Fermilab

May 8, 2025

Developing and Managing Data Acquisition Software Using Spack

Eric Flumerfelt

Spack Users Meeting

U.S. DEPARTMENT of ENERGY

Fermi National Accelerator Laboratory is managed by FermiForward for the U.S. Department of Energy Office of Science

FERMI NATIONAL ACCELERATOR LABORATORY

Agenda

- Case Study: The Mu2e Experiment at Fermilab
- Fixed Releases
- Code Development Model
- Container Deployment and CI
- Challenges
- LUNCH!

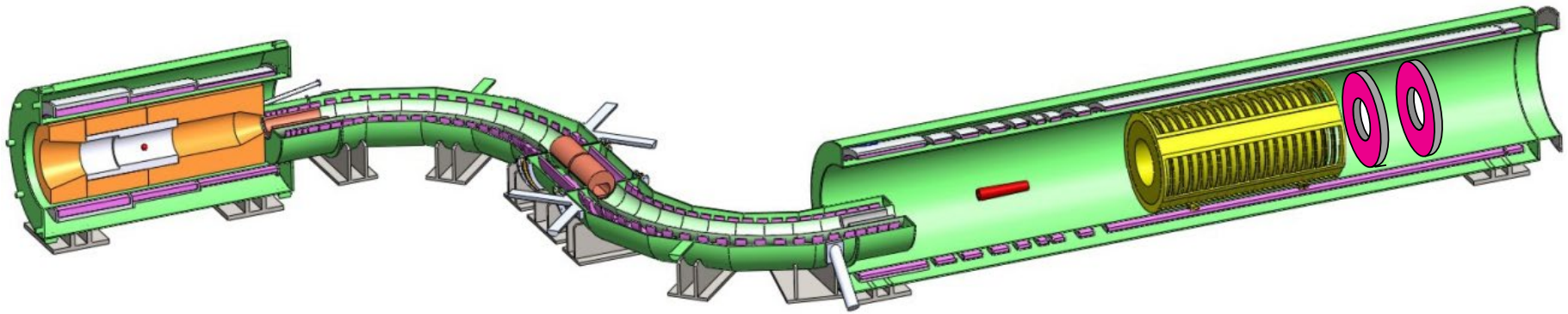
01

Case Study: The Mu2e Experiment at Fermilab

Case Study: The Mu2e Experiment at Fermilab

Trigger and DAQ (TDAQ)

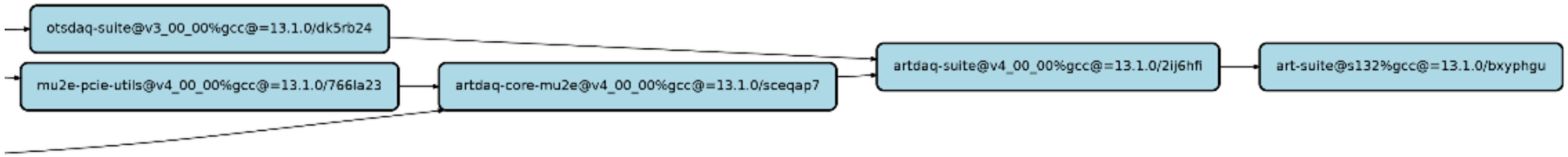
- We are supporting the Trigger and Data Acquisition (TDAQ) system for the Mu2e Experiment at Fermilab
- The TDAQ is split into several packages, and integrates code from the Mu2e Offline analysis suite
  - Data formats and data acquisition code are owned by the TDAQ group
  - Software trigger and analysis modules come from the Offline
  - DQM modules are owned by the TDAQ, but depend on analysis code from Offline as well



Case Study: The Mu2e Experiment at Fermilab

TDAQ Dependencies

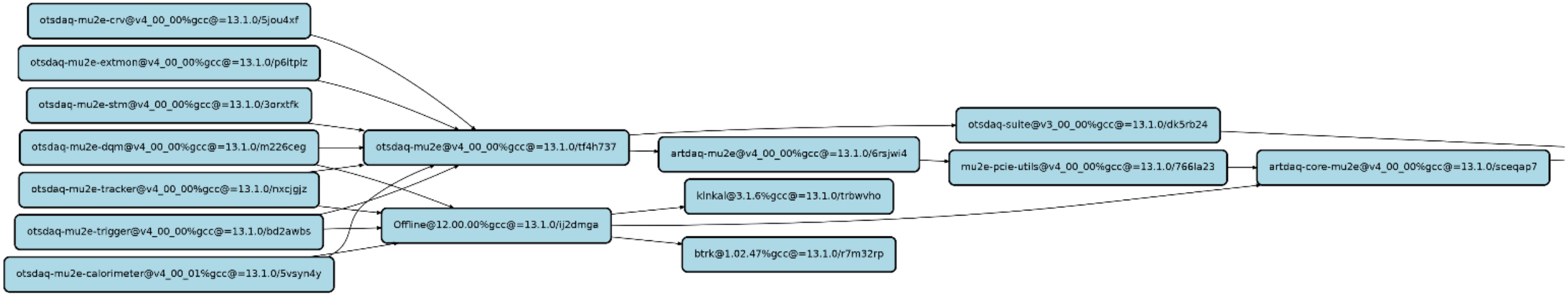
- Mu2e uses several software frameworks to implement their DAQ functionality
- *otsdaq* provides user interfaces for run control, configuration, and monitoring
- It uses *artdaq* (another Fermilab-developed project) for acquiring data from the hardware, building events, and managing the data pipeline
- *artdaq*, in turn, runs *art* analysis jobs for filtering and writing data to disk in the art/ROOT format
  - *art* is also a Fermilab product



Case Study: The Mu2e Experiment at Fermilab

Software Layers

- The Mu2e TDAQ, therefore, is composed of four well-defined software layers: Mu2e TDAQ, otsdaq, artdaq, and art
  - Each brings in additional dependencies
  - Each is versioned independently and has different release cadence



02

Fixed Releases

Fixed Releases

Bundle Packages

- We use BundlePackage to represent each layer of the Mu2e TDAQ
  - e.g. art-suite, artdaq-suite, otsdaq-suite, and mu2e-tdaq-suite
- Each bundle depends on the bundle below it, and defines a list of member packages and dependencies, with a fixed version for each
- Lower-level bundle dependencies are expressed as variants, and API changes are reflected in when= constraints on the available dependent bundle versions

```
from spack.package import *

class Mu2eTdaqSuite(BundlePackage):

"""The Mu2e TDAQ Suite, the software used for Mu2e trigger and data acquisition"""

    version("develop")
    version("v4_00_00")

    # The art-suite Dependency
    # (bundle version specified using variant not shown here)
    depends_on("art-suite +root")

    # The artdaq Dependency
    # (bundle version specified using variant not shown here)
    depends_on("artdaq-suite +db +epics")

    # The otsdaq Dependency
    # (bundle version specified using variant not shown here)
    depends_on("otsdaq-suite")

    # g4 Variant
    variant(
        "g4",
        default=False,
        description="Whether to build the G4 variant of the Offline",
    )

    # Bundle package, list packages that are part of the bundle
    with when("@v4_00_00"):
        depends_on("artdaq-core-mu2e@v4_00_00")
        depends_on("mu2e-pcie-utils@v4_00_00")
        depends_on("artdaq-mu2e@v4_00_00")
        depends_on("otsdaq-mu2e@v4_00_00")
        depends_on("otsdaq-mu2e-calorimeter@v4_00_01")
        depends_on("otsdaq-mu2e-crv@v4_00_00")
        depends_on("otsdaq-mu2e-extmon@v4_00_00")
        depends_on("otsdaq-mu2e-stm@v4_00_00")
        depends_on("Offline@12.00.00~g4", when="~g4“)
        depends_on("Offline@12.00.00+g4", when="+g4")
        depends_on("otsdaq-mu2e-tracker@v4_00_00")
        depends_on("otsdaq-mu2e-dqm@v4_00_00")
        depends_on("otsdaq-mu2e-trigger@v4_00_00")
        depends_on("mu2e-trig-config@v4_00_00")
```

Fixed Releases

Installation and Deployment

- Each bundle is installed in an independent Spack area, in an environment
  - E.g. spack_areas/artdaq-v4_00_00/spack, spack_areas/ots-v3_00_00/spack, …
- Spack upstreams.yaml are used to link these Spack areas, so that one installation of a lower-level bundle can be re-used multiple times
  - e.g. art-suite@s132 can be installed once and referenced by multiple artdaq-suite versions
  - Installation scripts ensure that upstreams are added in correct order
- To use a fixed release of the software, a user would source the setup-env.sh script and activate the relevant environment
  - i.e. source spack_areas/ots-v3_00_00/setup-env.sh && spack env activate ots-v3_00_00

03

Code Development Model

Code Development Model

- Due to the modular nature of the code, changes are often spread among multiple packages
  - For example, a data format change might happen in artdaq-core-mu2e to accompany a change to a readout module in artdaq-mu2e, with updated analysis in Offline
- Also, development often crosses bundle boundaries
  - If a bug is found in the artdaq readout, all intermediate packages must be checked out and rebuilt
- An attempt was made to use “spack develop”, however the full rebuilds of dependencies triggered by every change made the system impossible to work with
  - Developers could not wait an hour to test each incremental change
- The Spack MPD extension allows for incremental builds across multiple packages
  - Kyle Knoepfel and Marc Paterno introduced MPD yesterday

Code Development Model

- MPD creates an environment using dependency information from the packages under development
- We constrain our package dependencies so that this created environment uses the same dependency versions and variants as the fixed release, with the exception of those packages currently under development
- Putting these constraints in place also means that we don’t need to distribute solved release environments; the concretization result should be stable for the dependencies we care about

```
class OtsdaqMu2e(CMakePackage):
     """The toolkit currently provides functionality for data transfer,
     event building, event reconstruction and analysis (using the art analysis
     framework), process management, system and process state behavior, control
     messaging, local message logging (status and error messages), DAQ process
     and art module configuration, and the writing of event data to disk in ROOT
     format."""“

     homepage = https://mu2e.fnal.gov
     url = https://github.com/Mu2e/otsdaq-mu2e/archive/refs/tags/v1_02_02.tar.gz
     git = https://github.com/Mu2e/otsdaq-mu2e.git

     license("BSD")

     version("develop", branch="develop", get_full_repo=True)
     version("v4_00_00", commit="af80e7917f73dcf06bb40465609be82827f4d26a")
     version("v3_04_00", commit="7f1e42edb3bc1590e6ea5add941aafdaf9222bc2")

     variant(
         "cxxstd",
         default="20",
         values=("14", "17", "20"),
         multi=False,
         description="Use the specified C++ standard when building.",
     )

     depends_on("cetmodules@3.26.00:", type="build")

     depends_on("otsdaq@:v2_99_00", when="@:v3_99_00")
     depends_on("otsdaq@v3_00_00:", when="@v4_00_00:")
     depends_on("otsdaq-epics@:v2_99_00", when="@:v3_99_00")
     depends_on("otsdaq-epics@v3_00_00:", when="@v4_00_00:")
     depends_on("otsdaq-suite")

     depends_on("artdaq-mu2e@:v3_99_00", when="@:v3_99_00")
     depends_on("artdaq-mu2e@v4_00_00:", when="@v4_00_00:")
```

04

Container Deployments and CI

Container Deployments and CI

CVMFS Deployment

- We use the CVMFS distributed file system, which provides read-only access over HTTP to our software
- The fixed-release Spack areas with their installed bundle environments are built using automatic scripts in containers that mirror their final location in the CVMFS filesystem
- These are then copied to CVMFS distribution servers
- The next step is to create custom Docker images which reference the Spack areas on CVMFS and create development areas
  - Keeping the Spack installs in CVMFS allows for small Docker images to be made that nevertheless contain the complete software stack
  - CVMFS provides a Github Action recipe for installing and accessing CVMFS from a Github Actions job

Container Deployments and CI

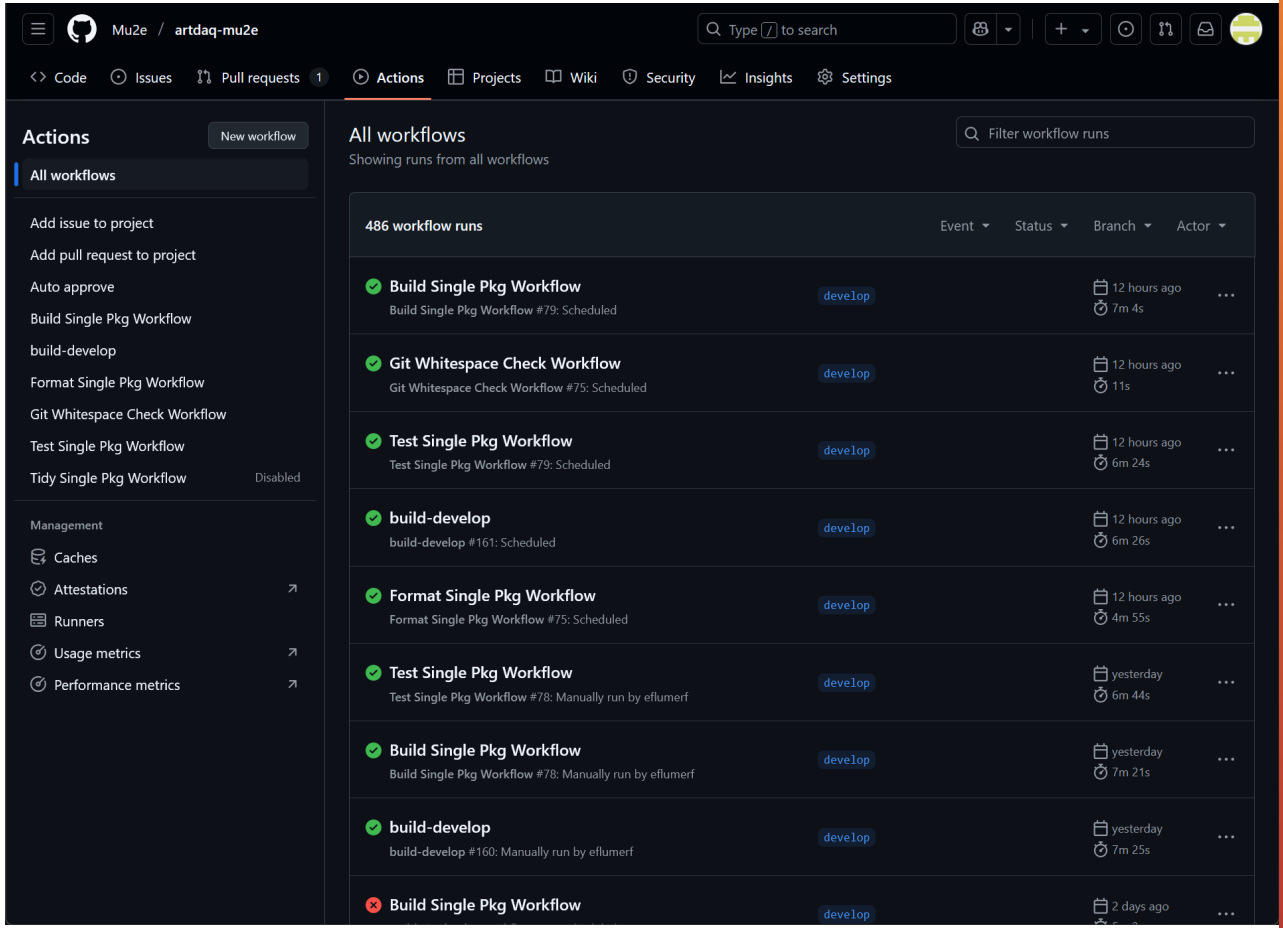
Docker Container

```
FROM eflumerf/alma9-spack:latest AS intermediate

SHELL ["/bin/bash", "-c"]

ARG MU2E_AREA=mu2e-tdaq-v3_03_01
ARG OTS_AREA=ots-v3_00_00
ARG ARTDAQ_AREA=artdaq-v4_00_00
ARG ART_AREA=art-suite-s132

WORKDIR /opt/mu2edaq

ADD https://raw.githubusercontent.com/Mu2e/otsdaq_mu2e/develop/tools/mu2e-quick-spack-start.sh /opt/mu2edaq/mu2e-quick-spack-start.sh

RUN mkdir -p /cvmfs/fermilab.opensciencegrid.org/products/artdaq/spack_areas/$ART_AREA
COPY spack_areas/$ART_AREA /cvmfs/fermilab.opensciencegrid.org/products/artdaq/spack_areas/$ART_AREA

RUN mkdir -p /cvmfs/fermilab.opensciencegrid.org/products/artdaq/spack_areas/$ARTDAQ_AREA
COPY spack_areas/$ARTDAQ_AREA /cvmfs/fermilab.opensciencegrid.org/products/artdaq/spack_areas/$ARTDAQ_AREA

RUN mkdir -p /cvmfs/fermilab.opensciencegrid.org/products/artdaq/spack_areas/$OTS_AREA
COPY spack_areas/$OTS_AREA /cvmfs/fermilab.opensciencegrid.org/products/artdaq/spack_areas/$OTS_AREA

RUN mkdir -p /cvmfs/fermilab.opensciencegrid.org/products/artdaq/spack_areas/$MU2E_AREA
COPY spack_areas/$MU2E_AREA /cvmfs/fermilab.opensciencegrid.org/products/artdaq/spack_areas/$MU2E_AREA

RUN chmod +x /opt/mu2edaq/mu2e-quick-spack-start.sh && \
     ./mu2e-quick-spack-start.sh --develop --dev-only --all-packages --no-kmod --arch linux-almalinux9-x86_64_v3 \
                                 --upstream /cvmfs/fermilab.opensciencegrid.org/products/artdaq/spack_areas/$MU2E_AREA \
                                 --upstream /cvmfs/fermilab.opensciencegrid.org/products/artdaq/spack_areas/$OTS_AREA \
                                 --upstream /cvmfs/fermilab.opensciencegrid.org/products/artdaq/spack_areas/$ARTDAQ_AREA \
                                 --upstream /cvmfs/fermilab.opensciencegrid.org/products/artdaq/spack_areas/$ART_AREA
RUN source setup-env.sh && spack env activate tdaq-develop

RUN rm -rf /cvmfs/fermilab.opensciencegrid.org/products

FROM eflumerf/alma9-spack:latest

COPY --from=intermediate /opt/mu2edaq /opt/mu2edaq

ENTRYPOINT ["/bin/bash", "-l", "-c" ]
```

Container Deployments and CI

Github CI

- With a full development area deployed in a container and leveraging the caching ability of Github Actions, a full deployment of the software in a ready-to-build state on a fresh Github Actions runner takes 2-4 minutes
- Our CI jobs include building all packages in a MPD area at the tip of the develop branches, building just one package against the fixed release at a specific commit, running defined unit tests, and code-checking tools such as clang-format and clang-tidy
- The jobs are fully integrated into the Github platform, and run on all pull requests opened, as well as on a nightly basis



05

Challenges

Challenges

Spack Gripes

- Development of Spack
  - DAQ installations require stability above all else
  - Using fixed versions of Spack, package repositories, and development tools (spack-mpd)
- Instability of Concretization
  - Despite all efforts to express dependencies in bundle packages, some packages are rebuilt in multiple environments along the “fixed release” path. *llvm* is a good example of a package that is often-rebuilt and painful to rebuild.
    - include_concrete should help with this, but we’ve had issues using it in development environments
  - We have had multiple instances where concretization succeeds but should not have, due to conflicts() directives in package recipes being ignored (possibly a result of an old default for a variant with the sticky flag)

Challenges

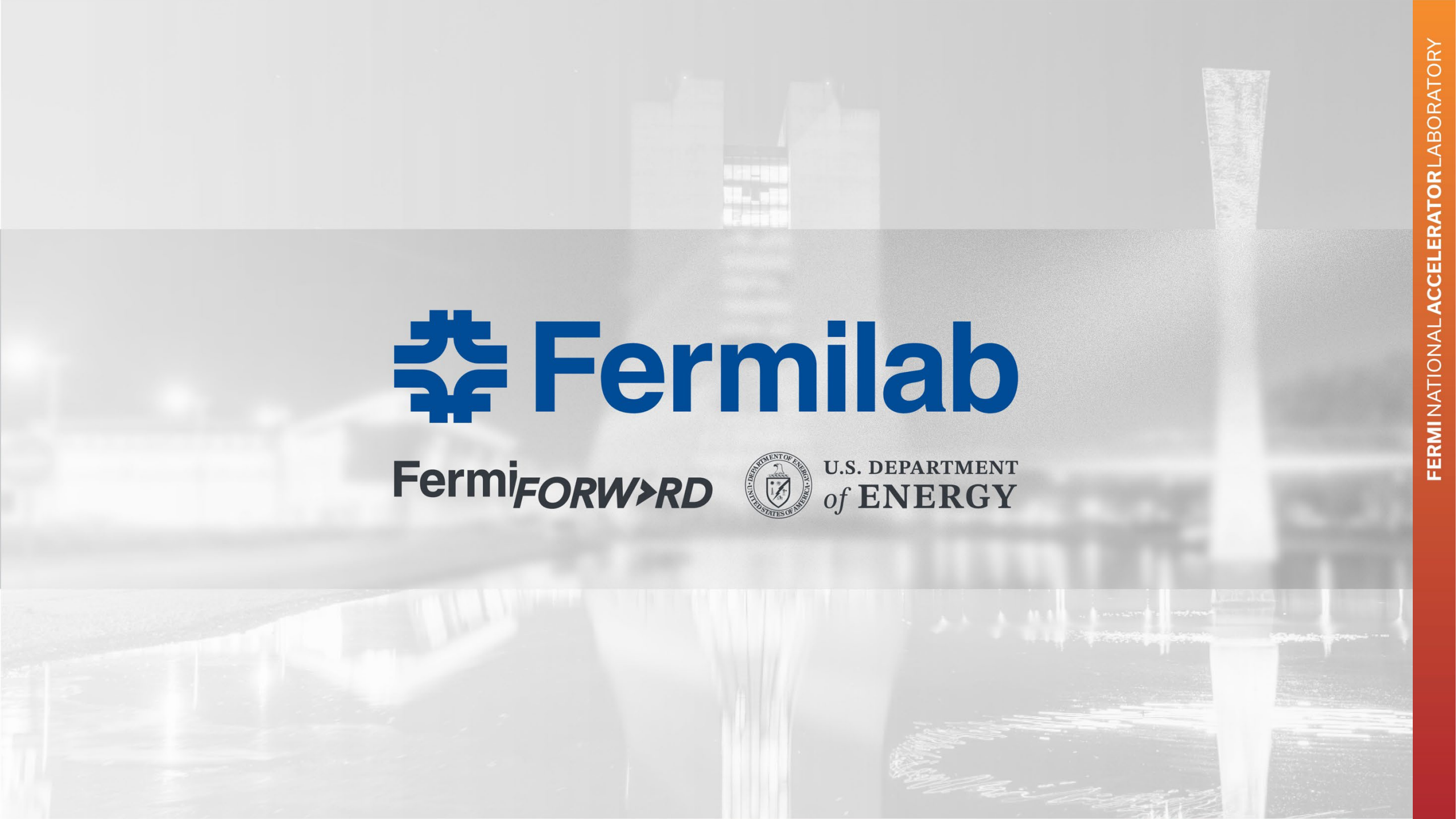
- Automation
  - The package management and build should be transparent to end users
  - We need to be able to quickly deploy the software on multiple systems in many locations
    - Test stands located at Fermilab, in Italy; Github CI containers
- End User Education
  - Spack is a new system for us, and users are not familiar with failure scenarios and recovery
  - When concretization fails, error messages are often cryptic and do not give guidance on how to fix the problem
  - Hard to tell when a failure is due to user error, a mistake in packaging, or simply missing a dependency spec
- Usage
  - Activation of an environment can take some time, this has been avoided by creating “run-time environment” scripts which represent the environment delta before/after setup-env and activating the environment (using “declare -x”)

06

Conclusions

Conclusions

- We have successfully deployed our software stack and automated the creation of both fixed releases which we intend to use in experiment operations, and development areas for fixing issues and implementing new features
- Mu2e is freezing their software deployment and starting to take initial data
- Development will continue
  - Updating to Spack 1.0
  - Fixing issues with include_concrete and minimizing package rebuilds
  - Investigating ways to minimize CI container size and speed up Github CI


Fermilab
FermiFORWARD
U.S. DEPARTMENT of ENERGY
FERMI NATIONAL ACCELERATOR LABORATORY