Linux on System z
Introducing the Linux Health Checker

Martin Schwidefsky
IBM Lab Böblingen, Germany

March 12 2014
Session 13517

Trademarks & Disclaimer

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries. For a complete list of IBM Trademarks, see www.ibm.com/legal/copytrade.shtml:

IBM, the IBM logo, BladeCenter, Calibrated Vectored Cooling, ClusterProven, Cool Blue, POWER, PowerExecutive, Predictive Failure Analysis, ServerProven, System p, System Storage, System x , System z, WebSphere, DB2 and Tivoli are trademarks of IBM Corporation in the United States and/or other countries. For a list of additional IBM trademarks, please see http://ibm.com/legal/copytrade.shtml.

The following are trademarks or registered trademarks of other companies: Java and all Java based trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries or both Microsoft, Windows,Windows NT and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both. Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
UNIX is a registered trademark of The Open Group in the United States and other countries or both. Linux is a trademark of Linus Torvalds in the United States, other countries, or both. Cell Broadband Engine is a trademark of Sony Computer Entertainment Inc. InfiniBand is a trademark of the InfiniBand Trade Association.
Other company, product, or service names may be trademarks or service marks of others.

NOTES: Linux penguin image courtesy of Larry Ewing (lewing@isc.tamu.edu) and The GIMP

Any performance data contained in this document was determined in a controlled environment. Actual results may vary significantly and are dependent on many factors including system hardware configuration and software design and configuration. Some measurements quoted in this document may have been made on development-level systems. There is no guarantee these measurements will be the same on generally-available systems. Users of this document should verify the applicable data for their specific environment. IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.
Information is provided “AS IS” without warranty of any kind. All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.
All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.
Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices are suggested US list prices and are subject to change without notice. Starting price may not include a hard drive, operating system or other features.
Contact your IBM representative or Business Partner for the most current pricing in your geography. Any proposed use of claims in this presentation outside of the United States must be reviewed by local IBM country counsel prior to such use. The information could include technical inaccuracies or typographical errors.
Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any

Agenda – Part 1

► 1. Introducing health checking

2. Using the Linux Health Checker

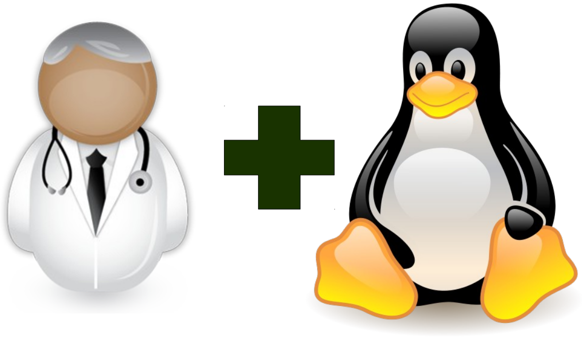
3. How to write a check

Introducing health checking

- **What is a health check?**
  - A process that identifies conditions which may lead to problems

- **What is the Linux Health Checker?**
  - A tool that performs an automated health check of a Linux system
  - Checks status and configuration
  - Presents report on identified problems

→ Helps keeping Linux systems healthy (operational)



What does it do?

- **Example problem classes**
  - Configuration errors
  - Deviations from best-practice setups
  - Hardware running at reduced capacity
  - Unused accelerator hardware
  - Single point-of-failures

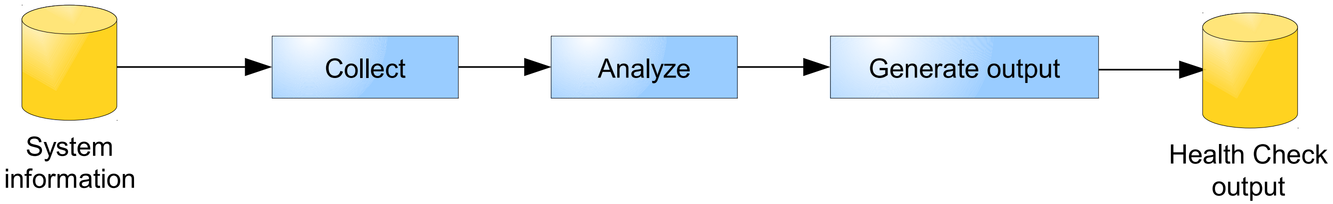
- **Detailed problem report**
  - Enable users to *understand* and *solve* problems
  - Make expert knowledge available to wider audience

Goals

- **Ease of use**
  - Simple setup: Install and run – no involved configuration
  - Primary tasks easily accessible through command line interface

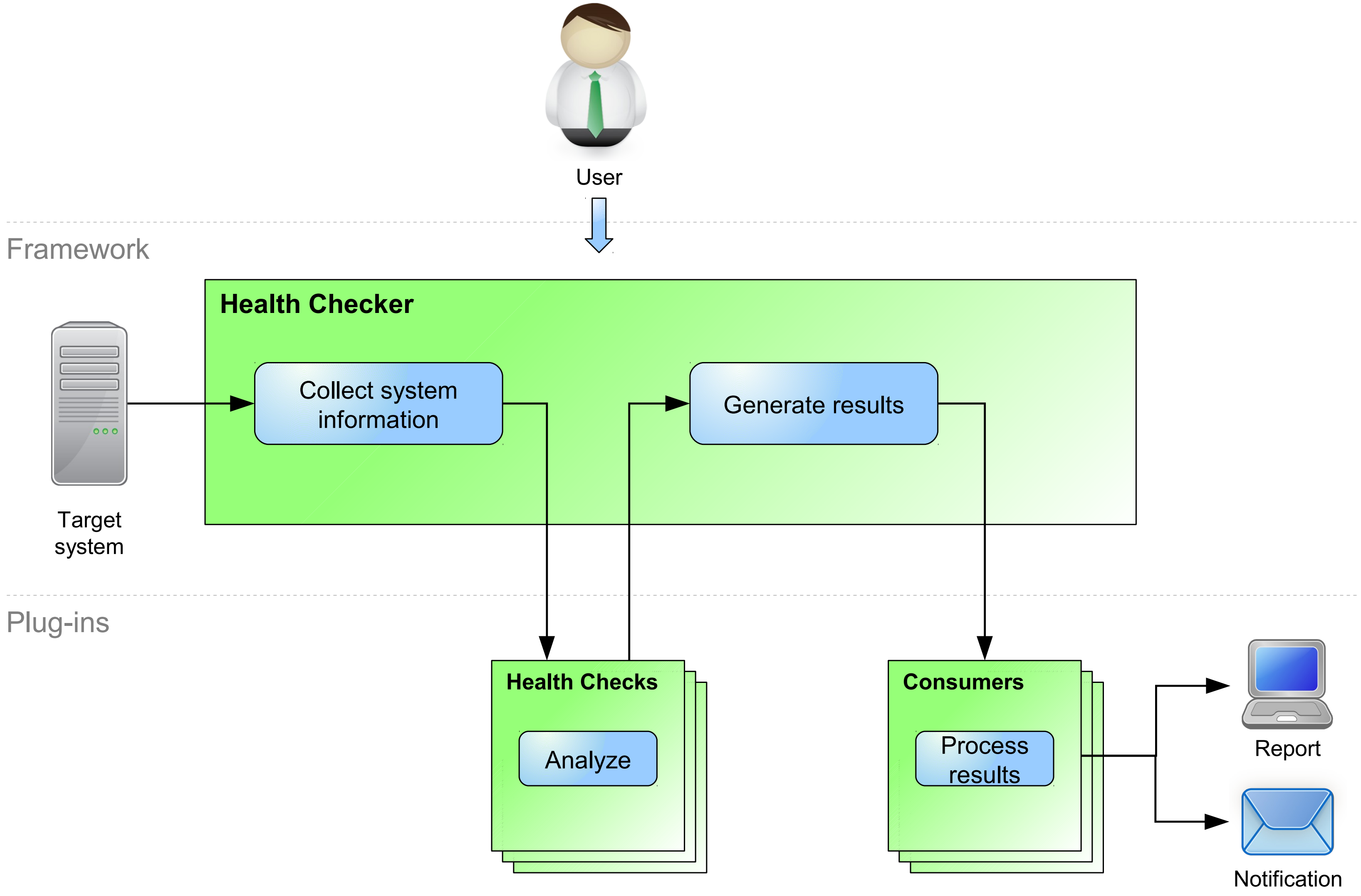
- **Flexibility through Framework/Plug-in concept**
  - Health check plug-ins
    - Contain all problem area specific knowledge
  - Consumer plug-ins
    - Handle output processing
  - Extend functionality by adding new plug-ins

Basic approach to health checking



- **Collect system information**
  - File contents, for example `/var/log/messages`
  - Program output, for example `/bin/df`
- **Analyze information**
  - Find relevant data points
  - Compare with best-practice values
- **Generate report**

System overview



User

Framework

Health Checker

Target system

Collect system information

Generate results

Plug-ins

Health Checks

Analyze

Consumers

Process results

Report

Notification

Health checks in version 1.3

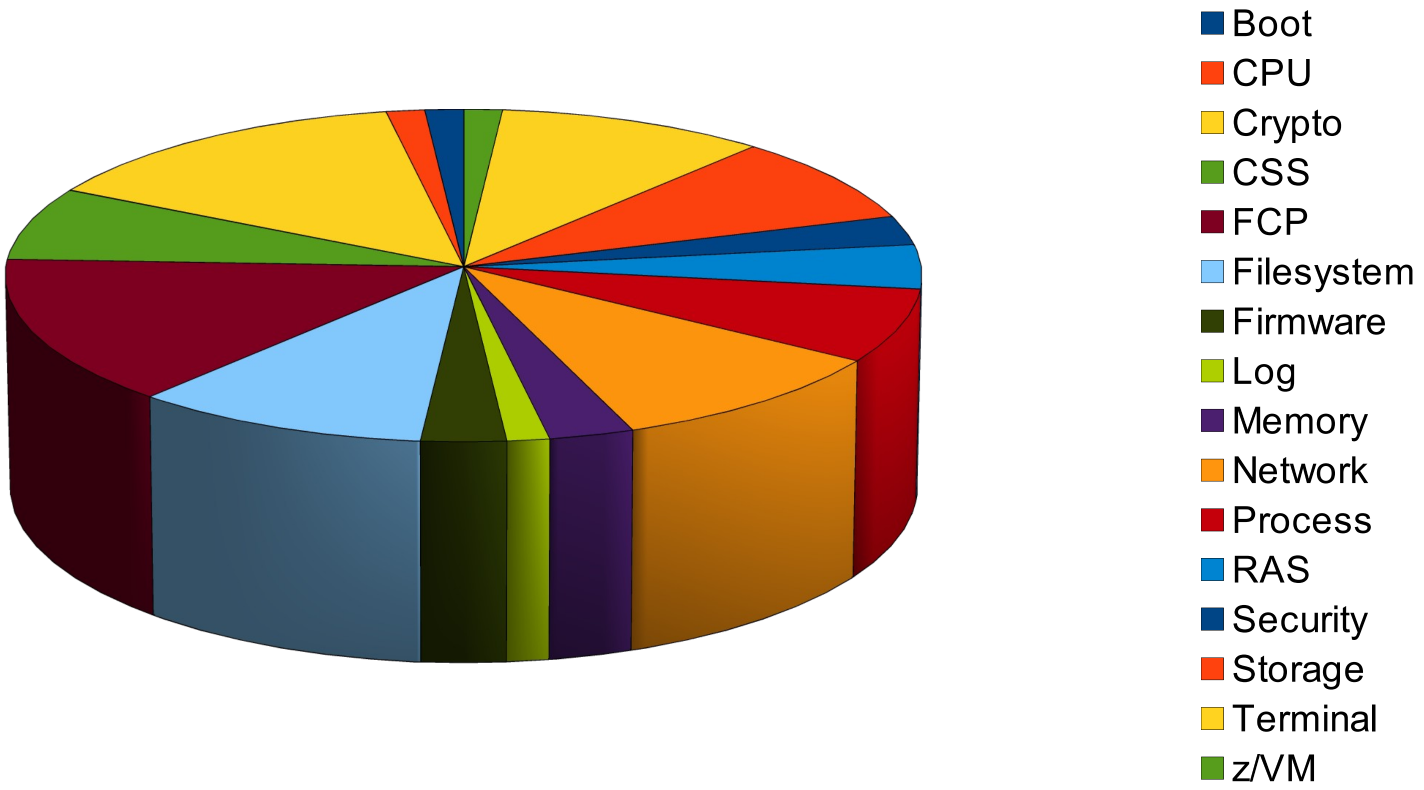
70 checks in total (v1.0 had 25):

Check whether the recommended runlevel is used and set as default
Check whether the CPUs run with reduced capacity
Confirm that CPACF is enabled
Verify System z cryptographic hw support through CCA stack
Verify System z cryptographic hw support for PKCS#11 clear key […]
Verify System z cryptographic hw support for PKCS#11 clear key […]
Verify System z cryptographic hw support for PKCS#11 secure key […]
Verify System z cryptographic hw support for PKCS#11 secure key […]
Check whether the path to the OpenSSL library is configured correctly
Verify System z cryptographic hw support through an OpenSSL stack
Verify System z cryptographic hw support through an OpenSSL stack
Confirm that the System z cryptography kernel module is loaded
Identify I/O devices that are in use although they are on the exclusion list
Check for CHPIDs that are not available
Identify unusable I/O devices
Check for an excessive number of unused I/O devices
Identify I/O devices that are not associated with a device driver
Identify unusable Fibre Channel(FC) remote ports
Verify that the bootmap file is up-to-date
Identify standard DASD device nodes in the fstab file
Check if filesystems are skipped by filesystem check (fsck)
Check file systems for an adequate number of free inodes
Check for read-only filesystems
Verify that temporary files are deleted at regular intervals.
Check file systems for adequate free space
Confirm that automatic problem reporting is activated
Check if control program identification can display Linux instance names
Verify that syslog files are rotated
Check if swap space is available
Ensure memory usage is within the threshold
Identify bonding interfaces that are configured with single network interfaces
Identify bonding interfaces that aggregate qeth interfaces with the same CHPID
Ensure nameserver is listed with correct address
Check for an excessive error ratio for outbound HiperSockets traffic
Check the inbound network traffic for an excessive error or drop ratio

Identify qeth interfaces that do not have an optimal number of buffers
Identify network services that are known to be insecure
Ensure processes do not hog cpu time
Ensure the system is running with optimal load
Check the kernel message log for out-of-memory (OOM) occurrences
Ensure processes do not hog memory
Ensure that privilege dump is switched off
Ensure kdump is configured and running
Confirm that the dump-on-panic function is enabled
Ensure that panic-on-oops is switched on
Identify unusable SCSI devices
Confirm that root logins are enabled for but restricted to secure terminals
Screen users with superuser privileges
Identify CDL-formatted DASD where the metadata area is used for storing data
Confirm 4K block size on ECKD DASD devices
Check Linux on z/VM for the "nopav" DASD parameter
Identify active DASD alias devices without active base device
Identify multipath setups that consist of a single path only
Identify multipath devices with too few available paths or too many failed paths
Verify that the multipath service starts automatically when the system launches
Check for two or more host ports and two or more target ports (WWPNs)
Spot getty programs on the /dev/console device
Check for current console_loglevel
Detect terminals with multiple device nodes
Confirm that all available z/VM IUCV HVC terminals are enabled for logins
Identify idle terminals
Identify idle users
Identify unused terminals (TTY)
Check whether N_Port ID Virtualization (NPIV) is active
Check if FCP device recovery failed
Identify FCP devices that share channel-path identifiers (CHPIDs)
Ensure that all LUNs configured for persistence are available
Identify if recovery of a zFCP LUN failed
Check if the recovery of a target port failed
Check the privilege classes of the z/VM guest virtual machine

Health checks in version 1.3

Checks by Component



Agenda – Part 2

1. Introducing health checking
2. ► Using the Linux Health Checker
3. How to write a check

Preparations

- **Obtaining the Linux Health Checker**
  - Releases: V1.0 released March 2012, V1.3 in December 2013
  - Open source under Eclipse Public License v1.0
  - Download RPM or source package from http://lnxhc.sourceforge.net
  - Install using RPM command or `make install`
  - Distribution support in progress

- **Requirements**
  - Linux
    - Framework should run on *any* hardware platform
    - Health checks may be platform specific
  - Perl 5.8 or later
    - Additional Perl modules which are usually part of default installation

First health check run

```
[user@lnxhost ~]$ lnxhc run
Collecting system information
Running checks (12 checks)
CHECK NAME                               HOST                     RESULT
=================================================================================
boot_zipl_update_required .............. lnxhost                  SUCCESS
css_ccw_availability ................... lnxhost                  SUCCESS
css_ccw_chpid .......................... lnxhost                  SUCCESS
css_ccw_no_driver ...................... lnxhost                  SUCCESS
css_ccw_unused_devices ................. lnxhost                  EXCEPTION-LOW

 >EXCEPTION css_ccw_unused_devices.many_unused_devices(low)
    Of 4664 I/O devices, 4659 (99.89%) are unused

fs_disk_usage .......................... lnxhost                  SUCCESS
mm_oom_killer_triggered ................ lnxhost                  SUCCESS
net_hsi_tx_errors ...................... lnxhost                  NOT APPLICABLE
ras_dump_on_panic ...................... lnxhost                  EXCEPTION-HIGH

 >EXCEPTION ras_dump_on_panic.no_standalone(high)
    The dump-on-panic function is not enabled

sec_services_insecure .................. lnxhost                  SUCCESS
sys_sysctl_call_home ................... lnxhost                  NOT APPLICABLE
sys_sysinfo_cpu_cap .................... lnxhost                  SUCCESS

10 checks run, 2 exceptions found (use 'lnxhc run --replay -V' for details)
```

Interpreting output

- **A potential problem was found**

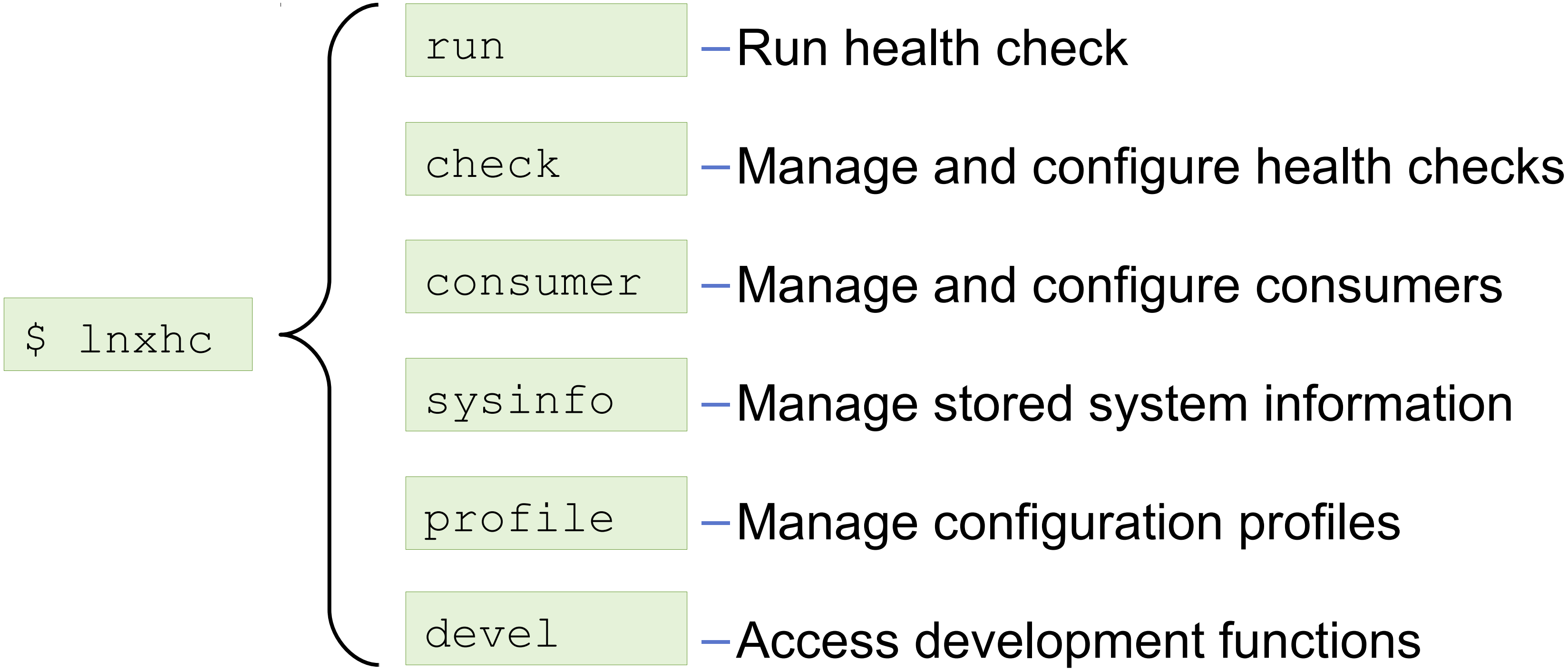
```
css_ccw_unused_devices ................. lnxhost                  EXCEPTION-LOW

 >EXCEPTION css_ccw_unused_devices.many_unused_devices(low)
    Of 4664 I/O devices, 4659 (99.89%) are unused
```

  - Full exception ID
    - `css_ccw_unused_devices.many_unused_devices`
  - Exception severity
    - `low`
  - Exception summary
    - `Of 4664 I/O devices, 4659 (99.89%) are unused`

Getting more details

```
[user@lnxhost ~]$ lnxhc run -V css_ccw_unused_devices
CHECK NAME                               HOST                               RESULT
======================================================================================
css_ccw_unused_devices ................. lnxhost                            EXCEPTION-LOW

 >EXCEPTION css_ccw_unused_devices.many_unused_devices(low)

  SUMMARY
    Of 4664 I/O devices, 4659(99.89%) are unused

  EXPLANATION
    The  number  of  unused (offline) I/O devices, 4664 (99.89%) of a total of 4659,
    exceeds the specified threshold. During the boot process, Linux senses and analyzes
    All available  I/O devices, including unused devices. Therefore, unused devices
    unnecessarily consume memory and CPU time.

  SOLUTION
    Use the "cio_ignore" feature to exclude I/O devices that you do not need from  being
    sensed  and  analyzed. Be sure not to inadvertently exclude required devices. To ex-
    clude devices, you can use the "cio_ignore" kernel parameter or a command like this:

    echo "add <device_bus_id>" > /proc/cio_ignore

    where <device_bus_id> is the bus ID of an I/O device to be excluded.

  REFERENCE
    For  more  information  about  the  "cio_ignore"  feature, see the section about the
    "cio_ignore" kernel parameter in "Device Drivers, Features, and Commands".
```

Additional functions



`$ lnxhc`

- `run` – Run health check
- `check` – Manage and configure health checks
- `consumer` – Manage and configure consumers
- `sysinfo` – Manage stored system information
- `profile` – Manage configuration profiles
- `devel` – Access development functions

Viewing health check information

```
[user@lnxhost ~]$ lnxhc check --info fs_disk_usage

Check fs_disk_usage (active)
============================

Title:
  Check file systems for adequate free space

Description:
  Some applications and administrative tasks require an adequate amount of free space on
  each mounted file system. If there is not enough free space, these applications might
  no longer be available or the complete system might be compromised. Regular monitoring
  of disk space usage averts this risk.

Exceptions:
  critical_limit=high (active)
  warn_limit=low (inactive)

Parameters:
  critical_limit=95
        File system usage (in percent) at which to raise a high-severity exception.
        Valid values are integers in the range 1 to 100.

        Default value is "95".

...
```

Modifying health check properties

- **Activation state**
  - Specifies if a check should be performed during health check run

```
[user@lnxhost ~]$ lnxhc check fs_disk_usage --state inactive
Setting state of check 'fs_disk_usage' to 'inactive'
Done.
```

- **Parameter values**
  - Values defined by health checks
  - Enable users to customize certain aspects of the health check

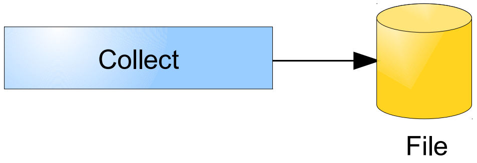
```
[user@lnxhost ~]$ lnxhc check --param fs_disk_usage.critical_limit=99
Setting value of parameter fs_disk_usage.critical_limit to '99'
Done.
```

- **See man page for full list of properties**
  - `man lnxhc_properties.7`

Advanced health checking modes

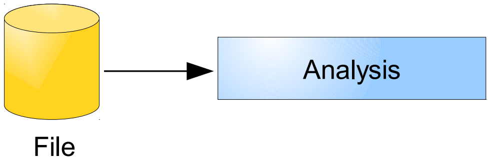
- **Collect data to file**

```
lnxhc sysinfo --collect --file lnxhost.sysinfo
```



- **Analyze from file**

```
lnxhc run --file lnxhost.sysinfo
```

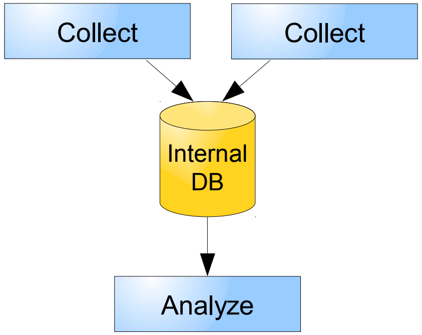


- **Analyze from remote host**

```
ssh user@remote lnxhc sysinfo -c -f - | lnxhc run -f -
```

- **Analyze from multiple hosts**

```
lnxhc sysinfo --clear
ssh user@remote1 lnxhc sysinfo -c -f - | lnxhc sysinfo --merge -
ssh user@remote2 lnxhc sysinfo -c -f - | lnxhc sysinfo --merge -
...
lnxhc run --current
```



Agenda – Part 3

1. Introducing health checking
2. Using the Linux Health Checker
3. ► How to write a check

Example idea

- **What to check?**
  - Value of sysctl setting `panic_on_oops` should be '1'

- **Why?**
  - “Kernel oops” = severe kernel error
  - Indication that the kernel can no longer be trusted
  - Kernel will continue anyway if panic_on_oops is '0'

- **How to check**

```
[user@lnxhost ~]$ cat /proc/sys/kernel/panic_on_oops
0
```

- **Solution**

```
[user@lnxhost ~]$ echo 1 > /proc/sys/kernel/panic_on_oops
```

Implementation without framework

Check program 'check.sh'

```
#!/bin/bash

FILENAME="/proc/sys/kernel/panic_on_oops"
PANIC_ON_OOPS=`cat $FILENAME`

if [ "$PANIC_ON_OOPS" -eq 0 ] ; then
        echo "The panic-on-oops setting is disabled"
        echo "Enable it using 'echo 1 > /proc/sys/kernel/panic_on_oops'"
        exit 1
fi

exit 0
```

Sample output

```
[user@lnxhost ~]$ ./check.sh
The panic-on-oops setting is disabled
Enable it using 'echo 1 > /proc/sys/kernel/panic_on_oops'
```

Writing checks for the Linux Health Checker framework

- **One directory per check**
  - Directory name is check name

- **Files for**
  - Meta data
  - Text
  - Check program

```
panic_on_oops
├── definitions
├── descriptions
├── exceptions
└── check
```

Definitions file

- Contains data about the health check

```
[check]
author = user@host
component = system
```

- Meta-data

```
[sysinfo panic_on_oops]
file = /proc/sys/kernel/panic_on_oops
```

- System information
  - Files, command output, etc.

```
[exception no_panic_on_oops]
severity = high
```

- Exceptions
  - ID and severity
- Optional parameters

Descriptions file

- **Contains health check and parameter descriptions**

```
[title]
Ensure that panic-on-oops is enabled

[description]
The panic-on-oops setting ensures that a
Linux instance is stopped if a kernel
oops occurs.
```

- Check title
- Basic check description
- Description of parameters

Exceptions file

- **Contains problem report text**
- **References exception specified in definitions file through label**

```
[summary no_panic_on_oops]
The panic-on-oops setting is disabled

[explanation no_panic_on_oops]
Without the panic-on-oops setting, a
Linux instance might keep running after
an oops.

[solution no_panic_on_oops]
Use the following command to enable the
panic-on-oops setting

echo 1 > /proc/sys/kernel/panic_on_oops

[reference no_panic_on_oops]
See kernel documentation on panic-on-oops
setting.
```

- Problem summary
- Explanation
  - Why is this a problem?
- Solution
  - Step-by-step instruction
- Reference for further reading
  - If available

Check program

- **Implements health check analysis logic**

```
#!/bin/bash
```

- Access system information

```
FILENAME=$LNXHC_SYSINFO_panic_on_oops
PANIC_ON_OOPS=`cat $FILENAME`
```

- Analyze and report exception

```
if [ "$PANIC_ON_OOPS" -eq 0 ] ; then
  echo "no_panic_on_oops" >> $LNXHC_EXCEPTION
fi
```

- Indicate result code
  - 0 = Success
  - 64 = Missing dependency
  - Other = Run-time error

```
exit 0
```

Putting it all together

```
[user@lnxhost ~]$ lnxhc run -V ./panic_on_oops
Collecting system information
Running checks (1 checks)
CHECK NAME                               HOST                            RESULT
=======================================================================================
panic_on_oops .......................... lnxhost                         EXCEPTION-HIGH

 >EXCEPTION panic_on_oops.no_panic_on_oops(high)

  SUMMARY
    The panic-on-oops setting is disabled

  EXPLANATION
    Without  the  panic-on-oops  setting,  a Linux instance might
    keep running after an oops.

  SOLUTION
    Use the following command to enable the panic-on-oops setting
    echo 1 > /proc/sys/kernel/panic_on_oops

  REFERENCE
    See kernel documentation on panic-on-oops setting.
```

- If it doesn't work, add more “`-V`”s
  - Increase level of verbosity to help debugging

Wrap-up

- **To implement a check**
  - Create a directory
  - Add files
    - Meta-data
    - Text files
    - Check program
  - Run/debug until it works

- **Health check creation dialog**

```
lnxhc devel --create-check my_check
```

  - Creates template files based on dialog input

Further reading

- **Man pages**
  - Once installed use '`apropos lnxhc`' to list man pages
  - Also available on the web: http://lnxhc.sourceforge.net/manpages.html
- **User's Guide**
  - http://lnxhc.sourceforge.net/documentation.html
- **Main web page**
  - http://lnxhc.sourceforge.net/
- **Mailing list**
  - Open for questions, comments, ideas, code contributions, etc.
  - lnxhc-list@lists.sourceforge.net

Questions?

Martin Schwidefsky

Linux on System z
Development

Schönaicher Strasse 220
71032 Böblingen, Germany

Phone +49 (0)7031-16-2247
schwidefsky@de.ibm.com