A Tutorial Introduction to RAJA and Umpire

Kristi Belcher, Arturo Vargas
on Behalf of the RAJA Team

LLNL-PRES-853131
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

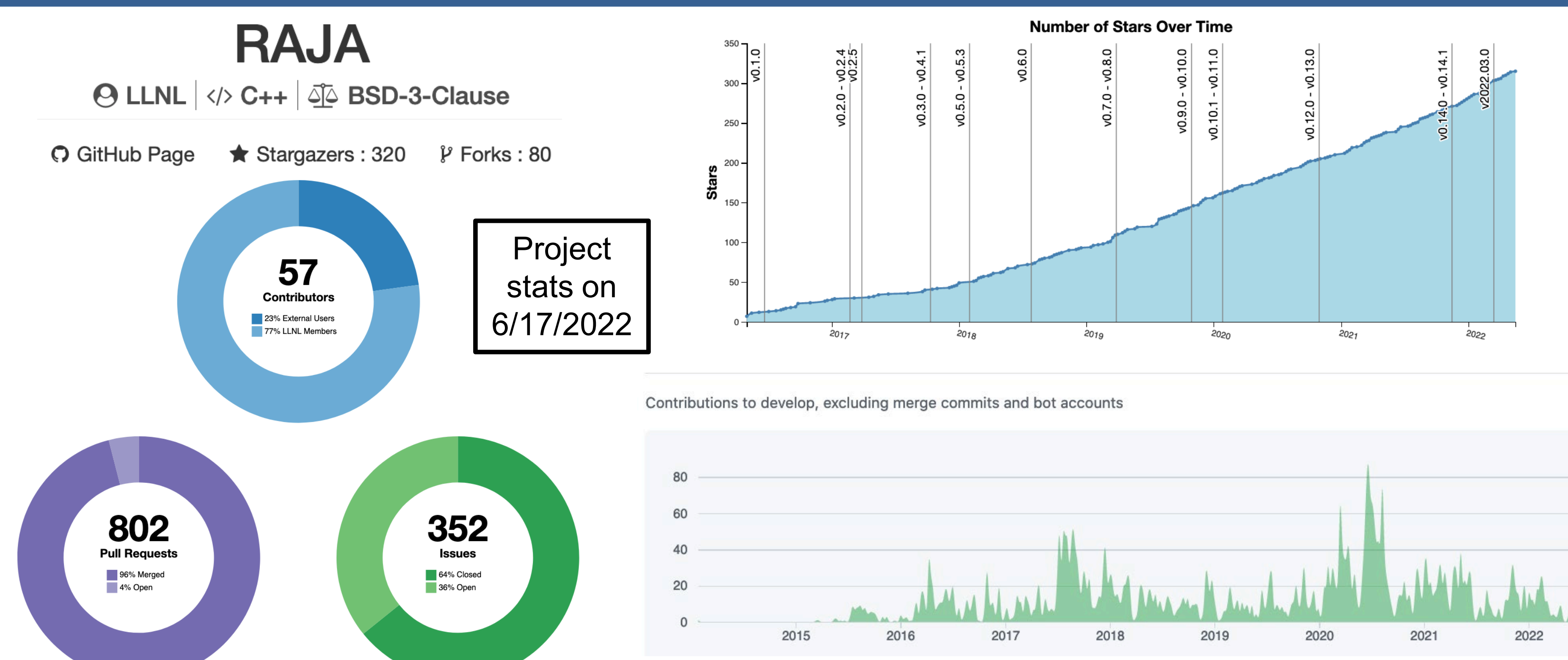
Welcome to the RAJA Umpire tutorial

- We will start by providing a brief high-level overview of RAJA and Umpire
- We will follow the README's in the tutorial exercise directory
- The tutorial contains descriptions of RAJA and Umpire features and shows how to use them
  - The RAJA code repository contains source files with exercises based on the tutorial docs that you can work through. Complete solution files are also provided if you wish to compare with your work, or if you get stuck.
  - Please don't hesitate to ask questions at any time on Webex chat or unmute yourself during this presentation.
  - We will primarily monitor Slack after the presentation.
- Our objective for today is for you to learn enough about RAJA and Umpire to start using it in your own code development
- RAJA contains other more advanced features that will be covered in future versions of the tutorial

We value your feedback…

- If you have comments, questions, or suggestions, please let us know
  - Send us a message to our project email list: raja-dev@llnl.gov
- We appreciate specific, concrete feedback that helps us improve RAJA and the tutorial material

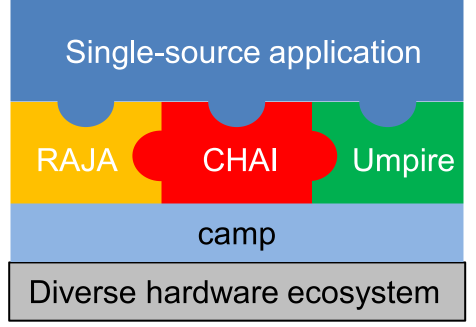
RAJA is an open-source project with a growing user and contributor base



RAJA is part of the RAJA Portability Suite, which contains four complementary projects

RAJA: C++ kernel execution abstractions

Enable single-source application code insulated from hardware and programming model details



Umpire: Memory management API

High performance memory operations, such as pool allocations, with native C++, C, Fortran APIs

camp: C++ metaprogramming facilities

Focuses on HPC compiler compatibility and portability

https://github.com/LLNL/RAJA
https://github.com/LLNL/CHAI
https://github.com/LLNL/Umpire
https://github.com/LLNL/camp

CHAI: C++ array abstractions

Automates data copies, based on RAJA execution contexts, giving apps the look and feel of unified memory, but with better performance

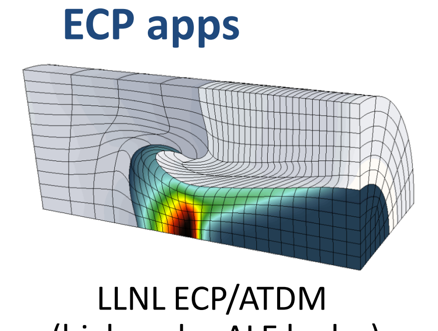
RAJA capabilities and core concepts

RAJA and performance portability

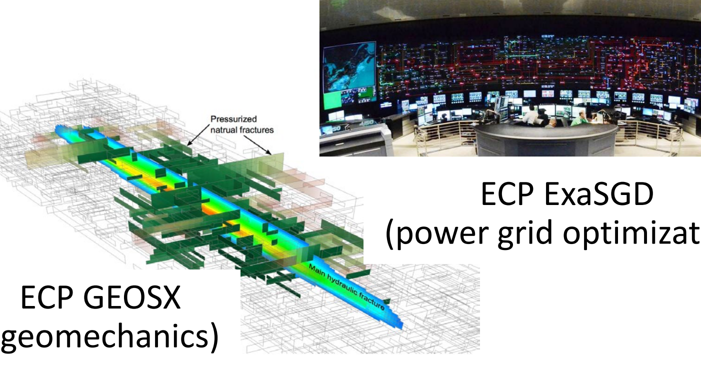
- RAJA is a **library of C++ abstractions** that enable you to write **portable, single-source** kernels that run on different hardware by re-compiling
  - Multicore CPUs, Xeon Phi, GPUs (NVIDIA, AMD, Intel), …
- RAJA **insulates application source code** from hardware and programming model-specific implementation details
  - OpenMP, CUDA, HIP, SIMD vectorization, …
- RAJA is used by many diverse production applications and libraries at LLNL and elsewhere, including ECP projects, university and vendor collaborators

The RAJA Portability Suite insulates applications from programming model and hardware architecture details

ECP apps

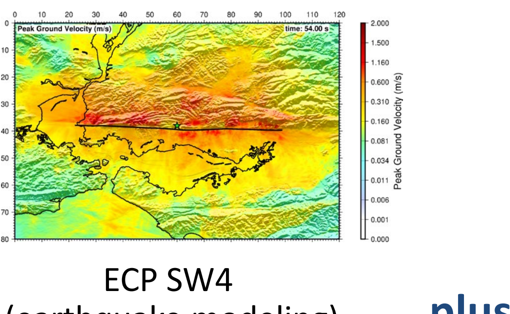


LLNL ECP/ATDM
(high-order ALE hydro)



ECP GEOSX
(geomechanics)

ECP ExaSGD
(power grid optimization)

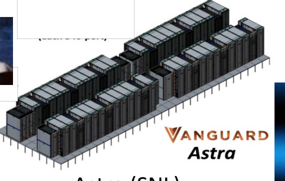


ECP SW4
(earthquake modeling)

plus, others…

RAJA / Umpire / CHAI

Perlmutter (LBL)
AMD Milan CPUs +
NVIDIA Ampere GPUs



Astra (SNL)
ARM architecture

Sierra (LLNL)
IBM P9 CPUs + NVIDIA Volta GPUs

Aurora (ANL)
Intel Xeon CPUs + Xe GPUs

Frontier (ORNL) &
El Capitan (LLNL)
AMD CPUs + GPUs

RAJA supports a variety of loop patterns and parallel constructs

Simple & complex loop patterns
- Non-perfectly nested loops
- Loop tiling
- Hierarchical parallelism
- Asychronous execution

Loop patterns and transformations (without changing app code)
- Change loop iteration patterns, permute loop nest ordering
- Multi-dimensional data views with offsets and index permutations
- Hierarchical parallelism, asynchronous execution
- Direct GPU thread-block mapping control
- CPU/GPU shared and thread local memory

Multiple execution back-ends
- Sequential
- SIMD (via vector intrinsics, in progress)
- OpenMP (CPU & device offload)
- Intel Threading Building Blocks (partial)
- CUDA
- AMD HIP
- SYCL (in development)

Portable reductions, scans, atomic operations, sorts…

Also, GPU kernel fusing (to reduce impact of GPU launch overhead for small kernels).

RAJA execution policy capsulates loop execution details

```
for (int i = 0; i < N; ++i)
{
  y[i] = a * x[i] + y[i];
}
```

```
#pragma omp parallel for
for (int i = 0; i < N; ++i)
{
  y[i] = a * x[i] + y[i];
}
```

```
int i = threadIdx.x +
        blockIdx.x*blockDim.x;
(i < N)
{
  y[i] = a * x[i] + y[i];
}
```

RAJA Execution Policy

```
RAJA::forall<EXEC_POL>( it_space, [=] (int i)
{
   y[i] = a * x[i] + y[i];
} );
```

Umpire capabilities and core concepts

Umpire provides a portable memory management API

Intuitive concepts

- Resources
- Allocators
- Operations

Supported memory types

- Host (CPU)
- MPI shared memory
- GPU global, constant, (host) pinned
- Unified memory
- Mmapped file memory
- Support for NVIDIA, AMD, Intel GPU devices

Features useful in HPC applications

- Various pool allocation strategies (fixed size, dynamic, monotonic, etc.)
- NUMA support
- Memory allocation advice (preferred location, mostly read, etc.)
- Thread safe allocators
- Memory introspection

- **Native interfaces for C++, C, and Fortran**
- **Logging, backtrace, and "replay" capabilities. Useful for investigating application performance, experimenting with different allocation scenarios, finding bugs, etc.**

Why We Need Umpire - Closer Look

- Depending on underlying architecture being used, calls to memory can look very different:

```
cudaMalloc((void**)&dev_ptr, SIZE * sizeof(float));        ← CUDA
hipMalloc((void**)&dev_ptr, SIZE * sizeof(float));         ← HIP
void* ptr = sycl::malloc_device(SIZE, queue_t);            ← SYCL*
void* ret = omp_target_alloc(SIZE, device);                ← OpenMP Target*
```

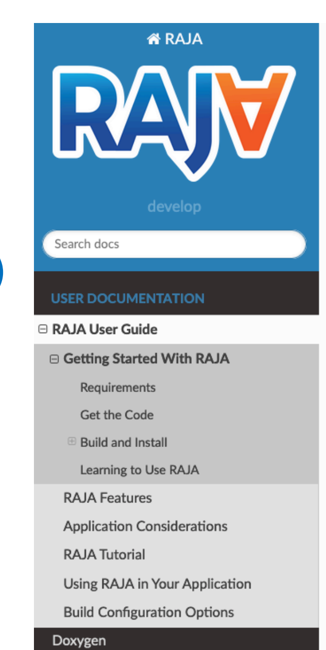
- Umpire simplifies this:

```
umpire::Allocator alloc = rm.getAllocator(”DEVICE");       ← CUDA, HIP, SYCL,
alloc.allocate(SIZE * sizeof(float))                         OpenMP Target
```

The same Umpire allocator can be used for several different backends.

URLs with RAJA information and examples...

- **RAJA User Guide:** getting started info, details about features and usage, etc. (https://readthedocs.org/projects/raja)
- **RAJA Project Template:** shows how to use RAJA and BLT in an application that uses CMake (https://github.com/LLNL/RAJA-project-template)
- **RAJA Proxy Apps:** a collection of proxy apps written using RAJA (https://github.com/LLNL/RAJAProxies)
- **RAJA Performance Suite:** a large collection of loop kernels used to assess compilers and RAJA performance (RAJA team, HPC vendors, DOE platform procurements, etc.) (https://github.com/LLNL/RAJAPerf)



Docs » RAJA User Guide » Getting Started With RAJA

Edit on GitHub

Getting Started With RAJA

This section will help get you up and running with RAJA quickly.

Requirements

The primary requirement for using RAJA is a C++14 compliant compiler. Accessing various programming model back-ends requires that they be supported by the compiler you chose. Available options and how to enable or disable them are described in Build Configuration Options. To build RAJA in its most basic form and use its simplest features:

- C++ compiler with C++14 support
- CMake version 3.14.5 or greater.

Get the Code

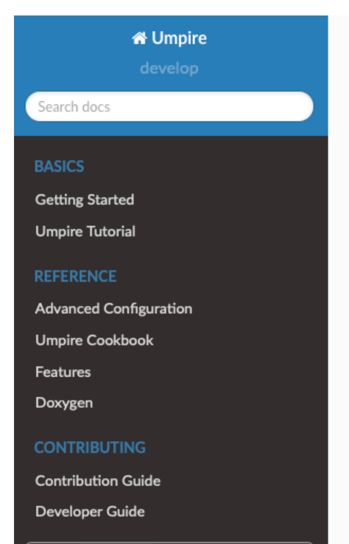
The RAJA project is hosted on GitHub. To get the code, clone the repository into a local working space using the command:

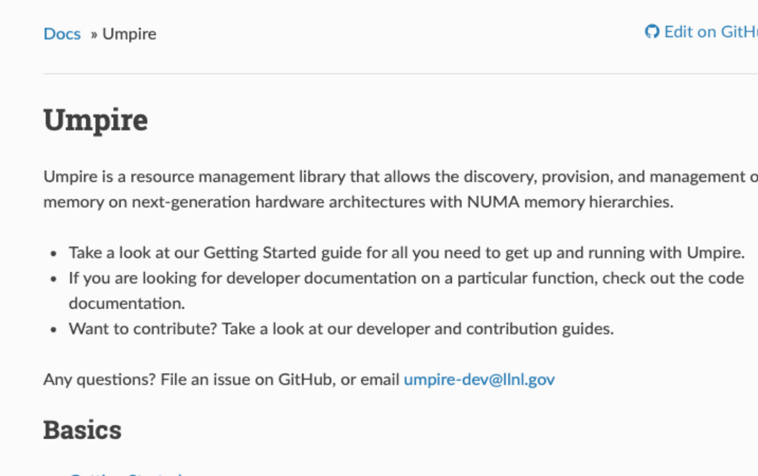
```
$ git clone --recursive https://github.com/LLNL/RAJA.git
```

All of these are linked on the RAJA GitHub project page.

We maintain user documentation, tutorials, and other code repos associated with the RAJA Portability Suite projects

- **Umpire User Guide:** getting started info, details about features & usage, tutorial materials (readthedocs.org/projects/umpire)
- **Umpire Interactive Tutorial:** interactive user tutorial using Jupyter notebooks (github.com/LLNL/umpire-interactive-tutorial)
- **CARE:** Collection of **C**HAI **A**nd **R**AJA **E**xternsions that are useful to application developers to help write portable code (github.com/LLNL/CARE)





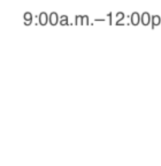
The RAJA Performance Suite and Proxy Apps are good sources of examples for RAJA usage.

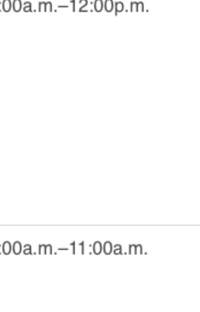
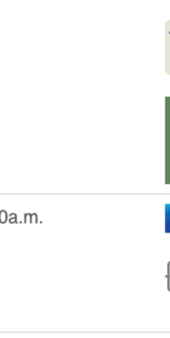
These are linked on the RAJA and Umpire GitHub projects.

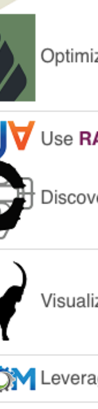
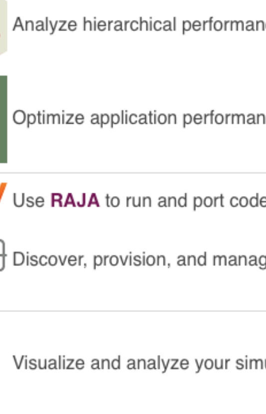
Getting started with the Tutorial…

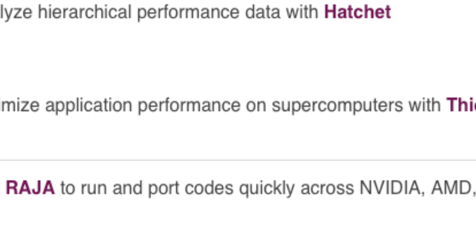
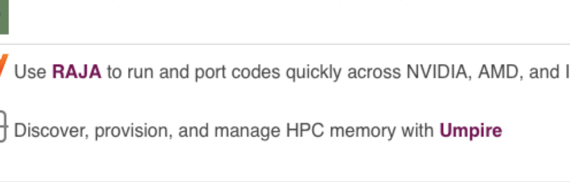
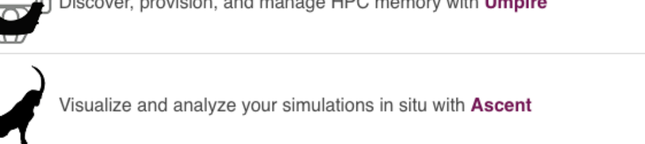
Welcome to the RADIUSS AWS Tutorial Series!

Go to: https://software.llnl.gov/radiuss/event/2023/07/11/radiuss-on-aws/ to learn more about our other tutorials and documentation!



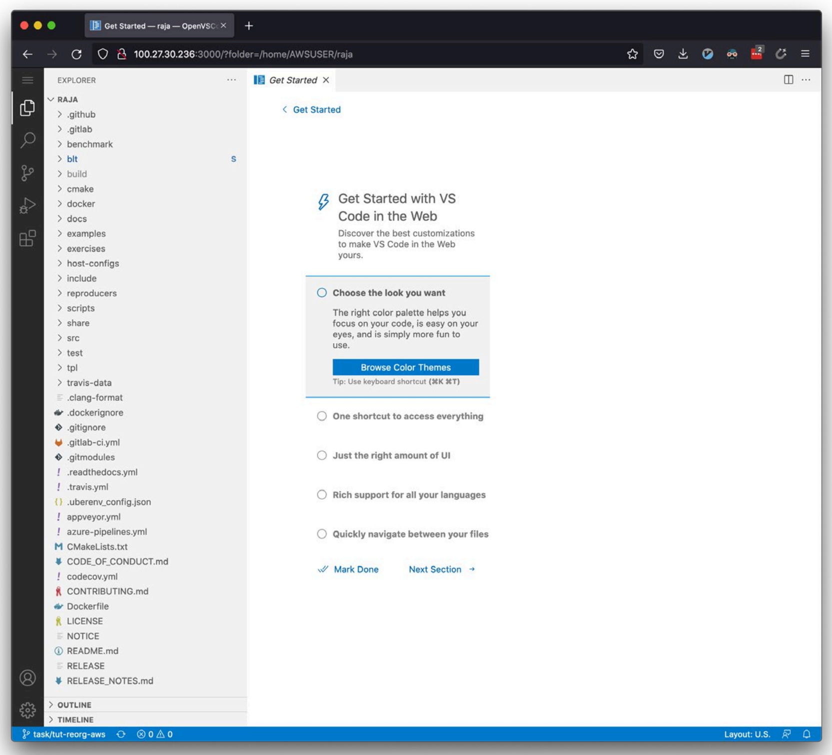



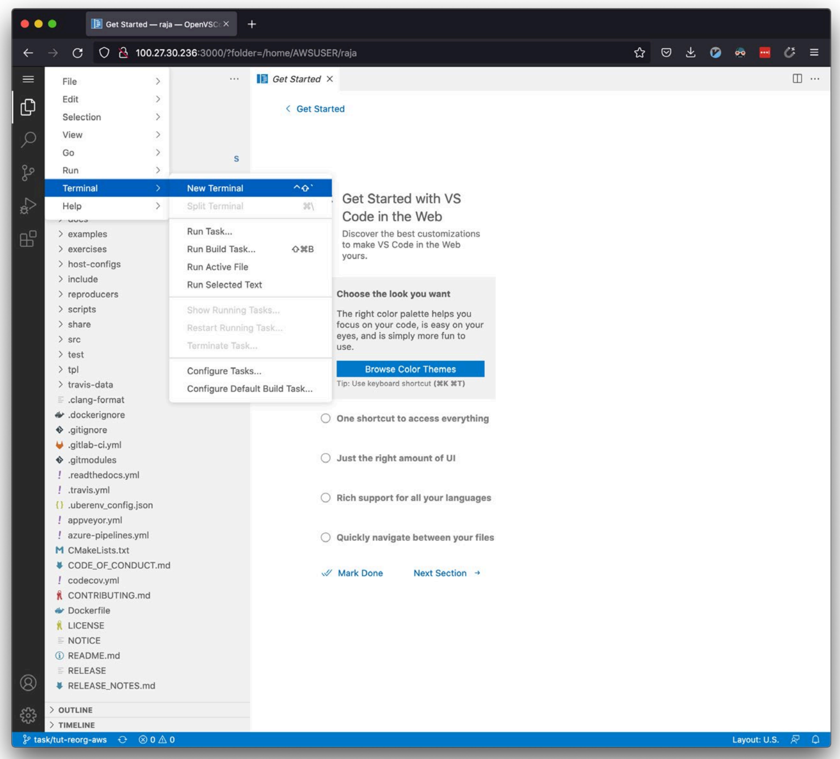
Date	Time (Pacific)	Project
August 3, 2023	9:00a.m.–11:00a.m.	Build, link, and test large-scale applications with **BLT**
August 8–9 2023	8:00a.m.–11:30a.m. both days	Learn to install your software quickly with **Spack**
August 10, 2023	9:00a.m.–11:00a.m.	Use **MFEM** for scalable finite element discretization application development
August 14, 2023	9:00a.m.–12:00p.m.	Integrate performance profiling capabilities into your applications with **Caliper** Analyze hierarchical performance data with **Hatchet** Optimize application performance on supercomputers with **Thicket**
August 17, 2023	9:00a.m.–11:00a.m.	Use **RAJA** to run and port codes quickly across NVIDIA, AMD, and Intel GPUs Discover, provision, and manage HPC memory with **Umpire**
August 22, 2023	9:00a.m.–11:00a.m.	Visualize and analyze your simulations in situ with **Ascent**
August 24, 2023	9:00a.m.–11:00a.m.	Leverage robust, flexible software components for scientific applications with **Axom**
August 29, 2023	9:00a.m.–11:00a.m.	Analyze runs of your code with **WEAVE**
August 31, 2023	9:00a.m.–11:00a.m.	Learn to run thousands of jobs in a workflow with **Flux**

Instructions for working with RAJA and Umpire on AWS



- Go to the link for your VSCode environment in your email.
- VSCode is not Visual Studios, it is just an interactive text editor.
- We will still build and run our exercises from the terminal.
- VSCode has a built-in terminal.

Instructions for working with RAJA on AWS



- To open a terminal

  Go to: ☰ → Terminal → New Terminal

Instructions for working with RAJA on AWS

- RAJA has been configure and pre-built for you in the `build` dir.
- `cd build`
- Executables live in

  `~/raja-suite-tutorial/build/bin`

- To run exercise one (from `build`)

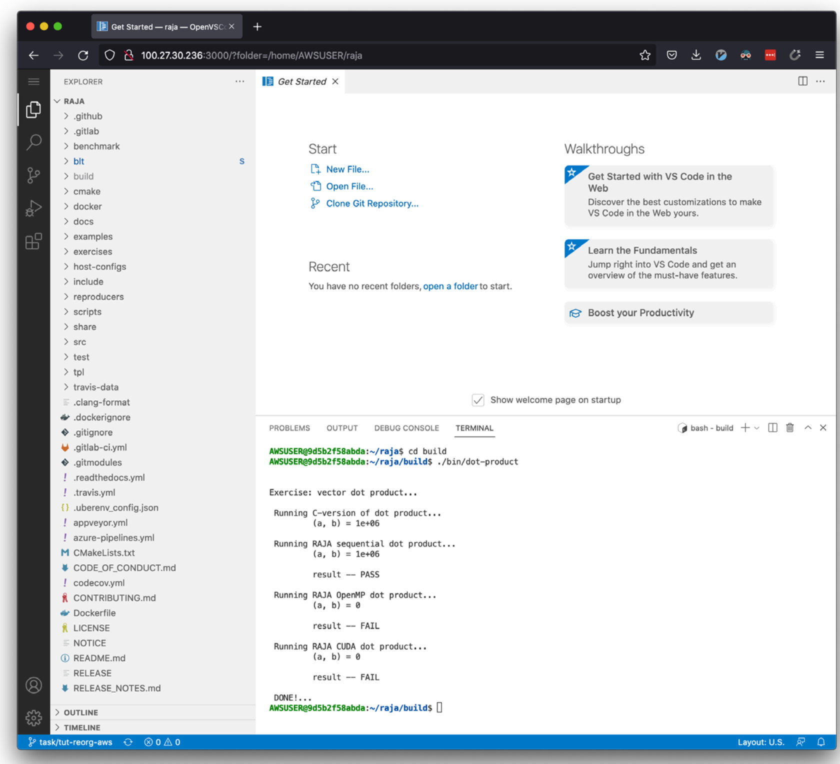
  `./bin/one`

Instructions for working with RAJA on AWS

- Editing one.cpp
- `make one`
- `./bin/one`

Disclaimer
This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

Instructions for working with RAJA on AWS

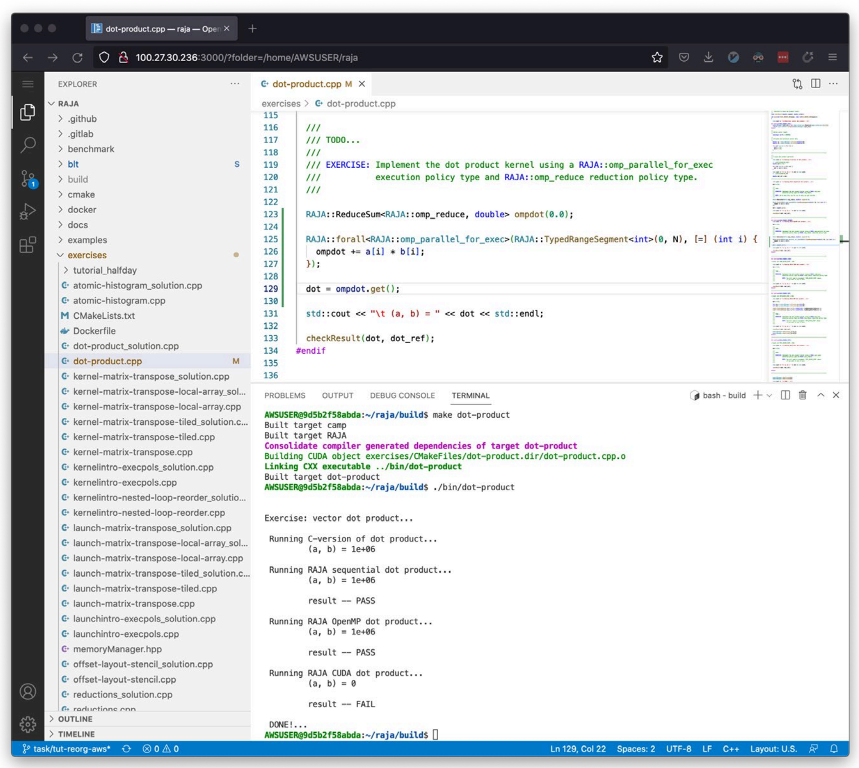


- RAJA has been configure and pre-built for you in the `build` dir.
- `cd build`
- Executables live in

  `~/raja-suite-tutorial/build/bin`
- To run exercise one (from `build`)

  `./bin/one`

Instructions for working with RAJA on AWS



- Editing one.cpp
- `make one`
- `./bin/one`