DEVELOPING WITH C++, FORTRAN, GO, AND RUST ON RED HAT ENTERPRISE LINUX 8

Matt Newsome, PhD
Senior Engineering Manager, Platform Tools Team
May 2019

http://bit.ly/rhel8tools

WHAT DEVELOPER TOOLS DOES RED HAT ENTERPRISE LINUX 8 CONTAIN?

http://bit.ly/rhel8tools

WHAT'S IN RED HAT ENTERPRISE LINUX 8?

C++	• GCC 8 [gcc, binutils, gdb, gfortran for Fortran developers, etc.] • LLVM 7 [clang, lldb, lld] • GCC 9 and later available in future releases
Rust	• Rust-toolset [rustc, cargo, clippy, rustfmt, rls]
Golang	• Go-toolset [golang]
Ancillary tools	• Make, cmake, flex, bison, byacc, ltrace, dwz, etc.
Performance analysis tools	• Valgrind, Performance Co-Pilot, SystemTap, dyninst, elfutils, etc.

HOW OFTEN WILL DEVELOPER TOOLS BE UPDATED IN RED HAT ENTERPRISE LINUX 8?

HOW OFTEN DO THE COMPONENTS UPDATE?

C++

- GCC 8 in Red Hat® Enterprise Linux® 8 as the system compiler
- Glibc-2.28, binutils-2.30
- Updates in minor Red Hat Enterprise Linux 8 releases [~6 months]
- LLVM/Clang will update to a newer upstream version [~6 months]
- Newer versions of GCC will be made available, similar to the process for Red Hat Developer Toolset

Rust

- Updated to newer upstream version ~3 months

Golang

- Updated to newer upstream version ~6 months

Ancillary tools

- Updated ~6 months, sometimes to new upstream versions

Performance analysis tools

- Updated ~6 months, sometimes to new upstream versions

GETTING STARTED WITH DEVELOPER TOOLS IN RED HAT ENTERPRISE LINUX 8

http://bit.ly/rhel8tools

GETTING STARTED

Installation in Red Hat Enterprise Linux 8

HOW TO INSTALL THEM

C++

```
$ sudo yum -y group install
"Development Tools"
(or individually, e.g.)
$ sudo yum install gcc-c++)
```

Fortran

```
$ sudo yum install gfortran
```

LLVM, Rust, and Go

```
$ sudo yum -y install llvm-toolset
$ sudo yum -y install rust-toolset
$ sudo yum -y install go-toolset
```

Where are they?

Toolchains: Red Hat Enterprise Linux 8 application stream

Libraries and binutils: BaseOS

How are they packaged?

GCC/gfortran tools: Regular RPMs

Libraries and binutils: BaseOS

GETTING STARTED

Installation on Red Hat Enterprise Linux 8

Ancillary tools

```
$ sudo yum -y install cmake
```

Performance analysis tools

```
$ sudo yum -y install valgrind
$ sudo yum -y install pcp-zeroconf
$ sudo systemctl start pmcd
$ sudo yum -y install systemtap
$ sudo stap-prep
$ sudo yum -y install valgrind
```

GETTING STARTED

Usage in Red Hat Enterprise Linux 8

HOW DO YOU USE THEM?

C++

```
$ gcc -v
$ gcc foo.c
$ g++ foo.cpp
$ clang -v
$ clang foo.c
$ clang++ foo.cpp
$ gdb a.out
```

Fortran

```
$ gfortran -v
$ gfortran foo.f90
```

GETTING STARTED

Usage in Red Hat Enterprise Linux 8

HOW DO YOU USE THEM?

Rust

```
$ rustc --version
$ rustc foo.rs
$ cargo new hello && cd hello
  && cargo run
```

Go

```
$ go version
$ go run foo.go
```

GETTING STARTED

Usage in Red Hat Enterprise Linux 8

HOW DO YOU USE THEM?

Ancillary tools

```
$ cmake -version
$ cmake .
```

Performance analysis tools

```
$ valgrind /bin/ls
$ pcp dstat
$ stap -L 'process.function
("*")' -c /bin/ls
```

MIGRATING C++, RUST, GO, AND FORTRAN PROJECTS TO RED HAT ENTERPRISE LINUX 8

MIGRATING FROM RED HAT ENTERPRISE 7

Developer toolset

GCC

- Red Hat Enterprise Linux 7 defaulted to C++98 while *Red Hat Enterprise Linux 8 defaults to C++14*
- Ada, Objective-C/Objective-C++, and GCJ *no longer supported* in Red Hat Enterprise Linux

LLVM

- Red Hat Enterprise Linux 7 has SCLs:

```
llvm-toolset-7, llvm-toolset-6.0, …
```

- Red Hat Enterprise Linux 8 has a module:

```
yum install llvm-toolset
```

MIGRATING FROM RED HAT ENTERPRISE 7

Developer toolset

Golang

- Red Hat Enterprise Linux 7 has SCLs:

```
go-toolset-7, go-toolset-1.8, …
```

- Red Hat Enterprise Linux 8 has a module:

```
yum install go-toolset
```

Rust

- Red Hat Enterprise Linux 7 has SCLs:

```
rust-toolset-7,rust-toolset-1.29,…
```

- Red Hat Enterprise Linux 8 has a module:

```
yum install rust-toolset
```

HEADLINE FEATURES IN THE VERSIONS WITHIN RED HAT ENTERPRISE LINUX 8

http://bit.ly/rhel8tools

HEADLINE FEATURES

Red Hat Enterprise Linux 8

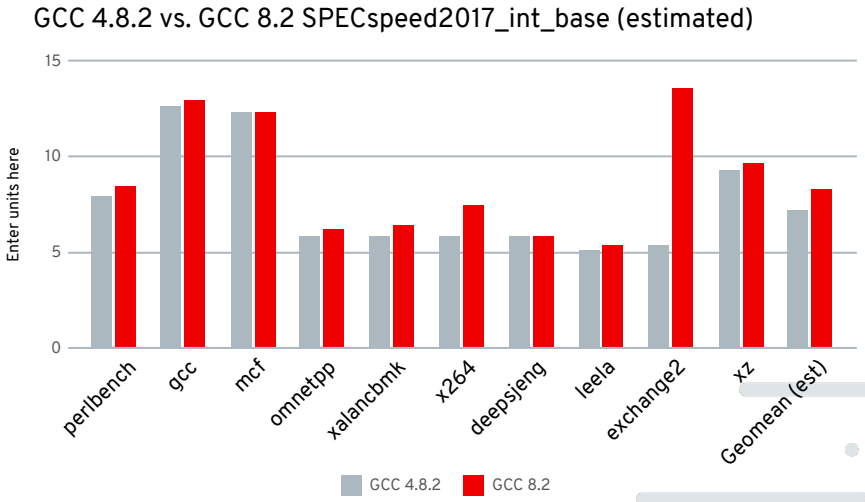
GCC

VERSION: 8

- Improved vectorization and optimization passes, especially devirtualization
- New warnings
- DWARF5 debugging support
- Full support for C++11 and C++14, and experimental support for C++17
- In-development code for C++2a (unsupported)
- C++14 is the default
- OpenMP 4.5 support
- Alongside GCC 8, Boost updated to 1.66

SPEC CPU2017 PERFORMANCE (ESTIMATED)

~15% estimated performance improvement over Red Hat Enterprise Linux 7's equivalent GCC



Source: Red Hat. Results are estimated based on measurements on a Red Hat internal non-production platform.

Used optimization flags: `-Ofast -flto -fno-fat-lto-objects -march=core-avx2 -mtune=core-avx2`

HEADLINE FEATURES

Red Hat Enterprise Linux 8

glibc

VERSION: 2.28

Standards-compliant features

- Support for ISO C threads via `#include <thread.h>`
- Unicode 11.0.0 support and localization data via ISO 14651 update
- IEEE binary128 format support (ISO/IEC TS 18661-3:2015)

- Improved malloc performance relative to Red Hat Enterprise Linux 7 (thread-local cache)
- `GLIBC_TUNABLES` environment variable for runtime tuning
- Dynamic handling of `/etc/resolv.conf` when networks change
- Improved security handling of abort()

HEADLINE FEATURES

Red Hat Enterprise Linux 8

GDB

VERSION: 8.2

- **Improved C++ debugging and usability**
  - C++11 and above features: rvalue references, alignof, etc.
  - Breakpoints match in all scopes and namespaces by default
  - No need for quoting when setting C++ breakpoints
  - Tab completion improvements (better overall, more so with C++ breakpoints)
- GDBserver virtually at feature parity with local/native debugging
- Rust language support
- DWARF5 support
- New Python scripting API extensions and abilities
- New commands, options, and fine-tuning of the CLI

HEADLINE FEATURES

Red Hat Enterprise Linux 8

Clang/LLVM

VERSION: 7.0.1

Performance

- Function multiversioning (FMV) support
- Each using a different architecture's specialized instruction-set extensions

---

- Experimental support for DWARF5 debugging information
- Speculative load hardening

Sanitizers

- MemorySanitizer, AddressSanitizer, and UndefinedBehaviorSanitizer
- Example: Implicit conversion sanitizer (part of the UndefinedBehaviorSanitizer)

```
-fsanitize=implicit-conversion
```

- Note: sanitizers also available in GCC8

HEADLINE FEATURES

Red Hat Enterprise Linux 8

Rust

VERSION: 1.31

Rust 2018 edition

- Opt-in; new keywords, module changes, and non-lexical lifetimes
- Still supports 2015 edition code, fully interoperable

Newly stabilized tools

- Clippy: extra lints for correctness, complexity, and style
- RLS: language server for IDE integration
- Rustfmt: automatic code formatter

And more

- const fn, lifetime elision, procedural macros, ...

HEADLINE FEATURES

Red Hat Enterprise Linux 8

Go

VERSION: 1.11.4

- FIPS certification planned for Red Hat Enterprise Linux 8.1
- Implemented through dynamic loading of OpenSSL by default
- Can be disabled during builds with `` `-tags=no_openssl` ``
- Preliminary support for Go modules
- Experimental support for the WebAssembly target
- Improved debug information: better DWARF information for optimized binaries

HEADLINE FEATURES

Red Hat Enterprise Linux 8

Ancillary tools

binutils 2.30, make 4.2.1, cmake 3.11.4, flex 2.6.1, bison 3.0.4, byacc-1.9.*, ltrace 0.7.91, dwz 0.12, etc.

Binutils 2.30

- Adds new security features to the linker
- Adds support for improved debugging with GDB

Make 4.2.1

- Improved target tracing with new -- trace feature
- Improved error reporting (e.g., exact line number of failure)
- Job server interface is now formalized and documented

HEADLINE FEATURES

Red Hat Enterprise Linux 8

Performance Co-Pilot

VERSION: 4.3

- **Next generation dstat: pcp-dstat(1)**
  - Reports configurable columns of arbitrary system statistics using config files
  - Adds historical and remote host analysis
- **pcp2spark(1)**
  - Sends PCP system metrics to the Apache Spark platform for analytics
- **pcp2elasticsearch(1)**
  - Sends PCP metrics to Elasticsearch for indexing and querying in the ELK stack

HEADLINE FEATURES

Red Hat Enterprise Linux 8

Performance Co-Pilot

VERSION: 4.3

- **New metrics**
  - Added *podman* container metrics into PCP
  - Access PCP metrics from eBPF scripts
  - Added v10 PostgreSQL server metrics into PCP
- **Vector web app adds heatmap and bcc support**
  - New heatmap visualization added to getvector.io
  - New eBPF metric visualizations added
- **Disk space savings allowing finer-grained metric recording**
  - Transparent, multivolume archive compression

HEADLINE FEATURES

Red Hat Enterprise Linux 8

SystemTap

VERSION: 4.0

- Prometheus exporter service, demos
- eBPF back-end support, including string data types
- Scripts are now more future-proof when working with syscalls

Other features since Red Hat Enterprise Linux 7.2

- `--monitor` and `--interactive` modes
- Scripts can take console input

HEADLINE FEATURES

Red Hat Enterprise Linux 8

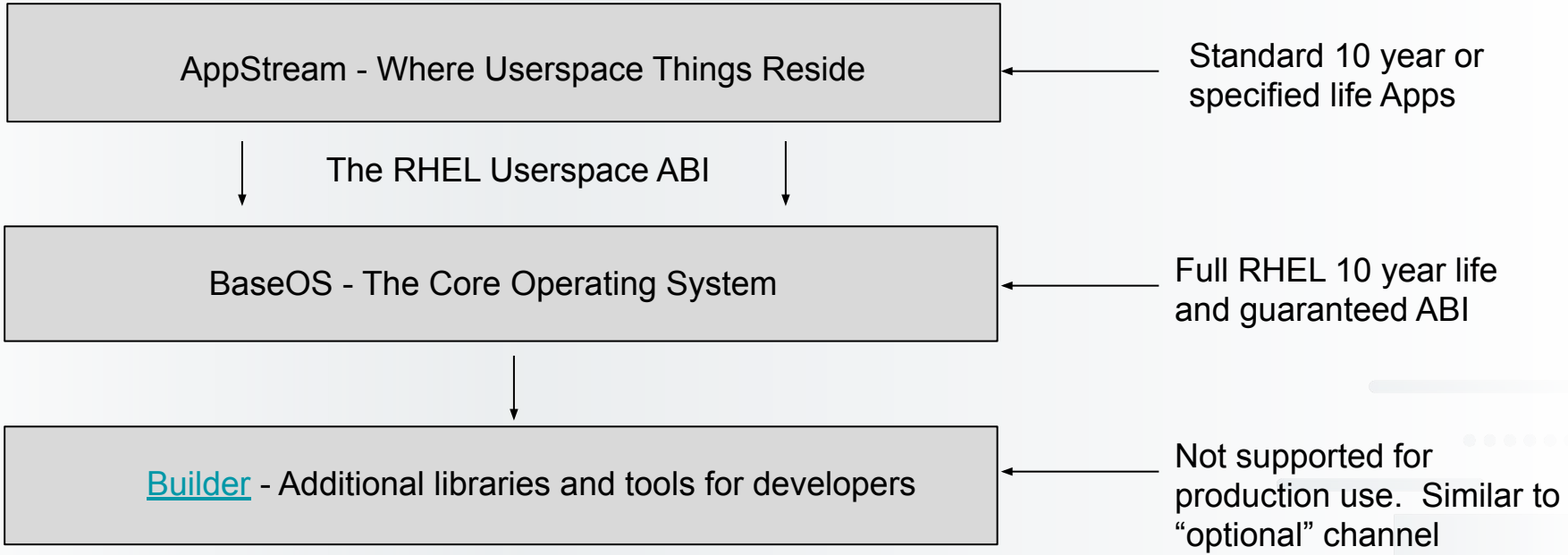
Valgrind

VERSION: 3.14

- IBM s390x (“System Z”) vector support
- Improved suppression specifications
- Faster performance

What is CodeReady Linux Builder?

For Developers: CodeReady Linux Builder



Available with Developer Subs, NFRs, and all RHEL Production Subs.

CodeReady Linux **Builder** Use Cases

Red Hat Enterprise Linux 8

CODE READY LINUX BUILDER

Web Developer, PHP, Ruby, and Perl

- PHP packages, Ruby gems, and Perl modules are provided in AppStream
- Ruby and Perl both have additional libraries made available in the **Builder** repository
- However they are less commonly used or used at build time only

Java Developers

- The functionality and jars you would expect to use normally have been provided in the AppStream
- Ant, maven and apache-commons-logging can be found in AppStream
- If you need some of the build-only components, those are in the **Builder** repository

CodeReady Linux **Builder** Use Cases

Red Hat Enterprise Linux 8

CODE READY LINUX BUILDER

.NET Developers

- Core Runtime & tools are in AppStream as the “dotnet” package
- As a .Net developer, you will not need the **Builder** repository

Go and Rust Developers

- Go and Rust statically link their own runtimes
- If you use one of these languages, you won’t need the **Builder** repository

CodeReady Linux Builder Use Cases

Red Hat Enterprise Linux 8

CODE READY LINUX BUILDER

C/C++ and Fortran

- Core libraries (e.g. glibc, libstdc++) are in BaseOS
- Compilers are provided directly in AppStream with tools to support development
- Many of the *header files*, *devel packages*, *etc.* are found in the **Builder** repository
- You'll likely want to have **Builder** enabled on build machines
- You should not, normally, need the repository enabled on your runtime deployments

Packaging and Deployment Tools

- e.g. meson, dejagnu, and doxygen
- Also found in the **Builder** repository

DEEPER DIVE

DEEPER DIVE

TOPICS

- C++ Language Variants and ABI
- Performance builds, analysis and tuning
- Security features in RHEL 8 Tools
- Accelerating your own development

DEEPER DIVE

C++11 and ABI in RHEL7

- **RHEL7 Developer Toolset (DTS) supports C++11**
- **Language variant is distinct from ABI**
  - GCC upstream `std::list` and `std::string` implementation didn't satisfy algorithmic complexity requirements introduced in C++11
  - RHEL7 and RHEL7-based DTS GCCs inherited that initial implementation from upstream
  - GCC ABI changed upstream in GCC5
  - RHEL7 Developer Toolsets introduced and continued support for C++11
  - But the standard RHEL C++ library didn't change ABI
  - Compile time flag in DTS GCCs (`-D_GLIBCXX_USE_CXX11_ABI=0` default)

DEEPER DIVE

C++11 and ABI in RHEL8

RHEL8 C++11 language support

- Full support for C++11 and C++14
- Experimental C++17 support
- Preliminary support for a few C++2a features

RHEL8 C++11 ABI support

- Supports both the old and new ABI
  - Can be changed at compile-time
  - Set `-D GLIBCXX USE CXX11 ABI=0`or `-D_GLIBCXX_USE_CXX11_ABI=1`
- GCC defaults to the new ABI
- True for later GCC releases for RHEL8 via Developer Toolset-like releases

DEEPER DIVE

C++11 and ABI in RHEL8

- **Binaries built with Developer Toolset on RHEL 7:**
  - Will work on RHEL 8 if it only depends on libstdc++.so
  - May not work if they depend on other system libraries
  - Mixing interfaces using the old/new ABI will lead to problems and is explicitly not supported [e.g. binary uses new ABI and library uses old ABI]
  - Specifically, interfaces between objects using `std::list`, `std::string`, etc.
  - Can cause silent issues as well as build-time or runtime linker errors
  - The two implementations can co-exist within one binary because the mangled names are different

DEEPER DIVE

C++ ABI: Guidance

- **Upshot:**
  - If you rebuild some of your project on RHEL8, you should rebuild all of it rather than try to link with code compiled on RHEL7
  - As usual, forwards support only:
  - Build on RHEL8 and run on the same of later RHEL8, not RHEL7
  - More information: https://developers.redhat.com/blog/2015/02/05/gcc5-and-the-c11-abi/

DEEPER DIVE

Building for performance

- **`-O2` or `-O3` are generally good places to start**
  - `-flto` is often beneficial, but can make debugging much more difficult
  - `-march=native -mcpu=native`
    - Can help too
    - But assumes the result always runs on the same hardware as its build
  - PGO almost always helps if you have good training data
- **Domain specific extensions including OpenMP**

DEEPER DIVE

Performance Analysis

- **Generating execution profiles via compile-time instrumentation**
  - Use `-fprofile-generate` to enable runtime gathering of profiling data
  - Use `-fprofile-use` to exploit the profiling data to improve optimization
- **Using perftools like SystemTap to gather additional data**
  - **Useful reading:**
    - https://access.redhat.com/articles/17839 (using systemtap kbase)
    - https://access.redhat.com/articles/767563 (performance cookbook)
    - https://sourceware.org/systemtap/examples/ (systemtap examples)

DEEPER DIVE

Performance Analysis

- **glibc tunable options**
  - Customize behavioral aspects of the C library (glibc)
  - Documented in the "Tunables" chapter of the manual
  - `$ info libc Tunables`
  - Explained for each glibc subsystem

DEEPER DIVE

Performance Analysis

- **glibc tunable options: Simple example**
  - Disable thread-local cache and allow only 1 arena in malloc (old RHEL 6 behavior):
    - `GLIBC_TUNABLES=glibc.malloc.tcache_count=0:glibc.malloc.arena.max=1`
    - `export GLIBC_TUNABLES`

DEEPER DIVE

Security Features in RHEL 8 Tools

- RHEL 8's own executables and libraries hardened against Stack Clash
- `-fstack-clash-protection` to harden your own code against Stack Clash
- Many new security focused warnings in GCC to detect buffer overflows, out of bounds array indexing, unterminated character strings, etc
- Automatic annotation of system binaries and executables for later examination of their security profile (annobin)

DEEPER DIVE

Spectre/Meltdown: Security vs Performance

- **Retpolines**
  - No need to do anything to mitigate Spectre style attacks for user space code
  - Retpolines are, in fact, harmful for user space code
- **Kernel modules**
  - Will automatically pick up appropriate mitigations if they use the kernel kbuild system
- **Userspace builds**
  - These should use stack-clash protection, PIE, stack protector, etc.
  - Performance impacts here are minimal

DEEPER DIVE

glibc enhancements

- **Improved security** in `abort()` processing
  - Terminate program as quickly as possible to avoid execution of malicious code
- `LD_LIBRARY_PATH`, and `LD_POINTER_GUARD` ignored for `AT_SECURE`
- Corrective handling of **ELF dynamic string tokens** in all forms of input in the binary
  - Particularly for `AT_SECURE`
- IDNA implementation uses system `libidn2` to **improve security**
- DNS **stub resolver** limits advertised UDP buffer size to 1200 bytes
  - Avoids **fragmentation-based spoofing** attacks
- Removal of most `alloca` and **VLAs** from glibc

Accelerate Your Own Development

Red Hat Enterprise Linux 8

GCC's diagnostic subsystem

- Now tracks source locations in terms of ranges of source, rather than points
- Makes it clearer exactly where problems are by underlining relevant subexpressions
- Can emit fix-it hints, suggesting to the user how to fix a problem
- Eclipse CDT is able to auto-apply such suggestions
- **Red Hat developers contributed these improvements to the upstream GCC project**

Numerous usability improvements to GCC vs RHEL7

- https://developers.redhat.com/blog/2018/03/15/gcc-8-usability-improvements/

Example: typo in field name

RHEL 8's gcc 8 suggests corrections for misspelled field names

```
[demo-user@rhel8 ~]$ g++ -c typo.cc
typo.cc: In function 'bool predicate(const item*)':
typo.cc:8:19: error: 'const struct item' has no member named 'intensty'; did you
 mean 'intensity'?
   return emitter->intensty > 0.5;
                  ^~~~~~~~
                  intensity
[demo-user@rhel8 ~]$
```

Example: parameter type mismatch

RHEL 7's gcc 4.8 is imprecise when reporting the location of the problem

```
[demo-user@rhel7 ~]$ gcc --version
gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-36)
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

[demo-user@rhel7 ~]$ gcc -c test.c
test.c: In function 'caller':
test.c:5:3: warning: passing argument 2 of 'callee' makes pointer from integer w
ithout a cast [enabled by default]
   return callee(first, second, third);
   ^
test.c:1:12: note: expected 'const char *' but argument is of type 'int'
 extern int callee(int one, const char *two, float three);
            ^
[demo-user@rhel7 ~]$
```

Example: parameter type mismatch

RHEL 8's gcc 8 uses underlining to show exactly where the problem is

```
[demo-user@rhel8 ~]$ gcc --version
gcc (GCC) 8.2.1 20180905 (Red Hat 8.2.1-3)
Copyright (C) 2018 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

[demo-user@rhel8 ~]$ gcc -c test.c
test.c: In function 'caller':
test.c:5:24: warning: passing argument 2 of 'callee' makes pointer from integer
without a cast [-Wint-conversion]
   return callee(first, second, third);
                        ^~~~~~
test.c:1:40: note: expected 'const char *' but argument is of type 'int'
 extern int callee(int one, const char *two, float three);
                            ~~~~~~~~~~~~^~~
[demo-user@rhel8 ~]$
```

Example: hints for accessing private fields

RHEL 8's gcc 8 can suggest accessors (e.g. when refactoring C++ code)

```
[demo-user@rhel8 ~]$ g++ -c accessor.cc
accessor.cc: In function 'double test(shape&, double)':
accessor.cc:12:12: error: 'double shape::m_scale' is private within this context
   return s.m_scale * prop;
            ^~~~~~~
accessor.cc:6:10: note: declared private here
   double m_scale;
          ^~~~~~~
accessor.cc:12:12: note: field 'double shape::m_scale' can be accessed via 'doub
le shape::get_scale() const'
   return s.m_scale * prop;
            ^~~~~~~
            get_scale()
[demo-user@rhel8 ~]$
```

DEEPER DIVE

Further Reading: GDB

- Write a **custom GDB Python pretty-printer** to simplify display of complex objects:
  - https://developers.redhat.com/blog/2017/11/10/gdb-python-api/
- Take advantage of GDB's **newly enhanced C++ breakpoint scope/namespace wild matching**:
  - https://sourceware.org/ml/gdb-patches/2017-06/msg00012.html

```
(gdb) b method[TAB]
A::method()
B::A::method()
```

```
(gdb) b method
Breakpoint 1 at method. (2 locations)
```

- **Debug your OpenShift container** via GDBserver
  - https://developers.redhat.com/blog/2015/04/28/remote-debugging-with-gdb/#more-415434

```
(gdb) target extended-remote | oc exec -i $POD -- gdbserver --multi -
```

SLIDES LINK REMINDER:
bit.ly/rhel8tools

Q&A

RED HAT SUMMIT

THANK YOU


linkedin.com/company/Red-Hat


facebook.com/RedHatinc

youtube.com/user/RedHatVideos


twitter.com/RedHat