# Continuous Infrastructure Components for HPC I/O Projects

**Kenneth Casimiro[1], Houjun Tang[2], Jean Luca Bez[2], and Suren Byna[2]**

[1]University of California, San Diego, [2]Lawrence Berkeley National Laboratory

## ABSTRACT

Software development is an ever-growing industry, and the need for continuous infrastructure is required to accelerate the development and delivery of software features without compromising quality. Throughout this internship term, I developed continuous integration, documentation, software packaging, and coding standard scripts for HPC-Input-Output (I/O) projects.

## BACKGROUND INFO

- **HPC-IO**
  A GitHub organization that contains various software tools that improve storing and accessing data in an efficient way on high performance computing (HPC) systems.
- **Proactive Data Containers (PDC)**
  An object-centric API and a runtime system with a set of data object management services. These services allow placing data in the memory and storage hierarchy, performing data movement asynchronously, and providing scalable metadata operations to find data objects.
- **H5bench**
  A suite of parallel I/O benchmarks or kernels representing I/O patterns that are commonly used in HDF5 applications on high performance computing systems.
- **VOL connectors (Async I/O, Cache, Provenance)**
  The Virtual Object Layer (VOL) is an abstraction layer within the HDF5 library that redirects I/O operations.

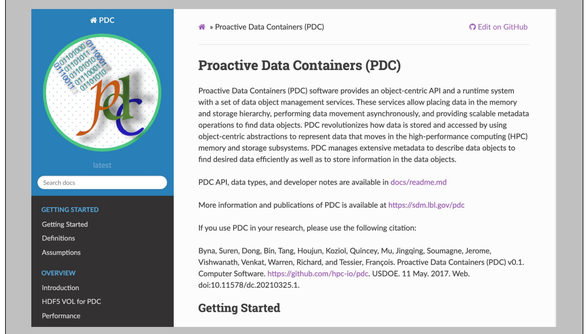
## LEARNING OBJECTIVE

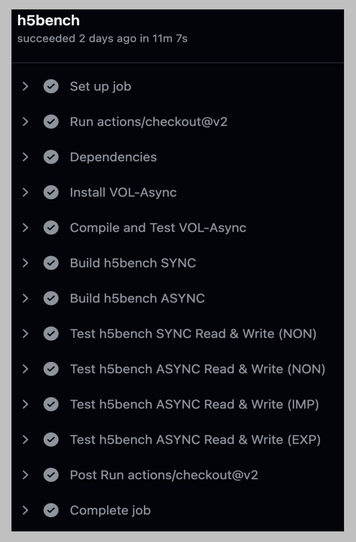
**Build and develop HPC software infrastructure for projects hosted on GitHub**

## METHODS

- **Continuous Integration (CI)**
  Automatically builds and run tests on new codes to the project to immediately surface any errors.
  Done through GitHub Actions
- **Documentation**
  Showcase the project to users and other developers. Demonstrate how to build, install, and use the downloaded software.
  Done through Sphinx and ReadtheDocs
- **Software Packaging**
  Build recipes used for Spack, a software package manager, to allow users to easily download the developed software.
- **Coding Standards**
  Automatically check and re-format the code to the specified coding style that the software developing team uses.

## RESULTS



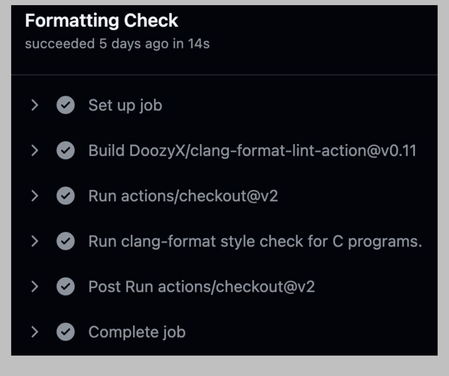




```
conflicts('%clang')
depends_on('libfabric')
depends_on('mercury')
depends_on('mpi')

root_cmakelists_dir = 'src'

def cmake_args(self):
    args = [
        self.define('MPI_C_COMPILER', self.spec['mpi'].mpicc),
        self.define('BUILD_MPI_TESTING', 'ON'),
        self.define('BUILD_SHARED_LIBS', 'ON'),
        self.define('BUILD_TESTING', 'ON'),
        self.define('PDC_ENABLE_MPI', 'ON'),
        self.define('CMAKE_C_COMPILER', self.spec['mpi'].mpicc)
    ]

    if self.spec.satisfies('platform=cray'):
        args.append("-DRANKSTR_LINK_STATIC=ON")
    return args
```




## CONCLUSION

- Produced and deployed continuous integration infrastructure to several code repositories to the Exascale Computer Project (ECP).
- Validated code changes with automated testing.
- Developed documentation for user-facing interactions.

| Project Name | Continuous Integration (CI) | Documentation | Spack Package Recipe | Coding Standard |
|---|---|---|---|---|
| PDC | ✅ | ✅ | ✅ | ✅ |
| H5bench | ✅ | ✅ | *In Progress* | ✅ |
| Async I/O | ✅ | ✅ | *In Progress* | ✅ |
| Cache | *Future Work* | *Future Work* | *Future Work* | ✅ |
| Provenance | *Future Work* | *Future Work* | *Future Work* | ✅ |

## FUTURE WORK

- Complete infrastructure components for VOL-Cache and VOL-Provenance projects.
- Develop additional CI for MacOS build/install/testing and other versions of the software.
- Explore testing coverage.

## ACKNOWLEDGMENTS

This work was prepared in partial fulfillment of the requirements of the STEM Core Program, managed by Workforce Development & Education at Berkeley Lab. This work used resources of the National Energy Research Scientific Computing Center.