Streamlining continuous integration/deployment of bespoke software environments and adhering to the FAIR principles and workflows

Tommy Gatti[1], Aidan Heerdegen[1], Harshula Jayasuriya[1], Paul Leopardi[1], Scott Wales[2], Rui Yang[3]

[1]Australia's Climate Simulator (ACCESS-NRI), Australia, [2]Bureau of Meteorology, Australia,
[3]National Computational Infrastructure (NCI), Australia

Australia's climate simulator

The Bureau of Meteorology

NCI AUSTRALIA

ACCESS National Research Infrastructure

Spack: A build from source package manager targeting HPC

spack.io

Build from source	• Build **exactly** what is required
Package manager	• Programmatically orchestrate builds • Abstract dependencies concretized • Automatically build and locate dependencies • Install multiple builds in isolated environments • Build provenance • Build reproducibility
Targeting HPC	• Operates within existing system • Use HPC components (MPI libraries, compilers)

Spack: A build from source package manager targeting HPC

spack.io

Low marginal cost building and deploying software

⟶ Can build and deploy many versions and dependencies

Cross platform by design

⟶ Enables replicability (HPC, GitHub Runners, EC2 AMD & ARM)

Generic programmable builds

⟶ Plug n' play tools possible

- Created Reusable Compilation CI GitHub Action
- Created Reusable Deployment GitHub Action

Free and open-source software written in Python

⟶ Popular, well known and extensible

Spack Configuration Files

Why these files matter

- Customises how Spack installs and builds packages for your environment.
- Facilitates consistent builds across multiple systems and users.

Where to find and edit configuration files

- Typically found in `~/.spack/`
- Custom configurations can be added globally or for specific environments, e.g. `$SPACK_ROOT/etc/spack/`

https://spack.readthedocs.io/en/latest/configuration.html

https://access-hive.org.au/getting_started/spack/

Spack Configuration Files (cont)

compilers.yaml:

- **Defines available compilers and their paths.**
- **Spack can find compilers (`$ spack compiler find`) and update the config file, but this file can be customized to specify compilers manually, e.g.**

```
- compiler:
    spec: intel@=2021.10.0
    paths:
      cc: /apps/intel-ct/wrapper/icc
      cxx: /apps/intel-ct/wrapper/icpc
      f77: /apps/intel-ct/wrapper/ifort
      fc: /apps/intel-ct/wrapper/ifort
    flags: {}
    operating_system: rocky8
    target: x86_64
    modules: [intel-compiler/2021.10.0]
    environment: {}
    extra_rpaths: []
```

Spack Configuration Files (cont)

packages.yaml:

- Defines preferred versions, variants, and options for specific packages, e.g.

```
packages:
  all:
    providers:
      # Completely ignoring higher-level configuration options
      # is supported with the :: notation for keys ...
      mpi:: [openmpi]
  openmpi:
    externals:
    - spec: openmpi@4.1.5 %intel
      prefix: /apps/openmpi/4.1.5
      modules: [openmpi/4.1.5]
      extra_attributes:
        environment:
          prepend_path:
            CMAKE_PREFIX_PATH: /apps/openmpi/4.1.5/include/Intel
    buildable: false
```

Spack Configuration Files (cont)

concretizer.yaml

- Very important to understand `granularity` and `unify`.

modules.yaml:

- Defines how Spack generates module files (e.g., Lmod or Environment Modules).

repos.yaml:

- Add a repository containing recipes that are not in Spack.

Containers

- Containerise the model environment for simple porting and testing
- Install Spack and Conda environments in an Apptainer container – all dependencies for a model, easy to get started. Also reduces file count on Lustre filesystems
- The same environment can be installed outside a container to use optimised MPI, but may require extra site configuration
- Environments have model dependencies, not the models themselves, to enable model development
- Environment has been tested at NCI, AWS and being set up at NCMRWF
- https://github.com/MetOffice/ngmo-environments

Momentum®

The Unified Earth Environment Prediction Framework

International Collaboration:

Met Office (UK), Bureau of Meteorology (AU), CSRIO (AU), NCMRWF (India), MSS (Singapore), NIWA (NZ)

Lots of HPC systems to support!

Creating a Spack Recipe

What is a Spack Recipe?

- Two types of recipes: Package and Bundle
- Package: A Python file defining how to download, configure, build, and install a package
- Bundle: A virtual package with dependencies and no source code to download.

Basic Package Recipe Structure

- Class definition
- Package URL, versions, checksum
- Dependencies
- Variants
- Build Process

Example Spack Package Recipe: fiat

```
# Copyright 2013-2024 Lawrence Livermore National Security, LLC and other
# Spack Project Developers. See the top-level COPYRIGHT file for details.
#
# SPDX-License-Identifier: (Apache-2.0 OR MIT)

# Based on spack/var/spack/repos/builtin/packages/fiat/package.py

from spack.package import *

class Fiat(CMakePackage):
    """FIAT (Fortran IFS and Arpege Toolkit) is a collection of selected
    Fortran utility libraries, extracted from the IFS/Arpege model."""

    homepage = "https://github.com/ecmwf-ifs/fiat"
    git = "https://github.com/ACCESS-NRI/fiat.git"
    url = "https://github.com/ecmwf-ifs/fiat/archive/1.0.0.tar.gz"

    maintainers("climbfuji", "penguian")

    license("Apache-2.0")
```

Define the package:
Class, homepage, URL, maintainers

Example Spack Package Recipe: fiat (cont)

```
    version("main", branch="main", no_cache=True)
    version("um", branch="um", no_cache=True)
    version("1.2.0", sha256="...")
    version("1.1.0", sha256="...")
    version("1.0.0", sha256="...")

    variant(
        "build_type",
        default="RelWithDebInfo",
        description="CMake build type",
        values=("Debug", "Release", "RelWithDebInfo"),
    )

    variant("mpi", default=True, description="Use MPI")
    variant("openmp", default=True, description="Use OpenMP")
    variant("fckit", default=True, description="Use fckit")
```

List versions and variants

Example Spack Package Recipe: fiat (cont)

```
    depends_on("ecbuild", type=("build"))
    depends_on("mpi", when="+mpi")
    depends_on("eckit", when="+fckit")
    depends_on("fckit", when="+fckit")

    patch("intel_warnings_v110.patch", when="@0:1.1.0")
    patch("intel_warnings_v120.patch", when="@1.2.0:")
```

List dependencies and patches

Example Spack Package Recipe: fiat (cont)

```
def cmake_args(self):
    args = [
        self.define_from_variant("ENABLE_OMP", "openmp"),
        self.define_from_variant("ENABLE_MPI", "mpi"),
        self.define_from_variant("ENABLE_FCKIT", "fckit"),
    ]
    if "+mpi" in self.spec:
        args.extend(
            [
                self.define("CMAKE_C_COMPILER", self.spec["mpi"].mpicc),
                self.define("CMAKE_CXX_COMPILER", self.spec["mpi"].mpicxx),
                self.define("CMAKE_Fortran_COMPILER", self.spec["mpi"].mpifc),
            ]
        )
    return args
```

Configure the build:
cmake, make, or other build commands

Spack Recipe Repositories

ACCESS-NRI

- https://github.com/ACCESS-NRI/spack-packages/

Bureau of Meteorology

- https://github.com/ScottWales/spack-environments/tree/master/repos/bom-ngm

NCI

- https://github.com/nci/spack-repo

UK Met Office

- https://github.com/MetOffice/ngmo-environments

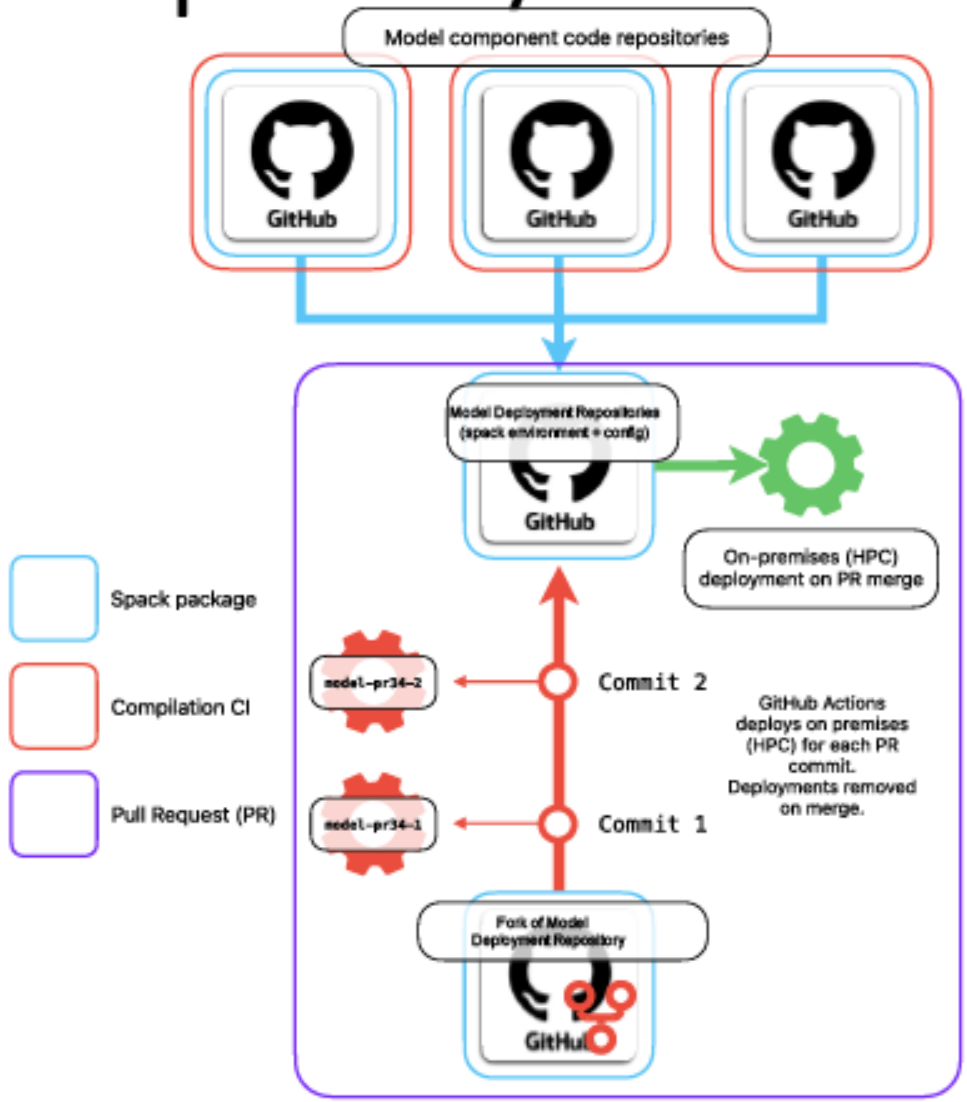
Continuous Integration/Deployment (CI/CD)

- Overview of CI/CD + Spack
- Deployment Pipeline
- The `spack.yaml` File
- The Why – Thoughts on integrating Spack into CI/CD

Overview – Spack CI/CD

- At ACCESS-NRI, our climate model deployment pipeline combines GitHub Actions and Spack to get models built and ready to use on supercomputers
- At its core, we have *model deployment repositories* that houses information on:
  - How to build the model via Spack (using `spack.yaml`)
  - What versions of `spack`, `spack-packages` and `spack-config` are used
  - Current Releases/Prereleases and where they are located
- We pass this information on to the supercomputers Spack instances for install
- Examples: ACCESS-NRI/ACCESS-OM2, ACCESS-NRI/ACCESS-ESM1.5

Overview – Spack CI/CD



ACCESS Model Update Deployment Workflow

The spack.yaml File

- The core of the CI/CD workflow is the `spack.yaml` file, a set of constraints that aids Spack in finding the best way to resolve dependencies
- Aspects that we have seen earlier (packages, config) can be overridden here because it is in its own independent environment

```
spack:
  specs:
    - model@git.2024.10.0
  packages:
    model-component:
      require: '@git.2024.03.10 +variant'
    all:
      require: '%intel@19.0.5.281 target=x86_64'
  config:
    source_cache: $spack/../some/source_cache
```

- When the file is installed via Spack, we also get a `spack.lock` file that captures the concretized result – giving full build provenance

Spack – The Why

Why use Spack for CI/CD, as an infrastructure maintainer?

Easy

- `spack.yaml` is the single source for versioning, dependencies and configuration, and Spack handles the installation

Portable

- Spack can be target-independent – container, supercomputers, local machine…

Provenance

- `spack.lock` file gives a full provenance chain, allows effortless rebuilds

Further information

- Documentation, e.g. "Basic Usage"https://forum.access-hive.org.au/t/building-a-spack-community-in-australia/3833 https://spack.readthedocs.io/en/latest/basic_usage.html
- Tutorials, e.g "Tutorial: Spack 101" https://spack-tutorial.readthedocs.io/en/latest/
- Communication -- Slack (https://spackpm.slack.com/) and ~fortnightly meetings
- Github -- Issues and pull requests (https://github.com/spack/spack) e.g. https://github.com/spack/spack/issues/42700 and https://github.com/spack/spack/pull/44614
- Find out more and collaborate by going to:

https://forum.access-hive.org.au/t/building-a-spack-community-in-australia/3833

The Bureau of Meteorology

NCI AUSTRALIA

Australia's climate simulator