

# Calculation Engine

Andrew Manning

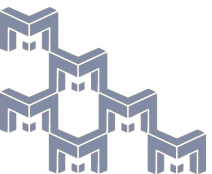
*National Center for Supercomputing Applications*



# What is the calculation engine (CE)?

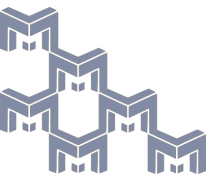
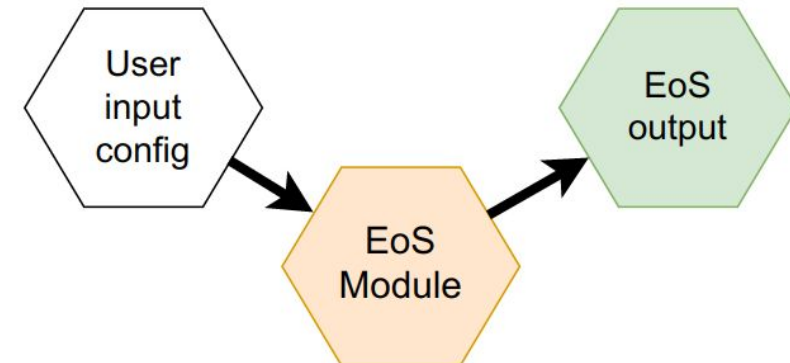
Logically, the calculation engine (CE) is a software system that

- manages user-submitted processings **jobs**
  - a **job** is the execution of one or more modules in a **workflow**.
- manages the movement of data between subsystems
- serves data for download
- enforces access control
- organizes and stores information in a structured database
- tracks data provenance



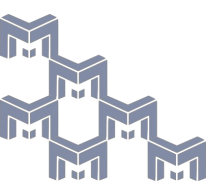
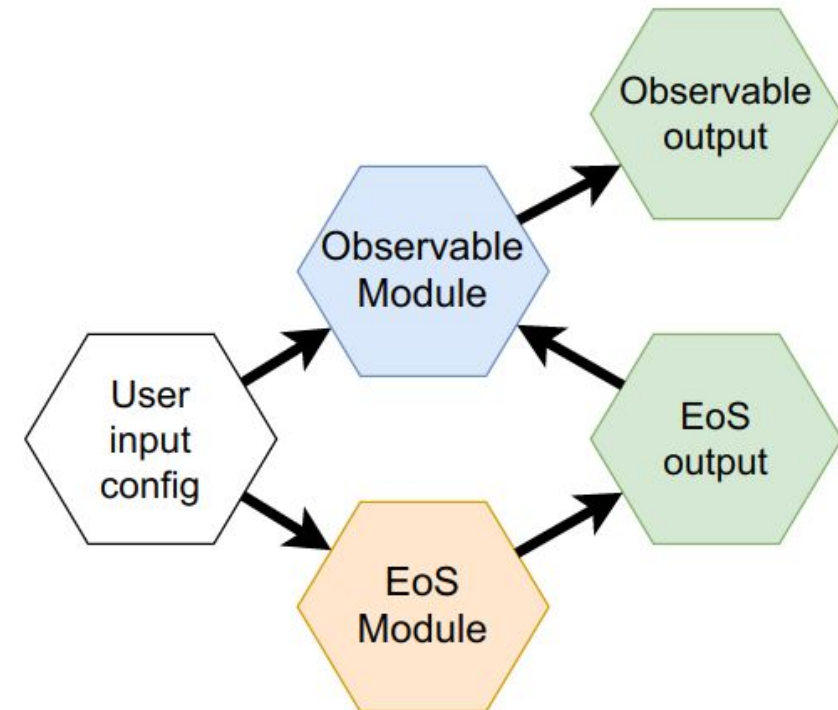
A workflow is a processing sequence of one or more modules, in which output of one module is used as the input to the subsequent modules.

The simplest case is a single module execution such as calculating an equation of state.



A workflow is a processing sequence of one or more modules, in which output of one module is used as the input to the subsequent modules.

The most common multi-module sequence will generate an EoS and then calculate some physical observables using a second module.

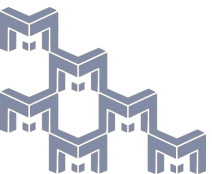
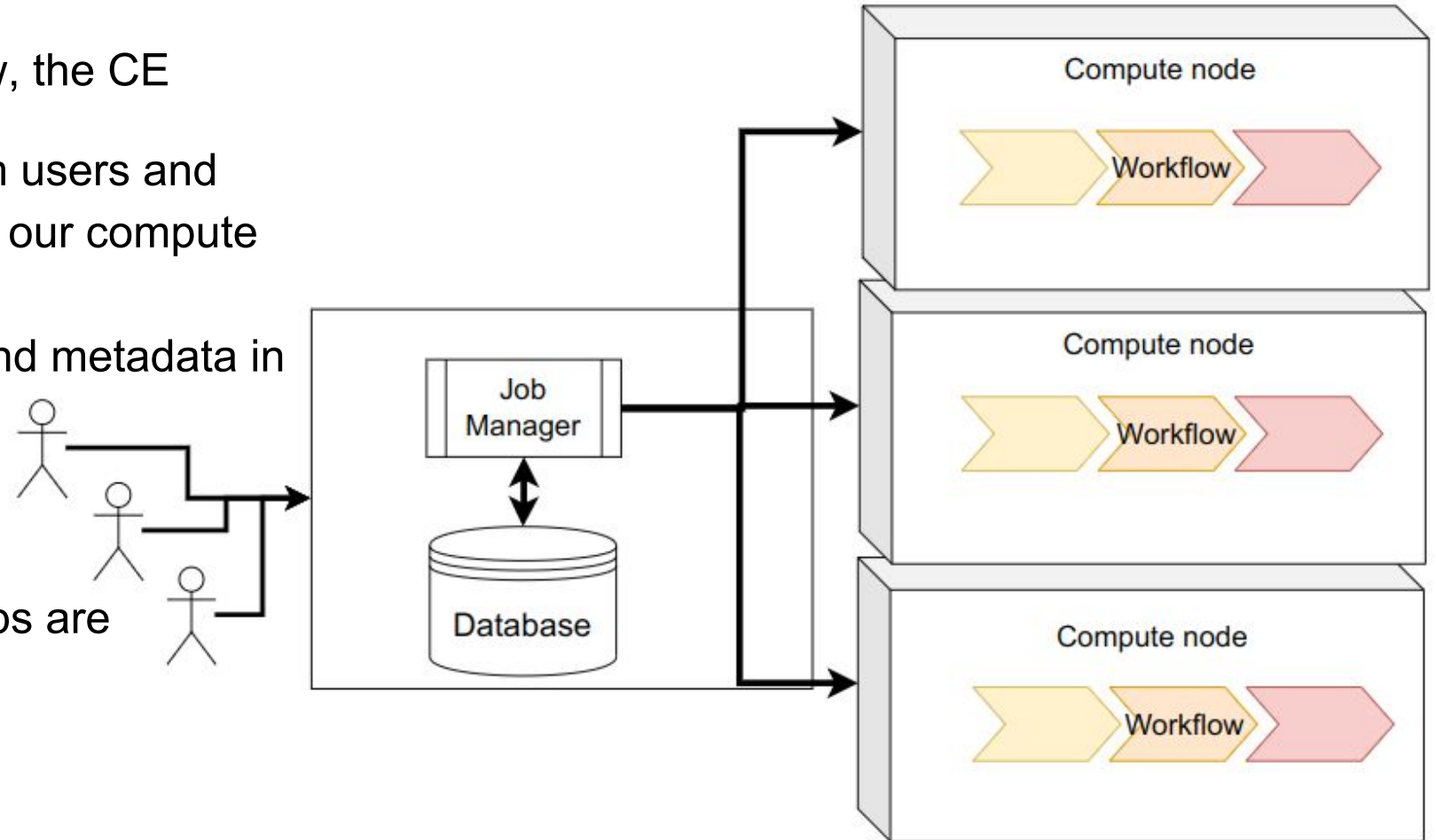


From the perspective of data flow, the CE

- takes the input requests from users and
- launches processing jobs on our compute nodes,
- keeping track of job status and metadata in a database,

and handling things like

- notifying users when their jobs are complete
- returning output data



Architecturally, the CE is comprised of several logical components.

Some of these components may be implemented using third-party solutions, custom code developed at NCSA, or blends of both.

Keycloak,  
Python/Django

Identity and Access  
Management

NGINX

HTTP webserver

Parsl,  
Python/Django

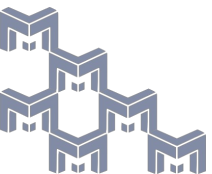
Job Manager

Python/Django

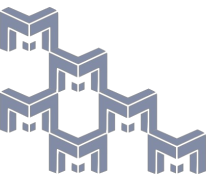
Workflow executor

PostgreSQL,  
MariaDB

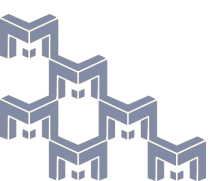
Database



- **Cyberinfrastructure** consists of systems, data and information management, advanced instruments, visualization environments, and people, **all linked together by software and advanced networks to improve scholarly productivity** and enable knowledge breakthroughs and discoveries not otherwise possible.
- In computer programming, a **software framework** is an abstraction in which software, providing generic functionality, can be **selectively changed by additional user-written code**, thus providing application-specific software. It **provides a standard way to build and deploy applications** and is a **universal, reusable software environment** that provides particular functionality as part of a larger software platform to facilitate the development of software applications, products and solutions.



- The CE is written in Python. We all love Python.
- We have limited developer resources, so we want to maximize productivity with judicious choice of framework
- There are many Python frameworks. We need one that provides
  - Plugin for OIDC authentication (use our MUSES single sign-on system)
  - Session management (cookies, tokens)
  - Object-relational mapping: is a technique that allows you to manipulate data in a relational database using object-oriented programming.
  - Support for rapid REST API prototyping, driven by Python-defined data models





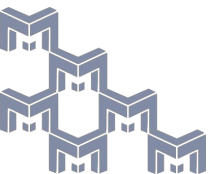
Modules must be containerized:

- Independent module development teams need freedom to choose their software environment
- In production, the CE must run the modules in containers for compatibility with arbitrary compute nodes



Containers are a challenge for researchers who want to run the CE locally:

- They must install Docker on their workstation
- If we want to reduce the complexity of their local software environment (installed libraries, packages, and shell config) required to run the CE, then it must run as a container, but then the CE must implement a Docker-in-Docker system, which is not very portable between operating systems.
- Conda or Docker Compose



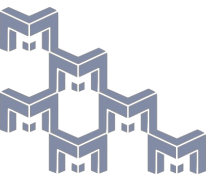
We will definitely run the CE as a hosted service via our Kubernetes cluster, and researchers can use this service according to some access control scheme we have yet to devise.

## Advantages

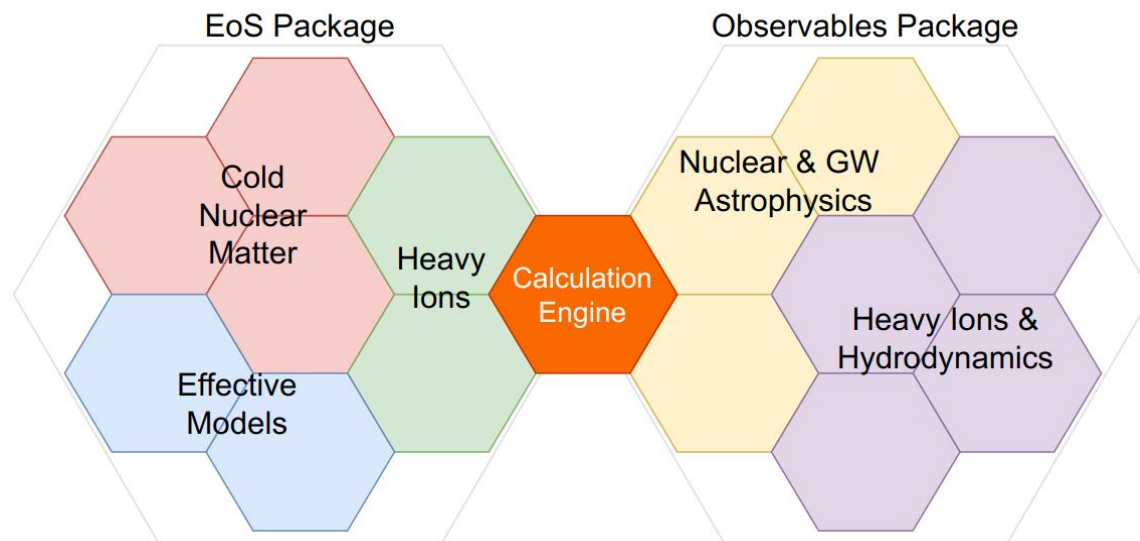
- No need for scientists to install software
- Good publicity for MUSES and increased community investment in our success

## Disadvantages

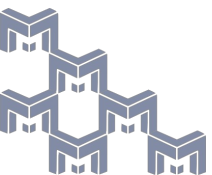
- Computing resources are limited
- Increased responsibility for operational stability and uptime
- Immediately confronted with issues related to data provenance and reproducibility



**Integration** is the process of adapting a code module to comply with the MUSES framework we have defined to allow the CE to execute workflows.



- **Containerization**
  - Modules must build and push container images that the CE can download and execute via Docker.
- **Manifest**
  - Each module defines a so-called **manifest** file, declaring information about itself in a standard format expected by the CE
- **API specification**
  - The inputs and outputs of a module must be declared in machine-level detail using the OpenAPI standard format

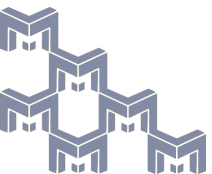


Declare a reference name for the module and the git URL and revision for the modules' manifests.

```
## List of registered modules, including some
## metadata and where to locate their manifests

modules:
- name: "test_module_1"
  path: "/modules/test-module-1"
  url: ""
  targetRevision: ""
- name: "test_module_2"
  path: ""
  url: "https://gitlab.com/nsf-muses/template-module/template-module"
  targetRevision: "72ab988f7cf2603880d7647c318d63ba22f789a6"
```

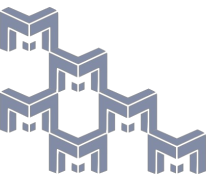
[See this forum post for more details.](#)



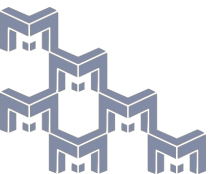
The manifest tells the CE the information needed to execute the module and wire up the inputs and outputs.

```
name:
  short_name: test_module_1
  display_name: Test Module 1
authors:
  - T. Andrew Manning
command:
  - 'python3'
  - '/src/main.py'
image:
  registry: "registry.gitlab.com"
  repo: "nsf-muses/calculation-engine/test-module-1"
  tag: "1.0.0"
  source:
    url: "https://gitlab.com/nsf-muses/calculation-engine"
    git: "https://gitlab.com/nsf-muses/calculation-engine.git"
    path: "docker/modules/test-module-1/Dockerfile"
    targetRevision: "0.5.2"
docs:
  path: "docs/src"
```

[See this forum post for more details.](#)  
[Example manifest](#)



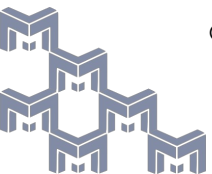
```
## Labels must be unique across both inputs and outputs
inputs:
  - label: config
    ## The name "config" is a reserved word used by the Calculation Engine to
    ## identify where to read the exit code
    description: "Configuration of the module runtime options"
    path: "/input/config.yaml"
    api:
      url: ""
      targetRevision: ""
      path: api/input/config.yaml#/components/schemas/Configuration
outputs:
  - label: status
    ## The name "status" is a reserved word used by the Calculation Engine to
    ## identify where to read the exit code
    description: "Module execution status"
    path: "/output/status.yaml"
    api:
      url: ""
      targetRevision: ""
      path: api/output/status.yaml#/components/schemas/Status
  - label: equation_of_state
    description: "Calculated equation of state data"
    path: "/output/eos.yaml"
    api:
      url: ""
      targetRevision: ""
      path: api/output/Equation_Of_State.yaml#/components/schemas/Equation_Of_State
```



Workflows define the available execution flows and how module inputs and outputs are connected

```
workflows:
- name: "eos_to_obs"
  description: "Simulates calculating physical observables from an equation of state."
  ## In task definitions, the module names must match registered modules. The order of
  ## the tasks sequence defines the workflow order. Only piped input/output connections
  ## are defined here. For now only single output-to-input ("one to one") pipes are supported;
  ## supported; output-to-multiple-inputs ("one to many") is not supported.
  tasks:
- module: "test_module_1"
  inputs: []
  ## The labels must match a file description in the module manifest.
  outputs:
- label: "equation_of_state"
  ## The "pipe" parameter is defined at the workflow level to indicate which
  ## outputs connect to which inputs in the subsequent task.
  pipe: "EoS"
- module: "test_module_2"
  inputs:
- label: "eos"
  pipe: "EoS"
  outputs: []
```

[See this forum post for more details.](#)

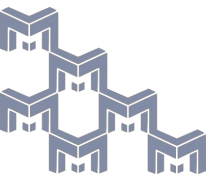


A database will be necessary for operational logistics like job management, but it also must

- store the metadata for data provenance and immutable references
- store the actual output data where feasible, or immutable links to archive files?

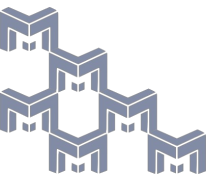
Modules may produce files appropriate for their needs (HDF, ZIP, YAML, etc), and the CE must know how to serve these as well as ingest the contents as needed for future analysis or interpolation (UTK EoS)

The YAML-formatted output of a module need not directly contain the output data itself; instead, it could specify a URI to the data

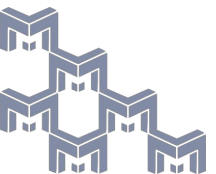


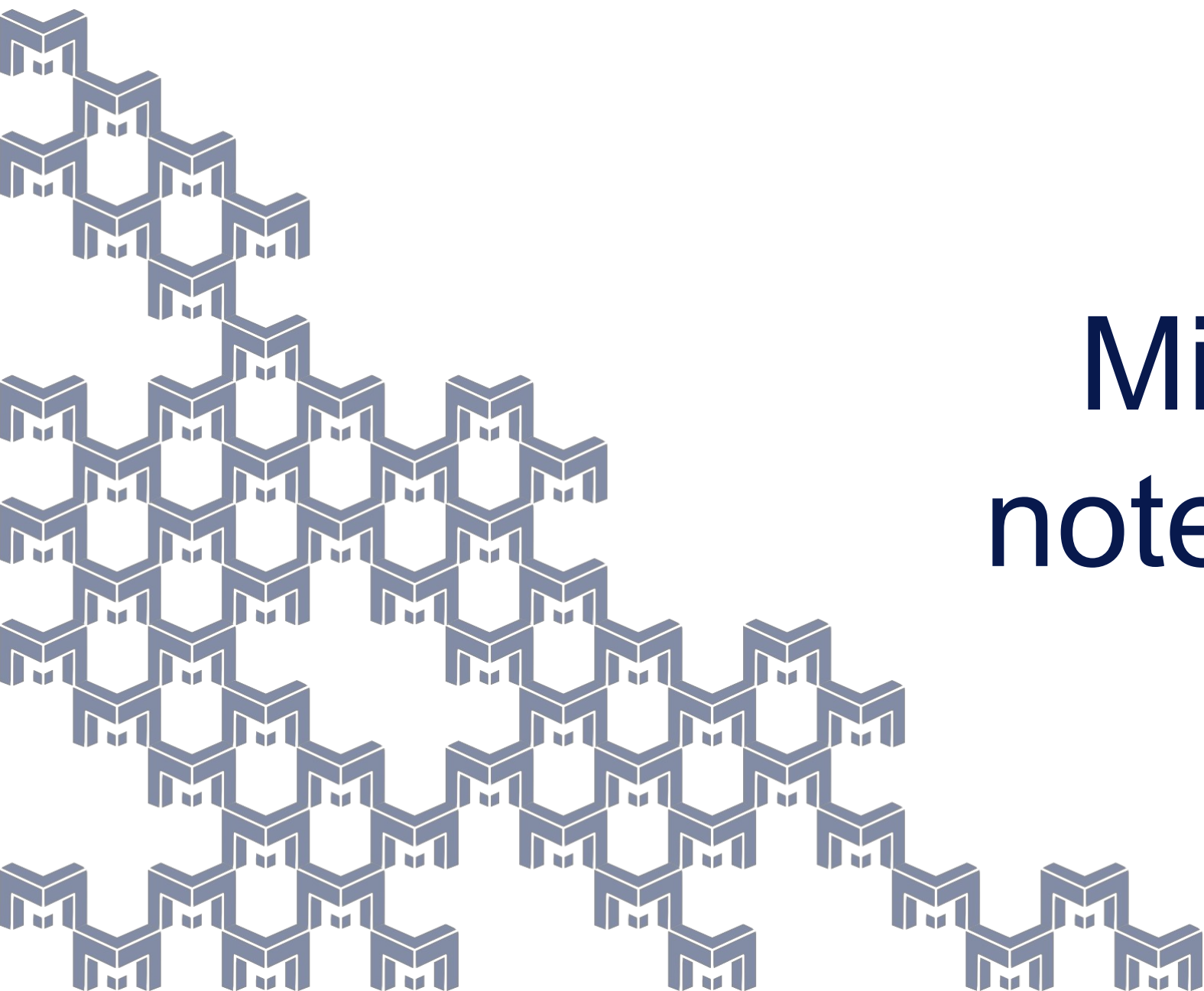


- When researchers begin using the hosted MUSES service, they will be generating data that will yield published scientific results, which demands a measure of reproducibility
- We must design and implement a way to reference a self-consistent and immutable
  - Self-consistent means that each datum is associated with a particular CE version (which captures the individual versions of modules)
  - Immutable means that there is a method of **retroactively accessing data or at least metadata** about the processing job suitable for scientific citation



- [https://forum.musesframework.io/tag/calculation\\_engine](https://forum.musesframework.io/tag/calculation_engine)
- <https://gitlab.com/nsf-muses/calculation-engine>





# Miscellaneous notes and ideas













- Visit <https://musesframework.io/account> to review and update your MUSES collaborator profile information
- The new web form replaces our original system where new collaborators would edit a table of information in [a wiki-style post on the forum](#).
- This system will make it easier to utilize the data to generate reports for NSF or generate author lists when preparing scientific papers.
- Profile data is now stored in our central identity system (Keycloak) as user attributes

MUSES // Account

T. Andrew Manning ▾

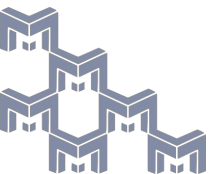
## User profile

Update your MUSES account profile information here. You may add multiple affiliations and multiple working groups. Surround acknowledgements and author name text with `$` character to indicate LaTeX syntax.

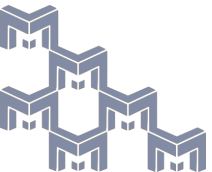
Name	T. Andrew Manning	
Email	<code>manninga@illinois.edu</code>	
Author name	<code>Timothy Andrew Manning</code>	 
Acknowledgements	<code>\$This work was supported in part by the National Science Foundation \$\left( NSF \right)\$ within the framework of the MUSES collaboration, under grant number OAC-2103680.\$</code>	 
Website	<code><a href="http://andrew.reticu.li/">http://andrew.reticu.li/</a></code>	 
Role	senior investigator	 
Affiliation	National Center for Supercomputing Applications, University of Illinois Urbana-Champaign, Urbana, IL 61801, USA	 
Working Group	cyberinfrastructure	 

Timothy Andrew Manning

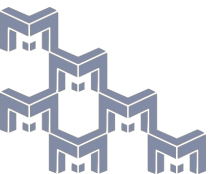
Update author name



- When module registration is updated, a build pipeline can automatically detect this and build a new tagged image (GitLab CI, GitHub Workflows)
- Do we want to use git tags for image tags? Or directly use the git commit hash?



- Develop the pipeline for compiling and publishing the OpenAPI specs for the registered modules, somewhere under <https://musesframework.io/docs/> perhaps. ([example from DES](#))
- The module manifest already includes a field for specifying where human-centric module documentation resides, so we could also compile the full documentation at the same time.
- This would be good publicity as well as being actually useful to researchers.



How can we leverage our dedicated JupyterHub service?

- Docker image build iterations, code compilation that takes a long time
- Consistent software environments for module development teams
- Real-time, ad-hoc code and data sharing

You can sync your JupyterHub data to your local machine automatically via Nextcloud desktop client

