Cornell University
Center for Advanced Computing

Incorporating Interactive Compute Environments into Web-Based Training Materials using the Cornell Job Runner Service

Aaron Birkland
apb18@cornell.edu
Consultant/Analyst

Susan Mehringer
shm7@cornell.edu
CAC Assistant Director, Consulting
XSEDE Training Lead

Cornell University Center for Advanced Computing (CAC)

Aaron and Lars

Why?

Traditionally, training materials and compute environments have been separate entities. Students learn from online materials in one window, then log into a new machine or session to try out new skills or concepts.

Accessing this second environment can impose obstacles such as

- Gaining access to the appropriate computer
- Learning to navigate a computer-specific login environment and file system

Goal: provide in-place realistic practice and experimentation

Requirements

Develop a software toolkit that will enable the online educational developer to design pages with these features:

- Embed a compute environment experience directly into web pages
- Try out commands and run jobs without obtaining an account or leaving the web page
- Embedded environment should look, feel, and react like a typical HPC login node / batch job
- The environment can be backed by real or virtual compute resources.

While we were working…

	IPython Notebook	LinkSCEEM	ISLET	Geordi
Integrate into existing, mature training documentation in the form of .html or .aspx files	Yes	Yes	No	Yes
Emulate the software stack and fundamental HPC technologies (MPI, OpenMP, job submission) of XSEDE resources such as Stampede	Yes	Yes	Yes	No
Allow users to do potentially dangerous things in a secure fashion, such as compile and run C code	No	Yes	Yes	Yes
Support interaction paradigms ranging from a full command line interface (shell), to clicking a button to run a code snippet embedded on a page	No	No	No	No

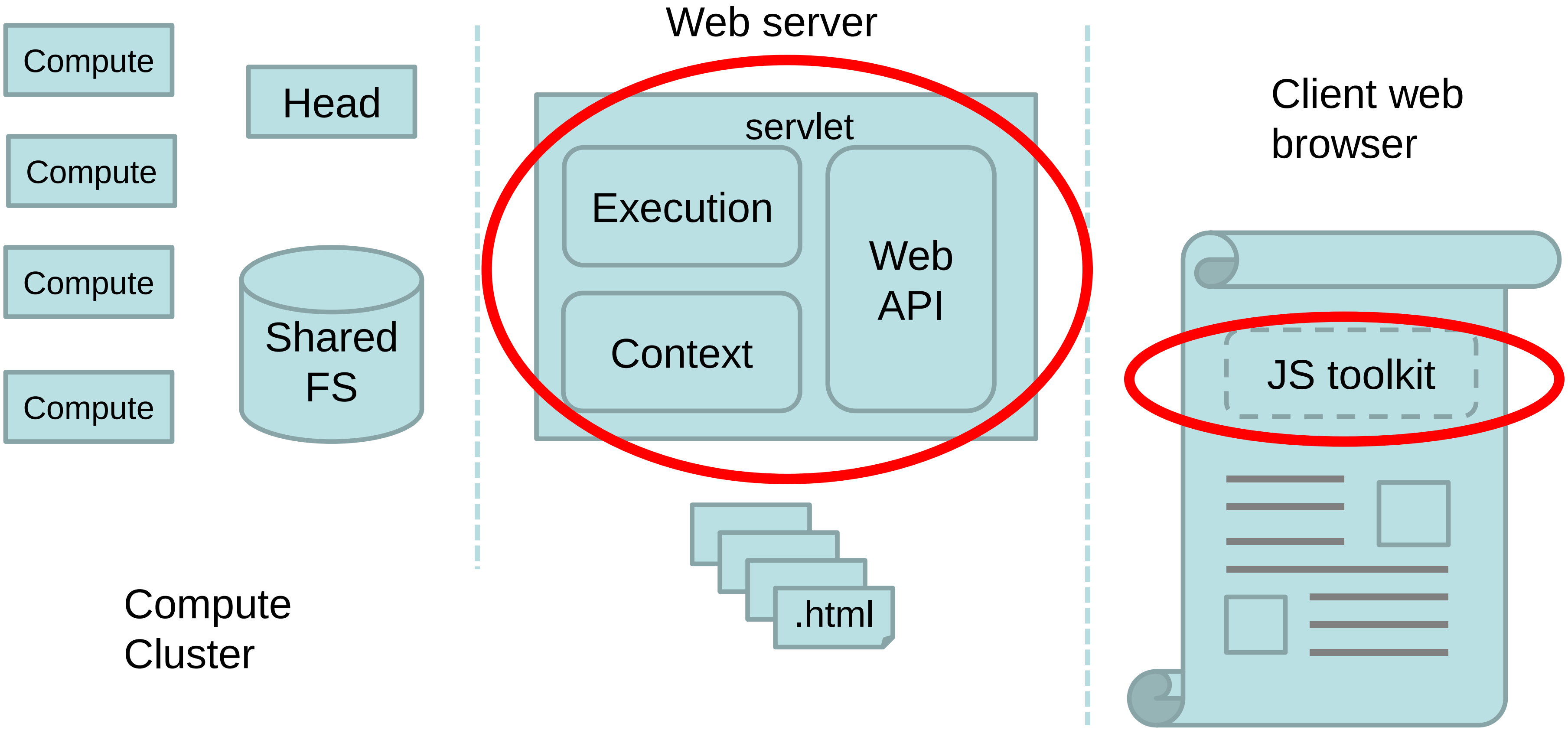
Solution

Using the *Cornell Job Runner Service*^{SM} (CJRS), along with the toolkit, we can embed a computing environment directly into web pages.

The CJRS toolkit can be used to configure different interactive modes. We began with these three specific scenarios:

1. Linux console configured as a general login node
2. Form element that launches a pre-defined SLURM job
3. Guided session which allows the user to walk through pre-planned steps of compiling, fixing, and running MPI code.

Architecture (CAC Contributions in red)



Demo 1:
Linux console configured as a general login node

https://cvw.cac.cornell.edu/environment/commands.aspx

https://cvw.cac.cornell.edu/linux/exerciseShells.aspx

- Embed a simple, scrollable preformatted text box (representing the console) in a web page
- The console displays the STDIN and STDOUT of any command that is typed in
- Used to execute individual commands

- It is not a window to a real console, and it is not a terminal emulator
- At present cannot be used for activities that assume that the learner is at a terminal, such as editing with vi

Demo 1:
Linux console configured as a general login node

Demo 1: What just happened?

- When “Launch a console” button is clicked, a SLURM interactive session is requested via `srun /bin/bash`, which executes a bash shell in a CJRS VM dedicated to Virtual Workshops.
- After typing text into the input box and pressing <ENTER>, this text is POSTed to a special file `.STDIN`, and is interpreted by the bash shell running on the VM.
- Any output (stdin or stdout) produced by the bash shell is directed to a special file `.CONSOLE,` which the javascript toolkit displays in the output text box.

Demo 2: Launch a pre-defined SLURM job

https://cvw.cac.cornell.edu/cintro/functions.aspx

- Run any command or program on-the-fly
- E.g. execute a run that is dependent on changing input
- Can be used as building blocks to demonstrate a set of tasks.

- A web page can contain two or more independent forms.
- Two forms cannot execute at the same time (the first will be disabled)
- The forms can be submitted any number of times, in any order.

Demo 2: Launch a pre-defined SLURM job

Demo 2: What just happened?

- When the "compile and run this code" button is pressed, the job runner service creates a temporary working directory for the job, and uploads two files: `cexample.c` and `job.sh`
  - The content of `cexample.c` is the c code shown on screen.
  - The content of `job.sh` is part of the html page in a hidden element
- A SLURM batch session is requested via `sbatch job.sh`. This executes on the job runner VM.
- The `job.sh` batch file compiles `cexample.c`, runs it, and directs its output to a file `output.txt`
- When the javascript client detects that `output.txt` has been created, it unhides a specified html element and places the content of `output.txt` in it, displaying it on the page.

Demo 3: Compile, edit, and run MPI

https://cvw.cac.cornell.edu/mpi/exerciseinteractive.aspx

Guided session which allows the user to walk through pre-planned steps of compiling, fixing, and running MPI code.

Demo 3: Compile, edit, and run MPI

Demo 3: What just happened? (1 of 2)

- When “Compile with mpicc” is clicked, a SLURM interactive session *with 4 parallel tasks* is requested via `srun -n 4 /bin/bash`, which executes a bash shell on the CJRS VM for Virtual Workshops
- The contents of the text box containing the MPI code is uploaded to a file `hello.c` in the current working directory of the job
- A command is sent to the bash shell that compiles the code, and directs any output to a file `bad_compile.out`
  `mpicc hello.c > bad_compile.out 2>&1`.
- The MPI code on the page is set to edit-enabled
- The content of `bad_compile.out` is shown on the page.

Demo 3: What just happened? (2 of 2)

- When the user edits the MPI code and clicks ‘Compile with mpicc’, the contents of the text box replaces `hello.c`, and it is re-compiled as before.
- When the content of `bad_compile.out` is empty (i.e. when it compiles successfully without error), a new section of the page is un-hidden which presents a text field for running `mpiexec` with various arguments
- When a user types an `mpiexec` command and clicks ‘Run’, the command is evaluated by the bash shell being run by `srun`, and directed to `mpi.out`.
- The contents of `mpi.out` are shown in a text box on the page.

Security

CJRS API can be configured to require an opaque “token” string which can be passed to an external validation service. All other security is external to the application:

- Network traffic can be secured via SSL, firewalls, or network traffic routing rules
- Web pages can require authentication
- SLURM can be used to enforce time or resource limits
- Docker could be used to add a layer of security and limit resources
- CJRS jobs run at CAC execute as a single, unprivileged user on a single virtual machine that is periodically terminated and relaunched from it’s base image

CAC Implementation

Intentionally designed to be light weight mechanism, leaving much of the exposed capabilities and performance characteristics to the environment in which it is deployed.

- Single virtual machine instance in Red Cloud, created from a master image (can be destroyed and recreated at any time)
- The VM hosts the CJRS, the SLURM scheduler, and all job runs
- SLURM is configured with a one node queue, capable of running 32 scheduled tasks
- Configured with Open MPI
- All jobs run as a single unprivileged user
- Temporary home directory lasts for the duration of a single job
- It is responsive, cost-effective, and meets modest demand

Extensibility

CJRS can be configured to use different JobExecutionService implementations.

Some alternatives:

- Configure local execution service which executes one of a list of allowed commands
- Configure SLURM to run on a cluster
- Use SLURM to dynamically launch and tear down needed cloud nodes

Future Development

- Testing, testing, testing
- Harden applications
- Provide more realistic environment (ctrl-c?)
- Explore more usage scenarios
- Incorporate into more modules
- InCommon authentication
- Simplify content developer tools
- Explore container technologies (Docker) for distribution
- Share code

Contact Us

Susan Mehringer
shm7@cornell.edu

Aaron Birkland
apb18@cornell.edu

Feedback from friendly testers welcome!