Center for Advanced Computing (CAC)

Using Containers to Create More Interactive Online Training and Education Materials

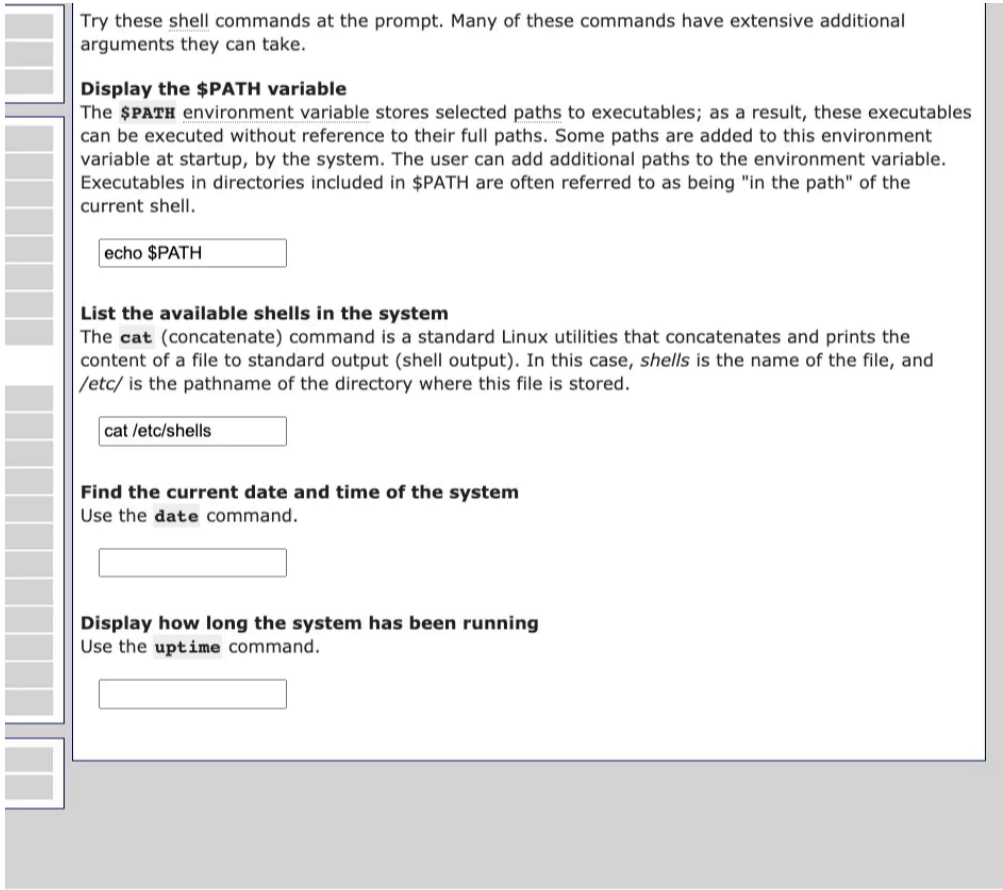
Brandon Barker - brandon.barker@cornell.edu
Susan Mehringer - shm7@cornell.edu

PEARC20 - 29 July 2020

Cornell Container Runner Service (CCRS)

- Cornell Virtual Workshop
  - Online training platform, evolving since 1994
  - Not-for-credit training units on computational topics
  - Goal: effective instruction through appropriate components
- CCRS was developed to incorporate hands-on exercises that are realistic and immediately accessible
  - No need to get an allocation
  - No need to leave the browser
  - No need to install software
- CCRS uses a container back-end built to look and feel like the target platform

Demo


Try these shell commands at the prompt. Many of these commands have extensive additional arguments they can take.
Display the $PATH variable
The $PATH environment variable stores selected paths to executables; as a result, these executables can be executed without reference to their full paths. Some paths are added to this environment variable at startup, by the system. The user can add additional paths to the environment variable. Executables in directories included in $PATH are often referred to as being "in the path" of the current shell.
echo $PATH
List the available shells in the system
The cat (concatenate) command is a standard Linux utilities that concatenates and prints the content of a file to standard output (shell output). In this case, shells is the name of the file, and /etc/ is the pathname of the directory where this file is stored.
cat /etc/shells
Find the current date and time of the system
Use the date command.
Display how long the system has been running
Use the uptime command.
© 2020 Cornell University | Cornell University Center for Advanced Computing | Copyright Statement

Demo

Program hello_mpi.c

Compile and run

```
#include <stdio.h>
#include <string.h>
#include <mpi.h>

int main(int argc, char **argv)
{
  char message[20];
  int  i, rank, size, tag = 99;
  MPI_Status status;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  if (rank == 0)
  {
    strcpy(message, "Hello, world");
    for (i = 1; i < size; i++)
      MPI_Send(message, 13, MPI_CHAR, i, tag, MPI_COMM_WORLD);
  }
  else
    MPI_Recv(message, 20, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &status);

  printf( "Message from process %d : %.13s\n", rank, message);
```

Result:

In case you're interested in learning the details, here are the roles of all the MPI routines in the above code, and types of parameters involved in each call.

Initializing an MPI process

MPI_Init must be the first MPI routine you call in each process. It can only be called once. It establishes an environment necessary for MPI to run. This environment may be customized for any MPI runtime flags provided by the MPI implementation (note that the command line arguments are passed to the C version of this call).

- int MPI_Init(int *argc, char ***argv)

Finding the number of processes

MPI_Comm_size returns the number of processes within a communicator. A communicator is MPI's mechanism for establishing separate communication "universes" (more on this later). Our sample program uses the predefined "world communicator" MPI_COMM_WORLD, which includes all your processes. MPI can determine the number of processes because you specify this when you issue the command used to launch MPI programs.

Why Develop CCRS?

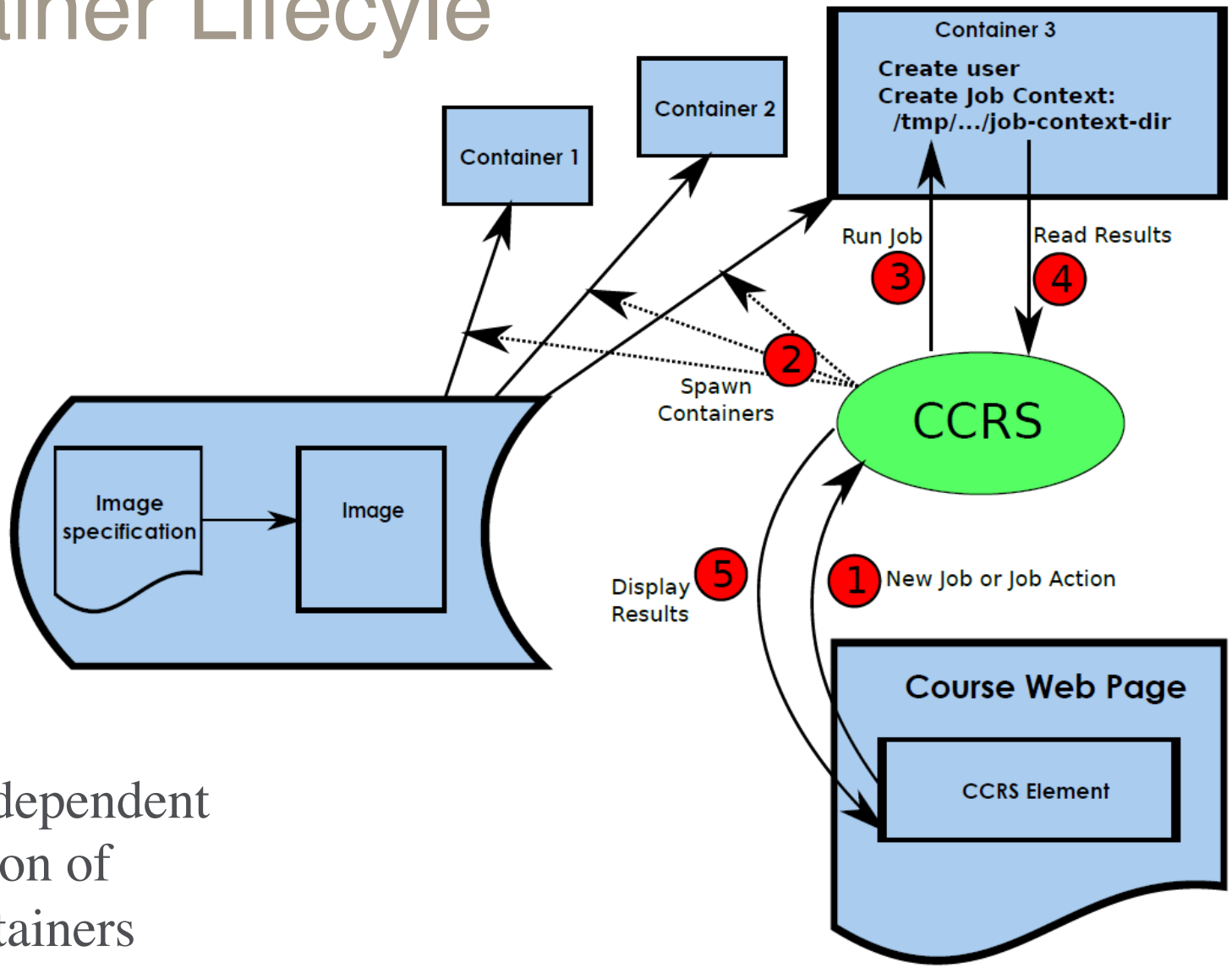
Unlike Jupyter and other notebooks:

- Take advantage of abundance and customizability of containers
  - A content author could supply their own or use a pre-existing container
  - Much simpler to create containers than a custom Jupyter kernel
- Easily embed in any training web page
- Support a variety of elements commonly used in HPC:
  - Customized editors, linked with custom commands, to custom containers!
  - Generic command runners
  - Shell interface (in the works)

From CJRS to CCRS

- CCRS based on the Cornell Job Runner Service (CJRS)
- CJRS developed at CAC immediately before CCRS by Aaron Birkland and Susan Mehringer
- CCRS uses custom “job” management instead of Slurm
- Traditional notion of job management not as much of a concern as flexibility attributed to containers.
- Exercise jobs are lightweight and can be limited via ulimit, container run parameters, etc.

CCRS Container Lifecyle



Not shown: container-type-dependent handling of garbage collection of job contexts, users, and containers

Implementation Overview

- Implemented in Scala (server) and Scala.js (browser client)
  - Allows a uniform language for RPCs between the frontend and backend
  - Employs the ZIO library to keep most code purely functional, reducing errors and simplifying concurrency
- Various job types implemented as *typeclasses*
  - More job types can be added easily in the future
  - Reduces the need to worry about inheritance
    - instead it is a decoupled interface
- Copious logging used for tracking potential problems due to:
  - Application logic, OS, or Container errors
  - Problems with users' jobs
  - And (hopefully unlikely) malicious users

An Example, but First: CCRS API

- Scala.js and JavaScript APIs exist
- The JavaScript API is a wrapper around Scala.js APIs
- The current API is fairly verbose, allowing greater customization
- We are exploring options for creating simpler JavaScript APIs on top of this for common use cases

IOW, the content developer can

- Use the simplified JavaScript API for common use cases, or
- Use the fully customizable JavaScript API

Integrating CCRS: An example

- To use the API, the page must load the requisite JavaScript code, optionally including ace.js if an editor-based example is used on the page:

```
<script type="application/javascript" src="http://w.x.y.z:port/ace/ace.js" charset="utf-8"></script>
<script type="application/javascript" src="http://w.x.y.z:port/target/web-client-jsdeps.js"></script>
<script type="application/javascript" src="http://w.x.y.z:port/target/web-client-opt.js"></script>
```

- Next, a particular element, such as a one-shot command, can be added:

```
<h2>Free-form single-command input</h2>
<input type="text"
       placeholder="Enter a command:"
       value="pwd"
       onkeydown="oneShotHandler(event)" />
</body>
<div id="one-shot-demo"></div>
```

Integrating CCRS: An example (continued)

```
<script type="application/javascript">
  var ccrsApiNamespace =
    "org.xsede.jobrunner.model.ModelApi";
  var pythonExampleMetaJson = {
    "$type": ccrsApiNamespace + ".SysJobMetaData",
    "shell": ["bash"],
    "containerType": {
      "$type":  ccrsApiNamespace + ".Singularity"
    },
    "containerId": [],
    "image": ["vsoch-master-latest.simg"],
    "binds": [],
    "overlay": [],
    "user": "ccrsdemo",
    "address": [],
    "hostname": [],
    "url": window.location.href
  };
  var pythonExampleMeta =
    CCRS.sysJobMetaData(pythonExampleMetaJson);
  var oneShotId = CCRS.makeJobId();
  var oneShotCommand = CCRS.makeOneShotCommand(
    document.getElementById("one-shot-demo")
  );
  var oneShotHandler = CCRS.makeCmdHandler(
    oneShotCommand,
    pythonExampleMeta,
    oneShotId
  );
</script>
```

In Conclusion

- Currently CCRS can
  - One-shot command, e.g. “echo $PATH”
  - Edit code, then run it (any software that is on the container)
- Additions currently in development include
  - Interactive terminal/shell
  - Improved security
  - Display generated images and files
- Implementation by the web page developer will be simplified by
  - Provide encapsulated JavaScript APIs for common use cases
- Possible community share via GitHub
  - Show of hands: Share this with the community?
  - Interest in testing beta? Please send email (~ 6 months)

Thank you.

Brandon Barker brandon.barker@cornell.edu
Susan Mehringer shm7@cornell.edu

Cornell University Center for Advanced Computing (CAC)