

Guix and Spack for Application Deployment Across Supercomputers

The NumPEX PC5 WP3 Team

Index

- Context and problems to solve
- Package managers: Guix and Spack
- Containers: Singularity
- Performance considerations
 - MPI
 - CUDA
 - ROCM

NumPEX WP3 (Exa-DI) Team

- Lead: Bruno Raffin, INRIA; Benoit Martin, CEA
- Part time participants: Ludovic Courtès, INRIA; Pierre Neyron, CNRS; Julien Bigot, CEA
- Full-time dedicated engineers:
 - Romain Garbage, Bordeaux
 - Fernando Ayats, Grenoble

Regular contacts with national compute centers (Idris, Cines and CCRT)

Context

Challenge:

- Exascale apps are increasingly **difficult to build, deploy, maintain or test**. The complexity of apps grows, as well as the machines' complexity. Current software deployment doesn't scale properly with this complexity.

At stake:

- Need for **HPC DevOps tools and methodologies**, that enhance productivity, interoperability and portability.

Our Target

Application developers

- Ease difficulty in building and developing apps.
- Portable solution from laptops to the supercomputers.
- CI/CD-ready

System administrators

- Ease package administration and testing.
- Introduce Guix in compute centers.

Application users

- Provide turn-key solutions for deployments.
- Fearless migration between machines and updates.

Guix and Spack and the Problems They Solve

Classical Way to Deploy Software

- Manually installing libraries (git clone, cmake, make install, etc)
 - ✗ Time-consuming
 - ✗ Error-prone
 - ✗ Automation scripts are fragile

```
#!/bin/sh  
tar -xvf vendor/mylibrary.tar.gz  
cd mylibrary && make ...
```

Classical Way to Deploy Software

- *module load <packname>*
 - + Cleaner solution than manually installing things
 - ✗ Modules are **specific to each cluster**, and different between each other
 - ✗ Not reproducible, both in the future or with colleagues
 - ✗ Limited to packages and versions provided by the admin team

```
#!/bin/sh
```

```
module load cuda
```

```
module load kokkos
```


Use a Package Manager!

```
#!/bin/sh
```

```
tar -xvf vendor/mylibrary.tar.gz
```

```
cd mylibrary && make ...
```

=>

```
$ guix install mylibrary
```

Use a Package Manager!

- Package managers take the task of managing your dependencies for you
 - + **Easy** installation of dependencies
 - + **Reproducible** stack of software, both in the future and with other contributors
 - + CI/CD Ready



NumPEx Strategy with Package Managers

There are many package managers, but Guix and Spack are HPC-ready:



GNU Guix (<https://hpc.guix.info>)

- Pure dependency tree, doesn't use system libraries
- Easier usage and creation of containers



Spack (<https://spack.io>)

- Can load system libraries, providing some flexibility
- Provides more package versions and variants

Installation of Guix and Spack



Guix installation:

Running the installation script

https://guix.gnu.org/manual/en/html_node/Binary-Installation.html

(*apt install guix* is **discouraged**)



Spack installation:

Cloning the Spack repo and sourcing the setup script,

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

Running Example: Chameleon

Chameleon (<https://project.inria.fr/chameleon/>)

A dense linear algebra software for heterogeneous architectures, developed at Inria.

Requires:

- MPI
 - StarPU
 - CUDA
-
- Packaged in [Guix](#) and [Spack](#)

Examples of Package Managers

```
$ guix search chameleon
```

```
$ spack list chameleon
```

Searching for packages

```
$ guix install chameleon
```

```
$ spack install chameleon
```

Installing packages

```
$ guix shell -m ./manifest.scm
```

```
$ spack env activate ./myenv
```

Activating environments

Examples of Guix

`$ guix install cmake gcc-toolchain openmpi` Installing packages globally

`$ guix shell cmake gcc-toolchain openmpi` Temporary environment

`$ guix shell -D chameleon` Get the libraries to build chameleon, but not chameleon itself

“Declarative” package management with a manifest.scm (like a requirements.txt)

`$ guix shell cmake gcc-toolchain --export-manifest > manifest.scm`

`$ guix shell -m manifest.scm`

`(specifications->manifest (list "cmake" "openmpi" "gcc-toolchain"))`

Examples of Spack

Installing packages globally

```
$ spack install cmake openmpi
```

Declarative package management with spack.yaml (like requirements.txt)

```
$ spack env create -dir ./myenv && spack activate ./myenv
```

```
$ spack add cmake openmpi
```

spack:

specs:

- **cmake**
- **openmpi**

What is a Package Definition?



definition file



```
$ guix install chameleon  
(reads chameleon.scm file)
```

```
...
```

```
=> /gnu/store/....-chameleon-1.2.0/bin/chameleon
```

Package Definitions in Guix

Uses Scheme

*Right: extract of the
chameleon definition*

```
(define-public chameleon
  (package
    (name "chameleon")
    (version "1.2.0")
    (home-page "https://gitlab.inria.fr/solverstack/chameleon")
    (synopsis "Dense linear algebra solver")
    (description
      "Chameleon is a dense linear algebra solver relying on sequential
      task-based algorithms where sub-tasks of the overall algorithms are submitted
      to a run-time system. Such a system is a layer between the application and
      the hardware which handles the scheduling and the effective execution of
      tasks on the processing units. A run-time system such as StarPU is able to
      manage automatically data transfers between not shared memory
      area (CPUs-GPUs, distributed nodes).")
    (license license:cecill-c)
    (source
      (origin
        (method git-fetch)
        (uri (git-reference
              (url home-page)
              (commit "v1.2.0")
              ;; We need the submodule in 'CMakeModules/morse_cmake'.
              (recursive? #t)))
        (file-name (string-append name "-" version "-checkout"))
        (patches (search-patches "guix-hpc/packages/patches/chameleon-cpp.patch"))
        (sha256
          (base32 "1gcn7061iz2xxb43rpfh52ynwc2227033alj5aw1d753aqyxq378"))
        (modules '((guix build utils)))
        ** Do not install 'confia.log' to avoid retaining a reference to GCC
```

Package Definitions for Spack

Uses Python

Right: extract of the chameleon definition

```
class Chameleon(CMakePackage, CudaPackage):
    """Dense Linear Algebra for Scalable Multi-core Architectures and GPGPUs"""

    homepage = "https://gitlab.inria.fr/solverstack/chameleon"
    url = "https://gitlab.inria.fr/api/v4/projects/616/packages/generic/source/v1.2.0/cham"
    git = "https://gitlab.inria.fr/solverstack/chameleon.git"
    maintainers("fpruvost")

    version("master", branch="master", submodules=True)
    version("1.2.0", sha256="b8988ecbfff19c603ae9f61441653c21bba18d040bee9bb83f7fc9077043e5f")
    version("1.1.0", sha256="e64d0438dfaf5effb3740e53f3ab017d12744b85a138b2ef702a81df55912f")

    depends_on("c", type="build") # generated
    depends_on("cxx", type="build") # generated
    depends_on("fortran", type="build") # generated

    # cmake's specific
    variant("shared", default=True, description="Build chameleon as a shared library")

    # chameleon's specific
    variant(
        "runtime",
        default="starpu",
        description="Runtime support",
        values=("openmp", "starpu"),
        multi=False,
    )
    variant("mpi", default=True, when="runtime=starpu", description="Enable MPI")
    variant("cuda", default=False, when="runtime=starpu", description="Enable CUDA")
    variant(
```

Package Collision

Both Spack, Guix and Nix use hash-based system, in which packages don't collide between each other.

```
$ guix build chameleon  
=> /gnu/store/000000000001-chameleon/bin/chameleon
```

```
$ guix build --tune=cascadelake chameleon  
=> /gnu/store/000000000002-chameleon/bin/chameleon
```

(Simplified)

Package Managers Summary

- ✗ Don't manually install dependencies
- ✗ Not relying on machine-specific modules will be better for reproducibility
- + Use a package manager
- + guix install ... / spack install ...
- + If a package you need is not available, contact us or the Guix / Spack maintainers (they will be happy to help)

Supercomputer Support via Containers

Supercomputer Support

Guix:

- + Provided by Tier-2 supercomputers, and developer's laptops
- X Not available on Tier-0 or Tier-1 (yet)

Spack:

- + Can be installed almost anywhere
- X Will saturate inodes on supercomputers (too many files)

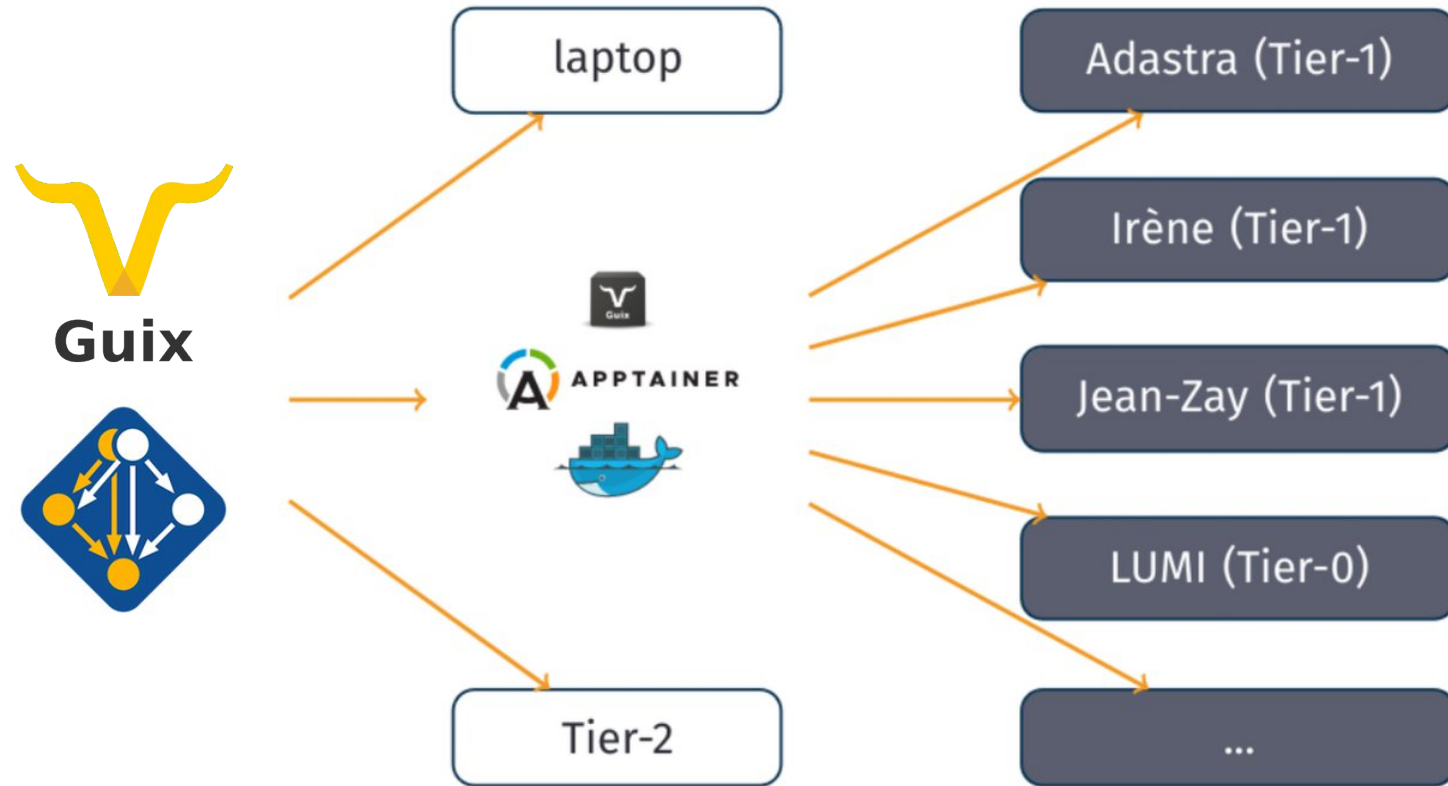
Supercomputer Support

Containers:

- + Available on Tier-0, Tier-1 and Tier-2 supercomputers
- + Generated from Guix and Spack



Supercomputer Support



What is a Container?



```
/usr/bin/cmake  
/lib/x86_64-linux-gnu/libc.so.6  
...
```

- Isolated from the system.
- Near bare-metal performance.
- Easy to use and share.
- **Docker**: most popular
- **Singularity**: HPC-ready, Docker compatible



What is a Container?

Example: official PyTorch containers <https://hub.docker.com/r/pytorch/pytorch>

```
$ python3
```

```
>>> import torch
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
ModuleNotFoundError: No module named 'torch'
```

```
$ singularity shell docker://pytorch/pytorch
```

```
Singularity> python3
```

```
>>> import torch
```

```
(OK!)
```

Building Containers

Singularity containers can be built with their official CLI tool, and definition files:

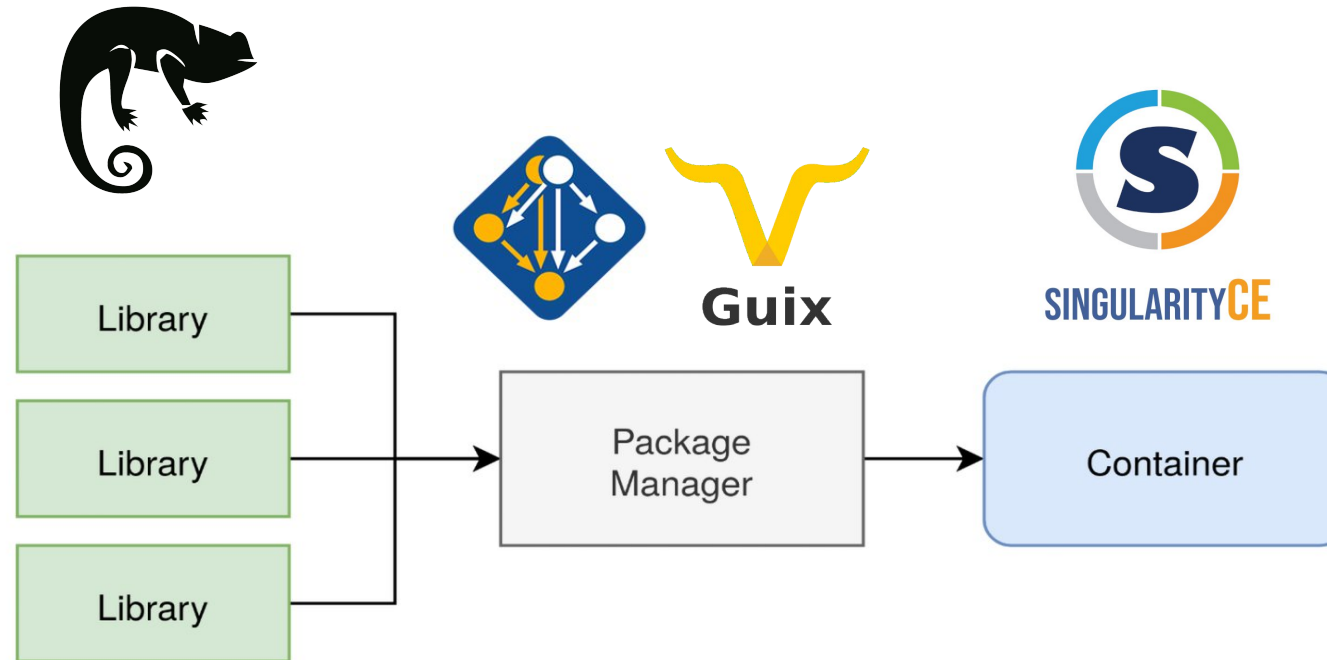
```
# singularity build container.sif definition.def
```

Building containers with Spack and Guix is easy:

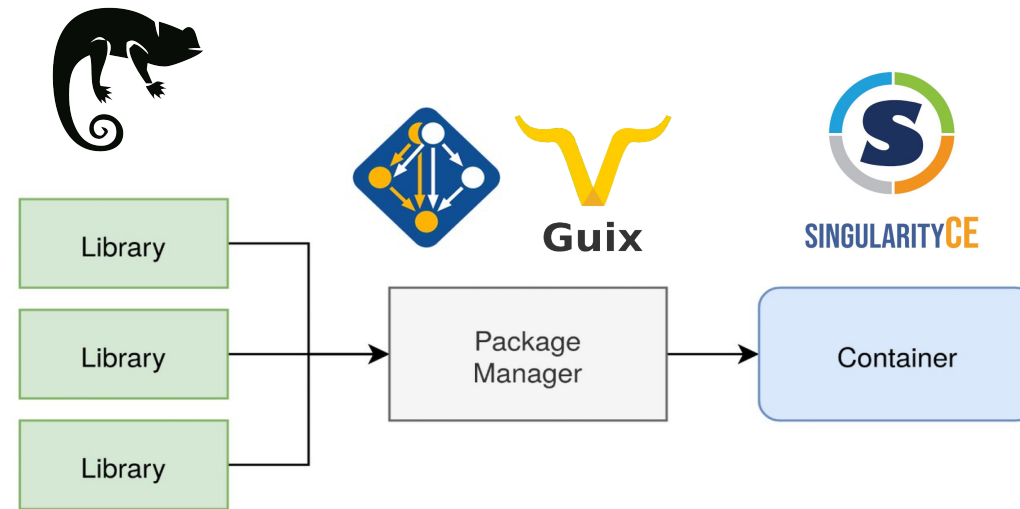
```
$ guix pack -f squashfs chameleon
```

```
$ spack buildcache push <myregistry> chameleon
```

Building Containers with Spack and Guix



Building Containers with Spack and Guix



Why not use APT or DNF to build the container?

- **Reproducible** and optimized software stack.
- Integration with local workflows with Guix and Spack.
- Access to big HPC software catalog.

Building Containers with Guix

```
$ guix pack -f squashfs -r ./image.sif bash chameleon
```

```
$ singularity shell ./image.sif  
Singularity> chameleon_testing ....
```

(Bash is a requirement for Singularity)

Building Containers with Guix

```
$ guix pack -f squashfs -r ./image.sif bash chameleon
```

```
$ scp ./image.sif lumi
```

```
$ ssh lumi
```

```
$ srun -pty <....> \  
    singularity shell ./image.sif
```

```
Singularity> chameleon_testing ....
```


Building Containers with Spack

Pre-requisite: **configure a container registry with Spack:**

- Official documentation:
https://spack.readthedocs.io/en/latest/binary_caches.html
- Tutorial: <https://numpex-pc5.gitlabpages.inria.fr/tutorials/hpc-env/workflow-example/index.html>

```
$ spack buildcache push --base-image ubuntu:24.04 myregistry chameleon
```

```
$ singularity shell docker://myregistry/...chameleon.spack  
Singularity> chameleon_testing ....
```

(The Docker registry is compatible with Singularity)

Building Containers with Spack

```
$ spack buildcache push --base-image ubuntu:24.04 myregistry chameleon
```

```
$ srun -pty <....> \  
    singularity shell docker://myregistry/...chameleon.spack
```

```
Singularity> chameleon_testing ....
```

Container Benefits

In summary:

It's preferred to **use Guix directly** if the supercomputer provides it...

You can **use Spack directly**, if you don't have problems with inodes, and want to use the system's libraries...

But using containers has the benefits of:

- + Easily share the *container.sif* with colleagues, CI/CD, etc.
- + Turn-key solution for developers
- + Reproducible software stack
- + Reproducible experiments for paper authors

Performance Considerations

Support for MPI

```
$ guix install openmpi
```

```
$ spack install openmpi
```

Network drivers:

- For Guix, all included by default (fabric, Cray, etc)
- For Spack, might need to change the flags `spack install openmpi fabrics=ofi`

OpenMPI 4 and 5, and MPICH provided by Guix and Spack

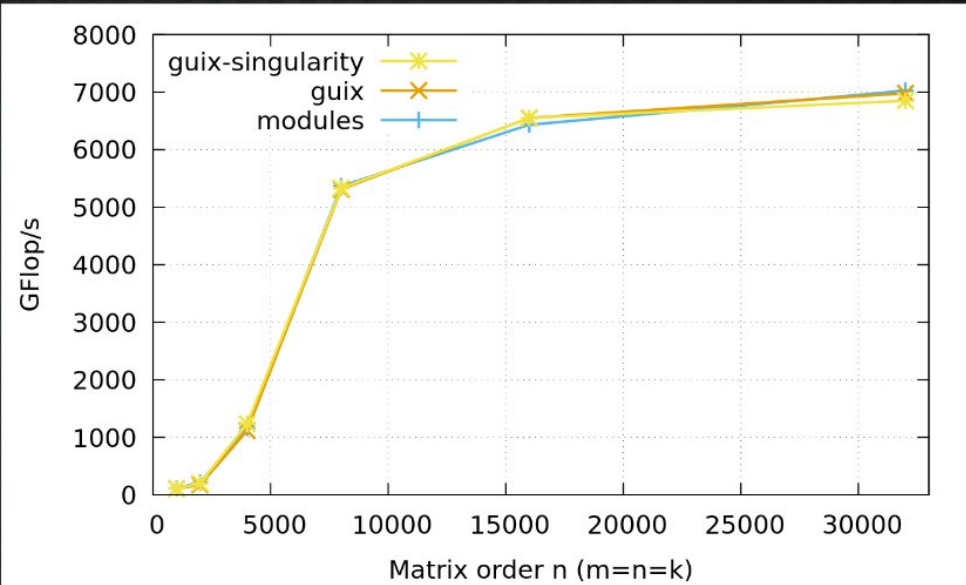
Support for MPI

Two ways of deployment:

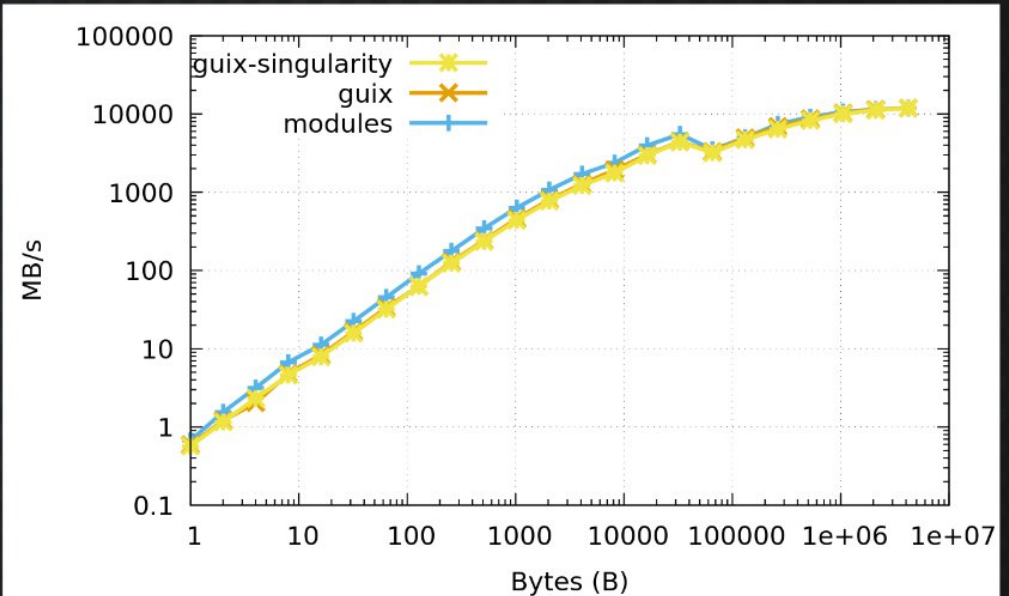
- Bring-Your-Own MPI in the container (**What we do**)
- Load MPI from the host (with ABI-matching)
- Guix: lends to BYO-MPI (software purity)
- Spack: lends to MPI loading (flexibility). *But with containers, we are doing the BYO approach currently.*

Support for MPI

chameleon homogeneous SGEMM - 2 nodes



Intel-MPI-Benchmark PingPong - 2 nodes



Reconciling high-performance computing with the use of third-party libraries?

JCAD, November 4-6, 2024, Bordeaux

Emmanuel Agullo

Support for CPU tuning

Automatically pass `–march` to packages and dependencies

```
$ guix install –tune=cascadelake chameleon
```

```
$ spack install chameleon target=cascadelake
```


Support for CUDA

The *cuda* package is provided by both Spack and Guix:

```
$ guix install cuda-toolkit
```

```
$ spack install cuda
```

Singularity requires running the container with the `-nv` flag

```
$ singularity exec -nv ...
```

Support for ROCM

You need to check which packages you need, for example:

```
$ guix install hipamd
```

```
$ spack install rocm-cmake
```

Spack package names:

<https://rocm.docs.amd.com/projects/install-on-linux/en/latest/how-to/spack.html#rocm-packages-in-spack>

What are package variants?

Similar to CPU tuning, packages can be modified to have different feature support. For example, variants with MPI or CUDA support, etc.

Spack:

Specification flags:

```
$ spack install chameleon +cuda
```

Guix:

Different package names:

```
$ guix install chameleon-hip
```

Which Supercomputers have we tested?

- Jean-Zay (Idris)
CUDA + InfiniBand, **Singularity fully supported**
* Requires an extra command to put container in “safe” directory
- Adastra (CINES)
ROCM + HPE Slingshot, **Singularity fully supported**
* Requires containers to be validated by the admin team
- Irene (TGCC)
CUDA + InfiniBand, In-house, Docker-compatible container runtime “pcocc”
- Vega, MeluXina (EuroHPC)
CUDA + MPI InfiniBand, **Singularity fully supported**
- LUMI (EuroHPC)
ROCM + HPE Slingshot, **Singularity fully supported**

Summary

Summary

- + Use a package manager (Guix or Spack) to install your C/C++ dependencies, it will make things easier.
- + Use Singularity containers to deploy it to supercomputers.
- + Get in contact with our team (PC5, WP3) if you need any help with setting up the new workflow.
- + Guix provides better reproducibility, while Spack flexibility.



“Getting Started” Steps

1. **Install Guix** on your development computer.
2. Use ***guix shell*** to get your development dependencies.
3. **Adjust your project** to find packages from the package manager, and remove your bundled dependencies.
4. Save your development dependencies to a declarative ***manifest.scm*** file, which is committed.
5. Create a container with the development dependencies with ***guix pack -m manifest.scm***
6. Package the application with Guix, and deploy it directly with Singularity.
7. Integrate Guix with CI/CD to build, test and generate containers.

Practical Session at 13:00

- Use Grid'5000 as a platform that:
 - Provides Guix and supports Spack
 - Provides Singularity
 - Can run a Chamelon with CUDA on GPU nodes

Make sure:

- You can connect to Lille site in Grid'5000
`ssh lille.g5k`
- You ran the guix pull command, as indicated in the email.

Time for questions!

Contact and info:

- Tutorials page: <https://numpex-pc5.gitlabpages.inria.fr/tutorials/>
- Slack: <https://numpex.slack.com/archives/C07UT051H7Y>

The slides will be sent in Slack #packaging channel

- Email
 - List: numpex-pc5-wp3@inria.fr
 - Fernando Ayats Llamas: fernando.ayats-llamas@inria.fr
 - Romain Garbage: romain.garbage@inria.fr