Leveraging the compiler for better code

Mihai Patrascoiu
CERN – IT Storage

Topics

- Address Sanitizer
- Experimenting with different compiler builds
- Own experience on EOS codebase

Address Sanitizer

- Open-Source tool developed by Google to detect memory corruption bugs
- Instruments code at compile-time
- Runtime library to replace memory allocation
- Helps detect stack and/or heap buffer overflows, use-after-free, memory leaks
- Average slow-down ~2x

https://github.com/google/sanitizers/

Using Address Sanitizer?

Compile time:

```
g++ -o program program.cpp -fsanitize=address (-lasan)
```

Runtime:

```
./program
```

CMake:

```
list(APPEND CMAKE_CXX_FLAGS “-fsanitize=address”)
```

Why Address Sanitizer? – Example #1

```
char* toUpperCpy(const char* source,
                 int len) {
  char* dest = (char*) malloc(len);

  for (int i = 0; i < len; i++) {
    dest[i] = source[i];

    if (dest[i] >= 'a' && dest[i] <= 'z') {
      dest[i] -= 32;
    }
  }

  dest[len] = ‘\0’;

  return dest;
}
```

```
int main() {
  char initial[100] = "Hello World!";

  printf("%s size=%d\n",
          initial, strlen(initial));

  char *upper =
          toUpperCpy(initial, strlen(initial));
  printf("%s\n", upper);

  return 0;
}
```

Why Address Sanitizer? – Example #1

Why Address Sanitizer? – Example #2

```
std::string toUpperStr(const char* source) {
  std::string upper = source;

  for (char& c: upper) {
    if (c >= 'a' && c <= 'z') {
      c -= 32;
    }
  }

  return upper;
}
```

```
int main() {
  char initial[100] = "Hello World!";

  printf("%s size=%d\n",
          initial, strlen(initial));

  std::string upperStr =
          toUpperStr(initial);
  printf("%s\n", upperStr);

  return 0;
}
```

Why Address Sanitizer? – Example #2

ASAN – ✓

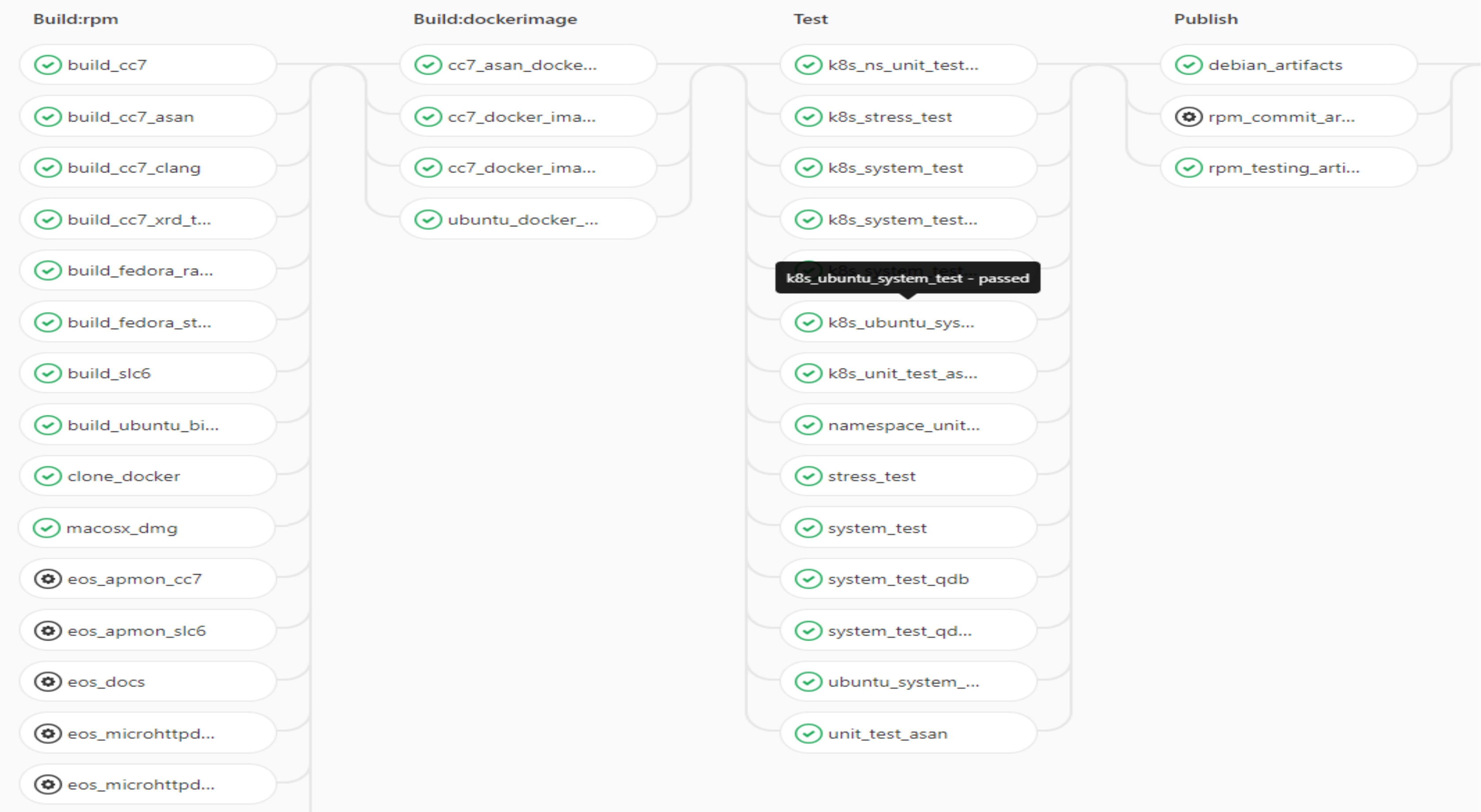
Result – ❗

Bottom line: *No silver bullet*

What about Clang?

Try out different compilers, compiler versions, build environments, etc.!

EOS Codebase builds



Build:rpm

- ✓ build_cc7
- ✓ build_cc7_asan
- ✓ build_cc7_clang
- ✓ build_cc7_xrd_t...
- ✓ build_fedora_ra...
- ✓ build_fedora_st...
- ✓ build_slc6
- ✓ build_ubuntu_bi...
- ✓ clone_docker
- ✓ macosx_dmg
- ⚙ eos_apmon_cc7
- ⚙ eos_apmon_slc6
- ⚙ eos_docs
- ⚙ eos_microhttpd...
- ⚙ eos_microhttpd...

Build:dockerimage

- ✓ cc7_asan_docke...
- ✓ cc7_docker_ima...
- ✓ cc7_docker_ima...
- ✓ ubuntu_docker_...

Test

- ✓ k8s_ns_unit_test...
- ✓ k8s_stress_test
- ✓ k8s_system_test
- ✓ k8s_system_test...
- k8s_ubuntu_system_test - passed
- ✓ k8s_ubuntu_sys...
- ✓ k8s_unit_test_as...
- ✓ namespace_unit...
- ✓ stress_test
- ✓ system_test
- ✓ system_test_qdb
- ✓ system_test_qd...
- ✓ ubuntu_system_...
- ✓ unit_test_asan

Publish

- ✓ debian_artifacts
- ⚙ rpm_commit_ar...
- ✓ rpm_testing_arti...

Findings on EOS codebase

Address Sanitizer:

- discovered linking problems between static and shared libraries on unit tests executables

Clang :

- incorrect uses of variadic arguments
- ambiguous std::move statements

Conclusion

- Address Sanitizer – very powerful tool for detecting memory bugs
- Experiment different compilers, compiler versions, build environments
- Remember: there is no silver bullet