Spack for Tools

What can Spack do for tools?

- Todd Gamblin
- Tim Haines
- Nate Hanford
- Rashawn Knapp
- Jim Kupsch
- Matt Legendre
- Martin Schulz

What can Spack do to help tools integrate with large software stacks?

- How should tool packages be modeled in Spack?
- How do tools’ relationships with packages they analyze/instrument differ from existing dependency types?
- What aspects of the build can Spack let tools control?
  - Compiler wrappers?
    - Flags?
    - Inject tools?
  - Build environment?

We came up with three usage scenarios

1. Basic instrumentation tools
   - Produce an instrumented version of some package

2. Preload tools
   - Ensure that compatible preloaded/injected libraries are available for some runtime environment

3. Analysis tools
   - Provide access to source/build/installation
   - Provide location to store output

Basic instrumentation

Example: score-p

spack install lammps ^tool

- lammps doesn’t declare a dependency on tool

- Tool declares itself as a tool package
  1. Declares what dependency type(s) it should be when injected
     - Probably some variant of a build dependency
     - Maybe link dependencies

  2. Injects self into Spack’s compiler wrappers in build env

  3. Produces a lammps build with a unique hash
     - Includes ^tool dependency in hashed metadata

Preload tools

Example: openspeedshop

- Build a version of some package that’s first-party (runtime/same process) compatible with another
  - spack install openspeedshop-runtime ::lammps
    - Make sure the runtime is compatible with this particular lammps
    - Lammps side of this matches against installations by default and builds openspeedshop runtimes per installation or per configuration
    - Tool frontend/backend
- Build openspeedshop-runtime in all configurations that make sense in an environment
  - Env has several MPI’s, C++ compilers, etc.
  - Need OSS runtimes per MPI per compiler, per some dep
    - Package says granularity of runtime peer dependencies needed
  - Combinatorial dependencies on peer packages
    - depends_on(“runtime”, <specify per what>, type=“peer”)
    - Per what == c x mpi (virtuals defining the matrix)

Tools that analyze source or binaries and produce output as an installation

Example: cppcheck

- Tool doesn't necessarily produce a new build
  - It produces analysis of tool run on that package

- Spack needs to:
  - give tools output installation directory
  - come up with output name and hash based on metadata
  - Peer deps on analysis inputs

- Need a syntax for doing a run that produces this output
  - These are bad:
    - spack install cppcheck-output ^lammps   # run cppcheck as part of the build
      - spack test lammps ^cppcheck   ??
    - spack install binary-analyzer-output ^lammps   # run

- Tools need to run at different times – provide hooks for these in tool package:
  - Source stage time
  - Configure time
  - Build time
  - After installation

Overrides in tooled packages

- Instrumented/analyzed packages could provide overrides for certain analyzer tools
    - Skip compiler instrumentation of this one C++ file
    - Restrict set of cpp directives that can be used by analyzers?
    - Provide own configurations for cppcheck, clang static analyzers?

- Could also have an option to ignore overrides
    - In case the tool REALLY wants to instrument everything

Hooks in tool packages

- To say how package should be shimmed into builds
  - Is it a run dep, link dep, etc.?

- How packages should use compiler wrappers
  - Strip args?
  - Translate args?
  - Add args?
  - Inject instrumenting compiler?

- What versions of peer deps should be built for a particular env?
  - Inject a guaranteed compatible runtime library as a link dependency
  - mpip as an example

- Need hooks before and after concretization
  - Injecting flags – before concretization
  - How to inject link/run deps into a package before concretization?
    - Match on particular packages that you want to “become” a dependency of

Discoverability of tools

- Could we tag applications as tools in Spack?
    - Can already tag packages with ids, but no great browser interface for that
    - Let users search specifically for tool packages
    - See syntax for using a tool with some package

- Could we use this to do comparative analysis of different tools?
    - Find all the instrumentation tools, compare what they do
    - Hard to generalize this.