *testset* does not finish after *timeout* seconds, the running test is killed and the *testset* returns immediately reporting an error. (override *testset* configfile)

`-l`
: Turns on logging via syslog. (Default is to report via STDOUT.)

`-v`
: Increase verbosity.

`-x`
: Extend the *testset* statistical summary by the name of all tests with warning or error return state. (give `-x` twice for listing of OK tests)

`--no-update`
: Do not try to update the configuration by running "psconfig update". (Default is to run update.)

`--no-action`
: Activate no-action-mode. Runs tests but never execute any action script.

`--dry-run`
: Activate dry-run-mode. Do not run tests or actions, only check the configuration and report witch tests and actions would run.

testset
: Name of the *testset* to run.

CONFIGURATION

== Testsets ==

A new *testset* is created by making a new *testset* directory tree as subtree of one of the directories configured in ${HC_TESTSET_PATH}.

There has to be at least the file testset.conf in the *testset* directory. An empty testset.conf is supported.

Options in testset.conf:

TS_TIMEOUT
: Timeout for the entire *testset* in seconds (Default: 0 (no *timeout*))

TS_BREAK
: *testset* break condition (Default: "never")

- "first": break on first error
- "never": never break, always run all tests

== Tests ==

Tests can be configured in two ways:

- Writing a testblock in a test configuration file
- Putting a test script into the all tests directory and creating a symlink in the *testset*'s tests directory

== Test Configuration Files ==

pshcgetconf(1) looks for test configuration files in all directories given in ${HC_TESTCONF_PATH}. All files found are read one by one in alphabetical order (as done be bash's pathname expansion).

```
[ethernet_eth0]
classes = cs:admin
testsets = reboot:prologue
test = /opt/parastation/lib/checks/ethernet eth0 100
timeout = 10
killwait = 2

[md5sum_parastation_cs]
classes = cs
testsets = reboot
test = /opt/parastation/lib/checks/md5sum /etc/parastation/classes/cs/md5sums

[md5sum_parastation_admin]
classes = admin
testsets = reboot
test = /opt/parastation/lib/checks/md5sum /etc/parastation/classes/admin/md5sums
```

Parameters:

classes
: (mandatory) Connected node classes for this test. (colon separated)

testsets
: (mandatory) Testsets containing this test. (colon separated)

test
: (mandatory) Command to be executed. Everything right of the equal sign is passed as-is to bash's eval.

timeout
: Timeout for the test is seconds. If the *timeout* is reached before the test returns, it is handled as if it returned an error. (Default: 0 (no *timeout*))

killwait
Time to wait between SIGTERM and SIGKILL in case the *timeout* of the test is reached. (Default: 1)

== Testset Directories ==

pshcgetconf(1) looks for testsets in all directories given in ${HC_TESTSET_PATH}. All directories found there are assumed to be *testset* directories with the subdirectories tests and actions and the file testset.conf.

== Test Scrips ==

Test scripts are very small scripts configuring and doing one test each.

```
#!/bin/bash
#classes cn:admin:node034
#testname Ethernet_Test
#timeout 10
#killwait 2
/opt/parastation/lib/checks/ethernet eth0 100
```

Parameters:

#classes
(mandatory) Connected node classes for this test. (colon separated)

#testname
(recommended) Name for the test used in output. (Default: Name of the test script)

#timeout
Timeout for the test is seconds. If the *timeout* is reached before the test returns, it is handled as if it returned an error. (Default: 0 (no *timeout*))

#killwait
Time to wait between SIGTERM and SIGKILL in case the *timeout* of the test is reached. (Default: 1)

Test scripts are executed after the tests defined in the test configuration files. They are meant for easily creating specialized or temporary tests by just dropping in a file to the correct place.

== Actions ==

The actions of a *testset* are executed as last step of the run. All scripts in the *testset*'s action directory are executed in lexical order. Scripts without the executable bit set as well as those with names matching the regexp (^[^a-zA-Z0-9])|(~$)|(.bak$)|(.orig$)|(.rpmnew$)|(.rpmorig$)|(.rpmsave$) are silently ignored.

The action scripts are executed in an environment with the following variables set:

TESTSET
The name of the *testset*.

TS_RESULT
The global result of the *testset* (0=SUCCESS, 1=WARN, 2=ERROR)

TS_COUNT_TOTAL
Total number of tests in the *testset*.

TS_COUNT_OK
Number of successful tests in the current run.

TS_COUNT_WARN
Number of test with warnings in the current run.

TS_COUNT_ERR
Number of failed tests in the current run.

TS_LIST_OK
List of successful tests in the current run. (", `"-separated`)

TS_LIST_WARN
List of tests with warning in the current run. (", `"-separated`)

TS_LIST_ERR
List of failed tests in the current run. (", `"-separated`)

FILES

`/opt/parastation/bin/pshealthcheck`
Healthchecker executable

`/opt/parastation/bin/pshcgetconf`
Script reading the configuration. (pshcgetconf(1))

${PSconfDir}/general/healthcheck.conf
General configuration file for **pshealthcheck**

*${HC_TESTCONF_PATH}/**
Tests configuration files

*${HC_TESTSET_PATH}/all/tests/**
All test scripts

*${HC_TESTSET_PATH}/all/actions/**
All action scrips

*${HC_TESTSET_PATH}/${TESTSET}/tests/**
Symlinks to test scripts to be executed in *testset* ${TESTSET} in classes given by magic line

*${HC_TESTSET_PATH}/${TESTSET}/actions/**
Symlinks to action scripts to be executed after *testset* ${TESTSET} failed

${HC_TESTSET_PATH}/${TESTSET}/testset.conf
Configuration file for *testset* ${TESTSET}

SEE ALSO

pshcgetconf(1)

COPYRIGHT

pshcgetconf

pshcgetconf — ParaStation Healthchecker Configuration Reader

Synopsis

pshcgetconf [-c *config_path*] [-p *testsets_path*] [-k *classes*] [-b] [-d] [-l] [--help | -h] *testset*

DESCRIPTION

pshcgetconf is part of the *ParaStation* Healthchecker. It reads and shows the configuration of the Healthchecker. This command is used by the pshealthcheck script and is intended to be used by the system administrator, too. It reads in the entire pshealthcheck configuration of the local node and provides it in two possible formats controlled by commandline parameters.

See the pshealthcheck(1) manpage for more information about the configuration.

OPTIONS

`-c` *config_path*
: The path to look for test configurations files (override global configfile)

`-p` *testsets_path*
: The path to look for *testset* directories containing test configuration scripts.

`-k` *classes*
: print configuration as if the local node would be in *classes*

`-b`
: print in format readable by bash "read" buildin

`-l`
: print a list of all testsets correctly configured

`-d`
: print debugging information

testset
: Name of the *testset* to run.

NOTE

pshcgetconf does not update the configuration by calling 'psconfig update' as pshealthcheck does.

SEE ALSO

pshealthcheck(1)

COPYRIGHT

Appendix A. List of implemented checks

This appendix lists all currently implemented checks. For a detailed description of the tests, refer to the check script itself.

Name	Description
bios_date.sh	Check BIOS date
bios_version.sh	Check BIOS version
cpu_count.sh	Check number of found cores
cpu_speed.sh	Check CPU speed
cpu_type.sh	Check CPU type
daemons.sh	Check running daemons
disc_free.sh	Check free disk space
disc_smart.sh	Check disk's SMART state
hpl.sh	Check HPL performance
infiniband_bandwidth.sh	Check Infiniband bandwidth
infiniband_counters.sh	Check Infiniband counters
infiniband_phy_state.sh	Check Infiniband physical state
infiniband_speed.sh	Check Infiniband link speed
infiniband_state.sh	Check Infiniband state
ipmi.sh	Check IPMI accessability
kernel.sh	Check kernel version
ldap.sh	check LDAP availability
md5_sum.sh	Check md5 sum of files
memory_free.sh	Check free memory
memory_mcelog.sh	Check MCE entries
memory_not_reclaimable.sh	Check not reclaimable memory
memory_size.sh	Check total memory size
memory_speed.sh	Check memory speed
memory_stress.sh	Check memory
mounts.sh	Check mount points
nameserver.sh	Check nameserver availablity
net_counters.sh	Check network counters
net_ping.sh	Check node reachability
net_ports_tcp.sh	Check TCP ports
net_ports_udp.sh	Check UDP ports
net_speed.sh	Check network interface speed
pbs_note.sh	Check for pbsnodes entries
pbs_prologue.sh	Check for prologue/epilogue errors
process_cleanup.sh	Check for job leftovers
psid_cpus.sh	Check for # of CPUs in psid
psid.sh	Check for psid
service.sh	Check for missing services
software_versions.sh	Check for various software versions
syslog_mce_errors.pl	Check for MCE errors in logfile

Table A.1. List of implemented checks

Appendix B. Extending the *Healthchecker*

This appendix explains how to customize the *Healthchecker* to particular needs. This includes adding new tests, adding new actions or even entire new test sets.

B.1. Adding new tests

To add a new test, the appropriate script must be copied to the directory `/opt/parastation/lib/checks`. The script must be executable, otherwise it's silently ignored. The script may only return values 0, 1 or 2.

Add a new test to one of the test configuration files in `/etc/parastation/healthcheck/testconf.d`. You may also create a new file within this directory. Add a new section with the new test name within brackets and add at least the key words *`classes`*, *`testsets`* and *`test`*. The later one should point to the new check.

Check the configuration by running **pshcgetconf *`testset`*** and **pshealthcheck *`--dry-run testset`***. Refer to Section 4.2, “Configuring the test” for details.

B.2. Adding new actions

Drop an action script to the directory `/etc/parastation/healthcheck/testsets/all/actions` and add a symbolic link to this script with the appropriate test set directory (`/etc/parastation/healthcheck/testsets/`*`testset`*`/actions`).

The script is called upon every run of **pshealthcheck**, therefore it should inspect the variable *TS_RESULT* about the result of all tests. Refer to Section 4.5, “Configuring actions” for more variables provided to the script.

B.3. Adding new test sets

New test sets may be added by creating a new directory within `/etc/parastation/healthcheck/testsets`. The directory name defines the test set name. Within this directory, create a file called `testset.conf`. An empty file is ok. For available configuration parameters refer to Section 4.3, “Test set configuration”.

To add tests to the new test set, modify the test configuration file. Add the new test set to the *`testsets`* entry of the particular test.

Appendix C. How to determine a node's class

The *Healthchecker* has no idea about the role of a node or the classes a node belongs to. This is beyond the *Healthchecker*'s scope.

The *Healthchecker* uses the command **psconfig** to retrieve this information. This command is part of the *ParaStation Cluster Suite*.

If this command is not available, a suitable replacement has to be provided. The command is called like:

```
# psconfig get jj01c01/NodeClasses
compute:jj01c01
```

Figure C.1. **psconfig** calling example

The variable *jj01c01* should be replaced by the actual node name. The command prints a list of classes, separated by colon. The list includes the node's own name, as each node defines by default a class of it's own.

Appendix D. Sample action script

Figure D.3, “Example test script”, shows a sample action script to set a node offline within a resource management system, namely Torque, if an error occured.

The script called `pbs_set_offline.sh` should be copied to the directory `/etc/parastation/healthcheck/testsets/all/actions`. A symlink in the directory `/etc/parastation/healthcheck/testsets/`*`testset`*`/actions` should point to this script.

After a failed healthcheck run with test set *`testset`*, the local node would be set off-line with an appropriate comment, like

```
pshealthcheck - 2010-09-01 01:10:23 - 1/31 - ethernet_eth0 - prologue
```

Figure D.1. Example action script output

Use the command **pbsnodes -ln** to check for automatically off-lined nodes:

```
# pbsnodes -ln
jf49c02              offline                   pshealthcheck - 2010-09-01 \
  01:10:23 - 1/31 - ethernet_eth0 - prologue
...
```

Figure D.2. Sample **pbsnodes** output

```
#!/bin/bash
#
#               ParaStation Healthchecker
#
# Copyright (C) 1999-2004 ParTec AG, Karlsruhe
# Copyright (C) 2005-2010 ParTec Cluster Competence Center GmbH,
# Munich
#
# Variables exported by calling pshealthcheck:
#   VERBOSE, LOGGING, TESTSET,
#   TS_COUNT_OK, TS_COUNT_WARN, TS_COUNT_ERR,
#   TS_LIST_OK, TS_LIST_WARN, TS_LIST_ERR


MAX_PBS_NOTE="55"
HOSTNAME=`hostname`
FAILED_SCRIPTS="${TS_LIST_ERR//, /,}"

# calculate statistics
((TS_COUNT_TOTAL=TS_COUNT_OK+TS_COUNT_WARN+TS_COUNT_ERR))

# set the node offline
msg=$(pbsnodes -ln "$HOSTNAME" | awk '{print $3}' 2>/dev/null)
[ "$VERBOSE" -ge 2 ] && echo "Setting node '$HOSTNAME' in pbs offline
..."

if [ -z "$msg" ]; then
        if [ "${#FAILED_SCRIPTS}" -gt "$MAX_PBS_NOTE" ]; then
            FAILED_SCRIPTS="${TS_LIST_ERR:0:$MAX_PBS_NOTE-3}"
            FAILED_SCRIPTS="$FAILED_SCRIPTS..."
        fi

        new_msg="pshealthcheck - `date +%Y-%m-%d\ %H:%M:%S`"
        new_msg="${new_msg} - (${TS_COUNT_ERR}/${TS_COUNT_TOTAL})"
        new_msg="${new_msg} - $FAILED_SCRIPTS - ts $TESTSET"

        pbsnodes -o -N "${new_msg}" "$HOSTNAME" || {
                echo "ERROR: setting node offline failed!";
                exit 2;
        }
else
        pbsnodes -o "$HOSTNAME" || {
                echo "ERROR: setting node offline failed!";
                exit 2;
        }
fi
```

Figure D.3. Example test script

Appendix E. Including the *Healthchecker* in a resource management system

The *ParaStation Healthchecker* may be included within a resource management system to automatically check nodes before and after running a job. Failed nodes may be set off-line, therefore they may be no longer used for further jobs.

This appendix describes the configuration for `Torque`. Other batch queuing system may use a similar approach.

E.1. Using the *Healthchecker* within a job's prologue

Figure E.1, “Sample prologue file”, shows a prologue script suitable to run the *ParaStation Healthchecker* on each node prior to the actual job start. In case of a problem, the local node may be automatically set off-line. See Appendix D, *Sample action script*, how to automatically off-line a node.

If any prologue script fails with an exit code of 2, the job is terminated and immediately re-scheduled. Therefore, the job is typically instantaniously re-run on a slightly different set of nodes. The failed node is set off-line and an appropriate note is appended to this node's resource list.

Removing a node and restarting the job is transparent to the users. The administrators should check every so often the list of off-lined nodes for corresponding messages. Alternatively, the action script may be enlarged to notify the administrator via email.

```
#!/bin/bash
#               ParaStation Healthcheck
#
# Copyright (C) 2009 ParTec Cluster Competence Center GmbH, Munich
#

# Prologue script arguments:
export PBS_JOBID=$1
export PBS_USER=$2
export PBS_GROUP=$3
export PBS_JOBNAME=$4
export PBS_LIMITS=$5
export PBS_QUEUE=$6
export PBS_ACCOUNT=$7

# start the ParaStation Healthchecker
#
# set timeout: [-t 240]
# log to syslog: [-l]
#
OLD_PATH=$PATH
export PATH=/sbin:/bin:/usr/sbin:/usr/bin:/opt/parastation/bin
/opt/parastation/bin/pshealthcheck.ng -t 240 -l prologue \
  &> /tmp/pshealthcheck_prologue.out
PSHC_EXIT="$?"
PATH=$OLD_PATH

# on any error force an exit status of "2" to requeue the job in
# torque
if [ "$PSHC_EXIT" -eq "0" ] || [ "$PSHC_EXIT" -eq "1" ]; then
    exit 0;
fi
exit 2;
```

Figure E.1. Sample prologue file

This script has to be copied to the file `/var/spool/torque/mom_priv/prologue`. For more information about prologue scripts refer to the `Torque` documentation.

The above example runs the *Healthchecker* with the test set *`prologue`*. It re-defines the test set's timeout to 240 seconds, which is suitable for very large systems only. In addition, each run is recorded in the system's logfile using option `-l` of **pshealthcheck**.

E.2. Using the *Healthchecker* within a job's epilogue

Similar to a prologue script, the *Healthchecker* may be run after a job terminates using an epilogue script. To do so, the following epilogue script should be copied to the file `/var/spool/torque/mom_priv/epilogue`:

```
#!/bin/bash
#               ParaStation Healthcheck
#
# Copyright (C) 2009 ParTec Cluster Competence Center GmbH, Munich
#

# Epilogue script arguments:
export PBS_JOBID=$1
export PBS_USER=$2
export PBS_GROUP=$3
export PBS_JOBNAME=$4
export PBS_SESSION_ID=$5
export PBS_LIMITS=$6
export PBS_RESSOURCES=$7
export PBS_QUEUE=$8
export PBS_ACCOUNT=$9

# start the ParaStation Healthchecker
#
# set timeout: [-t 240]
# log to syslog: [-l]
OLD_PATH=$PATH
export PATH=/sbin:/bin:/usr/sbin:/usr/bin:/opt/parastation/bin
/opt/parastation/bin/pshealthcheck.ng -t 240 -l epilogue \
  &> /tmp/pshealthcheck_epilogue.out
PSHC_PID="$!"
PATH=$OLD_PATH

# always exit with 0 to prevent setting the node down in moab
exit 0
```

Figure E.2. Sample epilogue file

The above example runs the *Healthchecker* with the TEST SET *epilogue*. It also re-defines the TEST SET's timeout to 240 seconds, which again is suitable for very large systems only. Again, each run is recorded in the system's logfile.

Glossary

Check	A reusable execution unit doing a single kind of check. Checks are used in the *Healthchecker* configuration to specify tests.
Class	Each test can be connected to one or more classes. The test is only executed on nodes that belong to one of its connected classes.
Compute Node	Server dedicated for computing tasks. Controlled and managed by a management node, aka frontend.
Golden Client	A node defined as a reference installation for a group of other nodes. The setup of this node is copied to other nodes using a one-to-one copy stored on an image server.
Master Node	The evaluation of temporary node lists while spawning new tasks is done only by one particular psid(8) within the cluster. The node running this daemon is called *master node*. The master node is dynamically selected within the cluster and may change, if the current master node is no longer available. Election is based on the node IDs, refer to parastation.conf(5).
MPI	Message Passing Interface.
MPI Network	Dedicated network for MPI data.
ParaStation	Cluster Middleware including MPI libraries and advanced process management.
Test	An execution unit including all needed configuration that returns one of the exit states 0 (SUCCESS), 1 (WARNING), or 2 (ERROR). In case of a warning or error a text string describing the problem can be returned.
Test set	A set of tests executed together (depending on connected classes) and returning the highest exit state of all included tests.