



Special Cases: Licenses, Interpreted Languages, and Containers for DHTC

Wednesday morning, 10:45 am

Christina Koch (ckoch5@wisc.edu)

Research Computing Facilitator

University of Wisconsin - Madison

Expanding Our Horizons

- Previously, we were using simple, open source code that could be easily compiled or built.
- This presentation discusses some special cases:
 - Licensed software
 - Running interpreted languages (Matlab, Python)
 - Using containers



Open Science Grid

LICENSING

Licensing

- Many scientific softwares are licensed.
- Licenses are restrictive, particularly for high-throughput computing

License Variations

- Per machine or 'single-install'
- Per *running* instance of the software (per “job”)
- Per username / user
- Via a license server
 - can support 1 - 1000s of concurrently running processes (“seats”)

Licensing implications for DHTC

- Per machine or 'single-install': can't be used for DHTC
- Per job: restrictive, limits the number of jobs you can have running, how do you access licenses from execute servers?
- Username: restrictive, could only run jobs on one system where your jobs run as *your username*

Approaches

- Seek out open source alternatives
 - Python or R packages that emulate specific software behavior
 - If you can't replace entire workflow, substitute free software where you can
- License-free workarounds (Matlab)
- Choose the least restrictive license possible



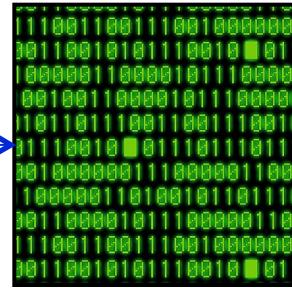
Open Science Grid

INTERPRETED LANGUAGES

Interpreted code

- Instead of being compiled and then run...

```
1 //pass
2 // - file("login.dat")
3 $i = 0; $i < count($users); $i++
4 $line = $users[$i];
5 if (eregi("username(.*)", $line))
6 {
7     // User gevonden, password is nu
8     // opgeslagen in $pass
9     $pass = $reg[$i];
10    break; // stop met de 'for'-loop
11 }
12 return $pass;
13
14 function IsLoggedIn()
15 {
16     Global $username, $password;
17     if ($username == $password)
18         $pass = md5(GetPassword);
19     else
20         $pass = FALSE;
21 }
```

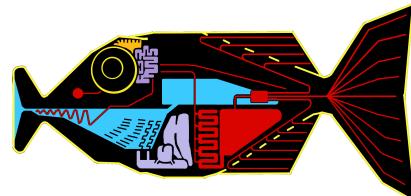


```
1110011001110010000000
10110010101110010101
0000001100001011100000
0010011000010111000000
1101101110011001110011
01110010101110110111
101000000111000001100
1000001101001011011111
101100010111000001110
1110011001110010000000
10110010101110010101
```



- ...interpreted languages are translated into binary code “on the fly”

```
1 //pass
2 // - file("login.dat")
3 $i = 0; $i < count($users); $i++
4 $line = $users[$i];
5 if (eregi("username(.*)", $line))
6 {
7     // User gevonden, password is nu
8     // opgeslagen in $pass
9     $pass = $reg[$i];
10    break; // stop met de 'for'-loop
11 }
12 return $pass;
13
14 function IsLoggedIn()
15 {
16     Global $username, $password;
17     if ($username == $password)
18         $pass = md5(GetPassword);
19     else
20         $pass = FALSE;
21 }
```

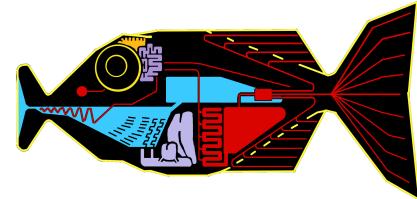


Interpretation

Script

```
1 $pass = null;
2 // file("login.dat");
3 $i = 0; $i < count($users); $i++
4 $line = $users[$i];
5 if (ereg("^\$username(.*)", trim($line))
6     // User gevonden, Password is nu
7     $pass = $regs[1];
8     break; // Stop met de 'for'-loop
9 }
10 return $pass;
11 }
12 function IsLoggedIn() {
13     global $username, $password;
14     if ($username && $password)
15         $pass = md5(GetPassword());
16     return ($pass == $password) ? TRUE : FALSE;
17 }
```

Interpreter



text turns
into binary
instructions

uses



Libraries





Open Science Grid

On the command line

A screenshot of a macOS terminal window titled "ckoch — bash — 53x14". The window has three tabs: "bash", "ckoch5@submit-5:~" (selected), and "ckoch5@os...ster/osg-ss" and "ckoch5@os.../osg/python".

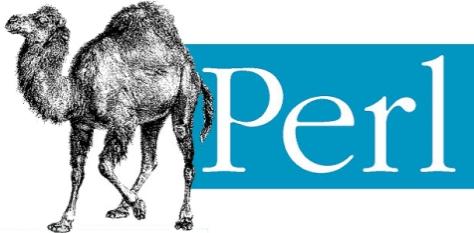
```
[~]$ cat hello.py
import sys

name = sys.argv[1]
print "Hello", name
[~]$ python hello.py "Open Science Grid"
Hello Open Science Grid
[~]$
```

The terminal shows the creation of a file named "hello.py" containing a simple Python script that prints "Hello" followed by the first command-line argument. The script is then run with the argument "Open Science Grid", resulting in the output "Hello Open Science Grid". A cursor is visible at the end of the command line.

Common interpreted languages*

- Python
 - R
 - Julia
 - Ruby
- Matlab
 - Perl
 - Javascript



*Note: the line between interpreted/compiled languages can be fuzzy. Many languages support both options, with one method being more common.

Running interpreted code in jobs

General procedure

- Need to bring along interpreter and script
- Use a wrapper script as the executable
- Wrapper script will:
 - “Install” the interpreter
 - Run the script using the local installation

Python on DHTC

1. Create a portable Python installation
(optional)
2. Bring along:
 - pre-built installation OR Python source code
 - your Python code
3. Use a wrapper script to:
 - unpack pre-built install OR install from source
 - run your Python script

(Similar to Exercise 1.4 this morning, will also work for R)

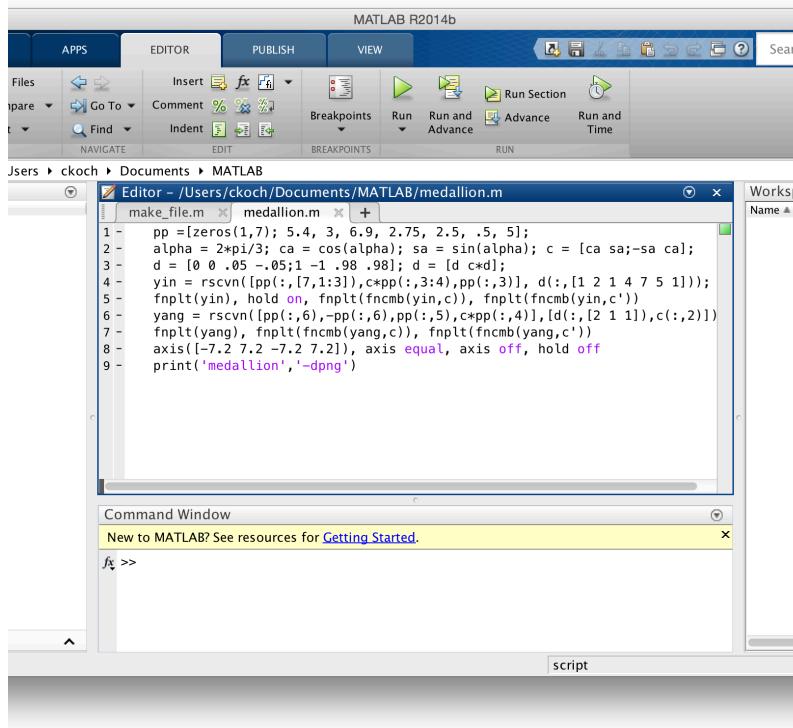
Matlab

- Wait a minute...isn't Matlab licensed?
- Yes, when interpreted on your computer using a normal Matlab installation.
- However, Matlab code can also be compiled.
- Once compiled, the code can be run without a license using a (free) set of files called the Matlab runtime (which acts like the interpreter).



Matlab contrast

Running Matlab on your computer Uses license per instance



**Running Matlab on DHTC
Uses license once, runs
many instances for free**

Matlab script(s)

compiled w/ Matlab compiler (uses license)

Compiled binary
interpreted by
Matlab Runtime (free)



Matlab on DHTC

1. Compile Matlab code using the Matlab compiler (mcc)
 - requires a license
2. Prepare a copy of the Matlab runtime
 - download for free from Mathworks
3. Write a script that “installs” the runtime
 - The Matlab compiler actually writes most of this script for you
4. Use the runtime install to run the compiled Matlab code

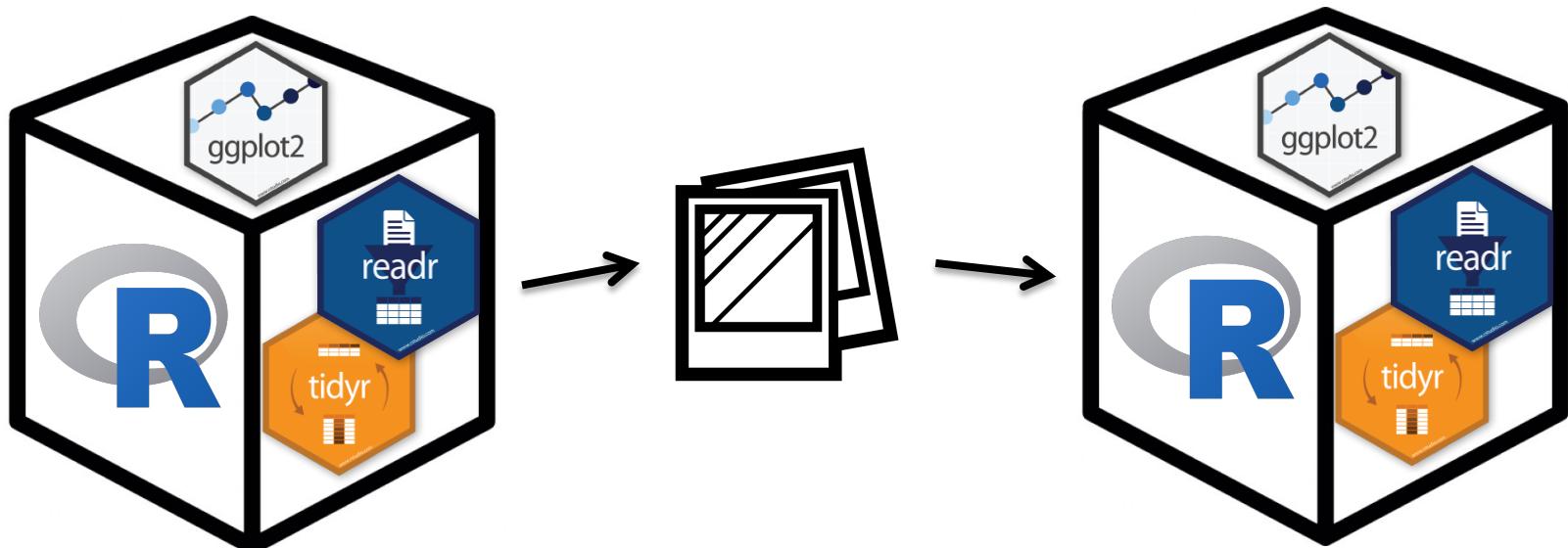


Open Science Grid

CONTAINERS

Containers

- Containers are a tool for capturing an entire job “environment” (software, libraries, operating system) into an “image” that can be used again.



Using Containers in DHTC

- Two common container systems:

Docker

<https://www.docker.com/>



Singularity

<http://singularity.lbl.gov/>



- Requirements:
 - Underlying container system needs to be installed on the computers where your job runs
 - Permissions on that system allow the use of containers

Container Workflow

1. Create a container or find one online
 - DockerHub: <https://hub.docker.com/>
 - SingularityHub: <https://singularity-hub.org/faq>
2. Place container into public or private registry
3. Create a customized script/submit file that fetches/uses the container
 - Docker: Use HTCondor's docker universe
 - Singularity: Wrapper script

Conclusion

To use any software in a DHTC system:

1. Create environment/software package
 - download pre-compiled code, compile your own, build your own, create/find a container
2. Write a script to set up the environment when the job runs
3. Account for all dependencies, files, and requirements in the submit file

Exercises

- Running Matlab Jobs
 - Exercise 1.6
- Running Python Jobs
 - Exercise 1.7: Pre-building Python and using that installation
 - Exercise 1.8: Writing a script that installs Python with every job
- Half of the room should start with Matlab, the other with Python

Questions?

- Feel free to contact me:
 - ckoch5@wisc.edu
- Now: Hands-on Exercises
 - 11:00am-12:15pm
- Next:
 - 12:15-1:15pm: Lunch
 - 1:15 onward: free time