Backpacking with Code: Software Portability for DHTC

Christina Koch (ckoch5@wisc.edu)
Research Computing Facilitator
University of Wisconsin - Madison
Goals For This Session

• Understand the basics of...
  – how software works
  – where software is installed
  – how software is accessed and run

• ...and the implications for Distributed High Throughput Computing (DHTC)

• Describe what it means to make software “portable”

• Learn about and use software portability techniques
An Analogy

Running software on your own computer is like cooking in your own kitchen.

Photo by jschantz on flickr, CC-BY
On Your Computer

• You know what you already have.
  – All the software you need is already installed.
• You know where everything is (mostly).
• You have full control.
  – You can add new programs when and where you want.
Running on a shared computer is like cooking in someone else’s kitchen.
On Someone Else’s Computer

• What’s already there?
  – Is R installed? Or Python? What about the packages you need?

• Do you know where anything is?

• Are you allowed to change whatever you want?
The Solution

• Think like a backpacker.
• Take your software with you
  – Install anywhere
  – Run anywhere
• This is called making software *portable*
PRELIMINARY CONCEPTS
Software Programs Are Files

• Principle
  – Software is a set of files.
  – These files have instructions for the computer to execute.

• Implications for DHTC
  – Isolate the specific software files needed for a job and bring them along.
How Software Works*

Program

/software, code, executable, binary/
How Software Works*

Program
(software, code, executable, binary)

Running Program
(process, instance)

launches to
Program
(software, code, executable, binary)

Running Program
(process, instance)

- Program launches to Running Program
- Running Program depends on Program
- Running Program runs own tasks

*Not to scale
How Software Works*

Program
/software, code, executable, binary/

Operating System

Running Program
/process, instance/

makes requests
monitors running programs

depends on

launches to

runs own tasks

*Not to scale
How Software Works*

Program
/software, code, executable, binary/

Running Program
/process, instance/

Operating System

Hardware
/processors, memory, disk/

Program launches to Running Program
Running Program makes requests to Operating System
Operating System monitors running programs
Operating System translates program’s request
Hardware

Program runs own tasks

*Not to scale
How Software Works

• Principle:
  – Software depends on the operating system, and other installed programs.

• Implications for DHTC:
  – Software must be able to run on target operating system (usually Linux).
  – Know what else your software depends on.
• Where can software be installed?

/  
- bin/
- usr/
- lib/
- home/

system locations

local locations

- bin/
- local
- ada/
- al/
Location, Location, Location

- Who can install the software?

```
/   bin/
   usr/ bin/
   lib/ local
  home/ ada/
      al/
```

- administrator
- folder owner
Location, Location, Location

- Who can access the software?

```
/ 
|--- bin/
|--- usr/
|--- lib/
|--- home/
     |--- bin/
     |--- local
     |--- ada/
     |--- al/
     +-- anyone
         +-- folder owner+
```
Location, Location, Location

• Principle:
  – Software files have to be installed somewhere in the file system.

• Implications for DHTC:
  – Software must be installable without administrative privileges.
  – The software’s location needs to be accessible to you.
Command Line

How to automate programs?

```
[-]$: /Applications/Calculator.app/Contents/MacOS/Calculator
```
Command Line

• Principle:
  – To automatically run software, need to use text commands (command line).

• Implications for DHTC:
  – Software must have ability to be run from the command line.
  – Multiple commands are okay, as long as they can be executed in order within a job.
Command Line and Location

• To run a program on the command line, your computer needs to know where the program is located in your computer’s file system.

$ ls
$ python
$ ~/wrapper.sh

How does the command line know what `ls` is? Where is python installed?
Two Location Options

Provide a path (relative or absolute)

[~/Code]$ mypy/bin/python --version
Python 2.7.7

Use “the” PATH

$ export PATH=/Users/alice/Code/mypy/bin:$PATH
$ echo $PATH
/Users/alice/Code/mypy/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
$ which python
/Users/alice/Code/mypy/bin/python
Command Line

• Principle:
  – To run a program on the command line, the computer has to be able to find it.

• Implications for DHTC:
  – There are different ways to “find” your software on the command line: relative path, absolute path, and PATH variable
Portability

• Run “anywhere” by:
  – bringing along the software files you need…
  – to a location you can access/control…
  – using the command line to run…
  – by providing the correct software location…
  – (on Linux).
BRING ALONG SOFTWARE FILES
Ways to Prepare Software Files

• Download pre-compiled software

• Compile yourself
  – Single binary file
  – Installation contained in a single folder
What is Compilation?

Source Code

{  
  
}

compiled + linked into

compiler

and OS

libraries

uses

run on

Binary

0101

binary code by Kiran Shastry from the Noun Project
Source Code by Mohamed Mbarki from the Noun Project
Computer by rahmat from the Noun Project
books by Viral faisalovers from the Noun Project
Static Linking

Source Code

{ ... }

compiled + static link into

compiler
libraries
and OS

Static Binary

run anywhere

Book by Aleksandr Vector from the Noun Project
Compilation Process

• Use a compiler (like gcc) directly
  – Can use options to control compilation process

• More common:
  ./configure # can also include options
  make
  make install
Interpreted code

• Instead of being compiled and then run…

• …interpreted languages are translated into binary code “on the fly.”
What Kind of Code?

- Programs written in C, C++ and Fortran are typically compiled.
- For interpreted (scripting) languages like perl, Python, R, or Julia:
  - Don’t compile the scripts, but *do* use a compiled copy of the underlying language interpreter.
Matlab

• Matlab is a scripting language…but can also be compiled.

compile .m files using Matlab compiler (mcc) [Requires license]

compiled file and Matlab Runtime work together to run program. [no license needed]
Ways to Run Software

Executable

- Software must be a single compiled binary file.

```plaintext
executable = program.exe
queue 1
```

Wrapper Script

- Software can be in any compiled format.

```plaintext
#!/bin/bash

# run_program.sh

tar -xzf program.tar.gz
program/bin/run in.dat
```
Single Binary Workflow

Option 1
compile

Option 2
download

Submit server

Static binary

Execute server

{ code }

{ 0101 }

{ 0101 }

{ 0101 }
Wrapper Script Workflow

Submit server

wrapper script

source code, compiled code or single binary

Execute server

set up  run

set up  run

set up  run

script by ✦ Shmidt Sergey ✦ from the Noun Project
BRING ALONG CONTAINERS
Containers

• Containers are a tool for capturing an entire job “environment” (software, libraries, operating system) into an “image” that can be used again.
Returning to Our Analogy…

• Using a container is kind of like bringing along a whole kitchen…

Photo by PunkToad on Flickr, CC-BY
Why Containers?

Why use containers instead of the methods we just discussed?
Why Containers?

- Complex installations: software that has a lot of dependencies or components.

Photo on pikrepo
Why Containers?

- Software that can’t be moved: do files or libraries have to be at a specific path?
Why Containers?

• Sharing with others: one container can be used by a whole group that’s doing the same thing.
Why Containers?

- Reproducibility: save a copy of your environment.
Using Containers

- To use a container as your software portability tool, need to either:
  - Find a pre-existing container with what you need.
  - Build your own container.*

* not covered today
Container Types

- Two common container systems:
  
  **Docker**
  https://www.docker.com/

  **Singularity**
  https://sylabs.io/

The container itself will always be some version of Linux - but can be run on Linux / Mac / Windows if Docker or Singularity is installed
Submit File Requirements

- **Docker (from CHTC submit server)**
  
  ```
  universe = docker
  docker_image = python:3.7.0
  requirements = (HasDocker == true)
  ```

- **Singularity (from OSG submit server)**
  
  ```
  +SingularityImage = "/cvmfs/singularity.opensciencegrid.org/centos/python-34-centos7:latest"
  requirements = (HAS_SINGULARITY == true)
  ```
Container Workflow

Submit server

Script

+ name of container image

Registry (e.g. DockerHub)

Execute server(s)
Conclusion

To use any software in a DHTC system:

1. Create/find software package:
   - download pre-compiled code, compile your own, create/find a container

2. Account for all dependencies, files, and requirements in the submit file.

3. If needed, write a script to set up the environment when the job runs.
Pre-existing Software

• The ideal for DHTC is to package and bring along your own software, but...

• You can use pre-existing software installations if the computers you’re running on have your software installed (or access to a repository with the software).
Pre-existing Software via OSG Connect

• On the Open Science Grid, jobs submitted from OSG Connect have access to a software repository maintained by OSG Connect staff.

• The software repository is available across the OSG.

• Software is accessed using “modules”.

OSG Virtual School Pilot 2020
Software Across the OSG

OSG Connect

United States

San Francisco
Los Angeles
Las Vegas
Phoenix
Denver
Chicago
New York
Philadelphia
Montreal
Ottawa
Atlanta
Houston

Open Science Grid
Module Commands

- See what modules are available
  
  
  $ module avail

  $ module spider lammps

- Load a module

  $ module load lammps/20180822

- See loaded modules

  $ module list
Module Workflow

1. Find a module for your software.
2. Write a wrapper script that loads the module and runs your code.
3. Include requirements to ensure that your job has access to modules.

```python
requirements = (HAS_MODULES == true) &&
(OSGVO_OS_STRING == "RHEL7") && (OpSys == "LINUX")
```
Module Workflow

Submit server

wrapper script
+ requirements

software modules

Execute server

load + run

load + run

load + run