NIST Interagency Report
NIST IR 8520

Report from the 2nd International Workshop on FAIR Containerized Computational Software

Peter Bajcsy
Nathan Hotaling

This publication is available free of charge from:
https://doi.org/10.6028/NIST.IR.8520

NIST NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
U.S. DEPARTMENT OF COMMERCE

NIST Interagency Report
NIST IR 8520

Report from the 2nd International Workshop on FAIR Containerized Computational Software

Peter Bajcsy
Software and Systems Division
Information Technology Laboratory (ITL)

Nathan Hotaling
National Center for Advancing Translational Sciences (NCATS)
National Institutes of Health (NIH)

This publication is available free of charge from:
https://doi.org/10.6028/ NIST.IR.8520

April 2024

U.S. Department of Commerce
Gina M. Raimondo, Secretary

National Institute of Standards and Technology
Laurie E. Locascio, NIST Director and Under Secretary of Commerce for Standards and Technology

NIST IR 8520
April 2024

Publication History
Approved by the NIST Editorial Review Board on 2024-04-25

How to Cite this NIST Technical Series Publication
Bajcsy P, Hotaling N (2024) Report from the 2nd International Workshop on FAIR Containerized Computational Software. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Interagency Report (IR) NIST IR 8520. https://doi.org/10.6028/NIST.IR.8520

Author ORCID iDs
Peter Bajcsy: 0000-0002-6968-2615
Nathan Hotaling: 0000-0001-8423-6464

Contact Information
peter.bajcsy@nist.gov

Abstract

Purpose: The National Institute of Standards and Technology (NIST) is evaluating and improving the specification to achieve containerized computational software interoperability. Adherence to a specification for *Findable, Accessible, Interoperable,* and *Reusable (FAIR)* Containerized Computational Software (FAIR–CCS) enables better reuse of containerized tools in complex data analyses by chaining tools into computational workflows. NIST requested information from the community on approaches to achieving the interoperability of containerized software, designing a container manifest file that meets the community's needs, and lowering the barrier for constructing such a manifest file. Responses to this Request for Information (RFI) informed a possible revision of the current approach to achieving FAIR–CCS via a manifest file, the entries in the current manifest file specification of FAIR–CCS, and the current tools that aim at automating adherence to the FAIR–CCS manifest specification.

Methods: NIST and the National Center for Advancing Translational Sciences (NCATS) of the National Institutes of Health (NIH) hosted the 2nd International Workshop on FAIR Containerized Computational Software on December 5-7, 2023, to discuss the responses to the RFI. *The main goal* for the virtual workshop was to establish a community consensus on creating interoperable containerized computational tools that can be chained into scientific workflows/pipelines and executed over extensive image collections regardless of the cloud infrastructure components.

Results: This NIST report summarizes the 2nd International Workshop on FAIR Containerized Computational Software. The workshop was attended by 111 out of 160 registered participants over three days. Each day was structured into one hour of general introductory presentations, two hours of breakout sessions discussing relevant topics the FAIR–CCS described via a manifest file, and one hour of summary presentations from the breakout sessions. The first day was devoted to container specifications for inputs/outputs and execution security. The second day covered topics related to graphical user interfaces (GUI) needed for end users to enter algorithmic parameters. The third day was focused on container specifications for basic hardware required to run container-packaged algorithms successfully.

Conclusions and recommendations: Each breakout session collected inputs from the participants. Subsets of consistent inputs are included in this report, as well as challenges and recommendations for overcoming them in the near term. The list of challenges includes (1) a lack of ontologies for input/output (I/O) types, graphical user interface (GUI) types, and hardware types, (2) heterogeneity of data sources and I/O libraries in containers, (3) variable security requirements, (4) high complexity of GUI, (5) significant dependency of container execution on Graphics Processing Units (GPUs) and other hardware accelerators, and (6) dependency of maximal resource use on programmatic knowledge of compatibility between container and specific system hardware/software system.

Keywords

Software containers; interoperability; computational workflows; metadata specifications.

Table of Contents

List of Tables

List of Figures

Foreword

With the increasing size of collected data, distributed computational environments provide an accelerated productivity option for completing data analyses over extensive data collections and for federated learning over many data collections. The challenges of data analyses lie in the fact that heterogeneous analysis tools are written in multiple programming languages and have many dependencies on other software libraries. Containerization of tools offers a valuable solution for software execution in distributed computational environments with heterogeneous hardware and software configurations at each computational node. Containerized software tools must be interoperable as they are chained into workflows to facilitate tool reuse and the creation of increasingly complex computational analyses (workflow*s*).

Preface

The workshop aimed to establish a community consensus on creating interoperable containerized computational tools for extensive image collections that can be chained into scientific workflows/pipelines, regardless of the software infrastructure components, and cloud or on-premises hardware resources.

Acknowledgments

We want to acknowledge the sponsorship of the Information Systems Group in the Software and Systems Division and the Information Technology Laboratory at NIST for the workshop. We would also like to thank the NIST Legal department for their help with preparing the Federal Register posting and the NIST conference program staff (Crissy Robinson, Terri Viezbicke, Kevin Hill, Joseph Nastus, Gregory Eaton, and Aliza Reisberg) for supporting the organization and execution of the workshop.

Finally, we would like to acknowledge contributions from all active participants and all speakers (Mylene Simon, Michael Bartock, Nick Schaub, Sunny Yu, Timothy Blattner, Michael Majurski, Derek Juba, Ben Long) and breakout moderators (Mylene Simon, Pushkar Sathe, Joe Chalfoun, Philippe Dessauw, Timothee Kheyrkhah, Timothy Blattner, Michael Majurski, Derek Juba, Guillaume Sousa from NIST and Sameeul Samee, Kevin Duerr, Nick Schaub, Sunny Yu, Andrew Toler, Mohamed Ouladi, Antoine Gerardin from NCATS NIH).

Executive Summary

This NIST report summarizes the 2nd International Workshop on FAIR–CCS on December 5-7, 2023. The workshop focused on establishing a community consensus for descriptors of computational containerized software to enable chaining multiple containers into data processing workflows, as illustrated in Figure 1.

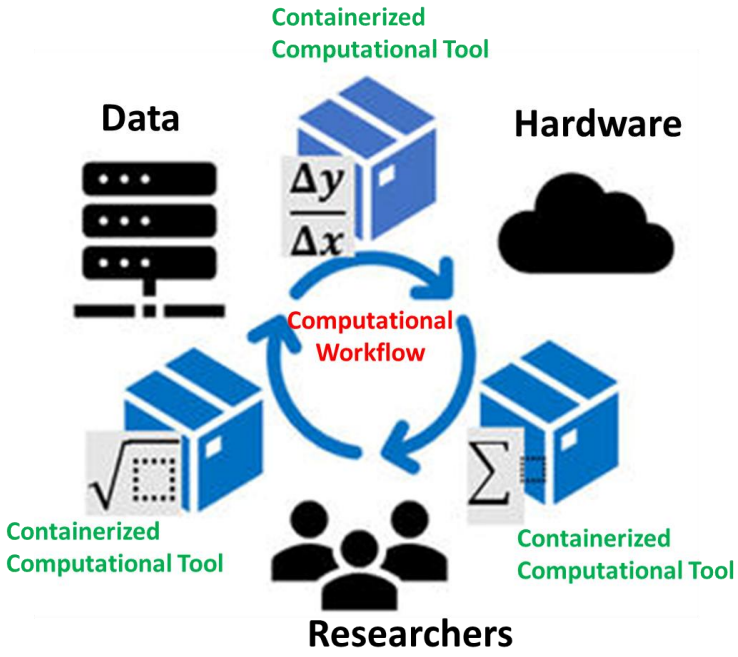

Figure 1: An overview of container-based computational workflows including data, hardware, and end users (researchers).

A general workflow scenario is shown in Figure 2. It shows three containerized computational tools that form a computational workflow applied to a dataset and run on unspecified hardware. Researchers would build such containerized computational tools, chain them together into workflows, and configure any input parameters for executing a container-based workflow.

The workshop aimed to discuss the container descriptors stored in a container manifest file. Figure 2 shows the role of container manifests in the software eco-system support for *Findable, Accessible, Interoperable,* and *Reusable (FAIR)* containerized computational software (CCS). The eco-system includes manifest files describing each container, a repository with manifest files and workflows, and workflow engines that orchestrate the execution of container-based workflows based on the available hardware and the manifest information.

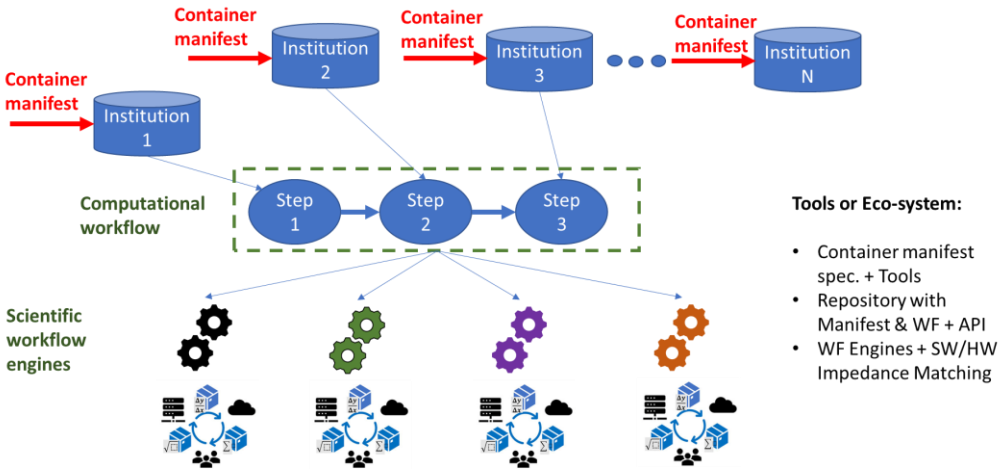

Figure 2: The role of container manifests in the software eco-system supporting FAIR containerized computational software.

Table 1 summarizes the main challenges and recommendations of the workshop based on all inputs from workshop participants.

Table 1: Summary of main workshop conclusions

Challenges	Recommendations
Lack of ontologies for input/output (I/O) types, graphical user interface (GUI) types, and hardware types	Develop a controlled vocabulary for each I/O type, GUI element type, and hardware type. Development should be coordinated while leveraging existing specifications, such as Slurm, Kubernetes, Common Workflow Language (CWL), etc., for hardware requirements. Investments are needed to build, maintain, and curate these controlled vocabularies.
Heterogeneity of data sources and I/O libraries in containers	In the manifest file, define the path to a file with file format suffix and develop/utilize robust data conversions to a consistent format. Investments are needed to implement/adopt conversion tools in many fields.

Challenges	Recommendations
Security requirements vary a lot.	The workshop participants needed more experience identifying general metadata descriptors for security requirements. They rely on security provided by computational platform providers and container technology providers. In the manifest file, include a unique ID, such as a container digest or a custom hash. Provide a publicly available, signed hash of the container contents for validation. Such a capability would require investments.
High complexity of GUI (complex logic with a variety of GUI primitives)	Support limited GUI descriptors/primitives directly encoded in the manifest. Provide validation tools for these. Any customizations and enhancements outside these primitives are not community-supported but can be defined and ignored in the manifest file.
Significant dependency of container execution on Graphics Processing Units (GPUs) and other hardware accelerators.	Include hardware information in the manifest file whose type matches hardware discovery information and is helpful to job schedulers. Investments are needed to develop such capabilities.
Programmatic knowledge of compatibility between the container and specific hardware/software system for maximal use of resources challenging.	Develop a software environment management system that automatically handles software/hardware compatibility issues and application programming interface (API) for inspecting containers. Investments are needed to build, maintain, and operate such capabilities, for example, for testing container executions.

1. Introduction

NIST hosted the 2nd International Workshop on *Findable, Accessible, Interoperable,* and *Reusable (FAIR)* Containerized Computational Software (FAIR-CCS) on December 5-7, 2023. The workshop was co-organized with NCATS NIH. The workshop aimed to evaluate and improve the specification for achieving the interoperability of containerized computational software. Adherence to a specification for FAIR–CCS) enables more straightforward and more productive reuse of containerized tools in complex data analyses by chaining tools into computational workflows.

NIST requested information from the community on approaches to achieving the interoperability of containerized software via the Request for Information (RFI). Responses to this RFI and the workshop discussion form inputs into a possible revision of the current approach to achieving FAIR–CCS via a manifest file, definitions of the entries in the current manifest file specification of FAIR–CCS, and the needed tools for automating adherence to the FAIR–CCS manifest specification.

The workshop was organized into breakout sessions dedicated to the main workshop themes, including Container Inputs and Outputs, Security of Container Execution, Graphical User Interface, and Execution Hardware Requirements. Each workshop day consists of one hour of general introductory presentations, two hours of breakout sessions discussing relevant topics, and one hour of summary presentations from the breakout sessions.

As the main goal for the virtual workshop was to establish a community consensus on creating interoperable containerized computational tools that can be chained into scientific workflows/pipelines, the following sections of this report summarize discussions in the breakout sessions. For each day and break out session covered below, we start by listing its high-level goals of the session and the overarching summary of feedback given. We follow that by short contextual explanations of breakouts pointing to the appendix containing bulleted lists of information or facts derived from each session. Often bulleted lists are lists of tools highlighted by participants that are relevant examples pursuant of the overall session theme or topic. These lists are meant to show the summary of background information that went into each overarching summary and are not meant to be exhaustive descriptions of all information covered or integrated into generating the high-level summaries of this report. This approach was thoughtfully done to conserve space as the target audience for this report is the public.

2. Theme: Container Inputs and Outputs

This theme aimed at specifying how containerized software inputs and outputs should be described in a manifest file so that a workflow (a chain of containerized software algorithms) can be formed by matching inputs and outputs. A manifest file is the metadata file that would be used to check whether an output of container 1 is compatible with an input of container 2 as a workflow consisting of container 1 → container 2 is formed.

2.1. Introductory presentations

The introductory presentations overviewed the workshop goals, the current draft of a manifest file managed by NIST to enable chaining containers into workflows, and the experiences at NCATS NIH with the current draft during the development of many image analytics workflows.

- Peter Bajcsy (NIST) presented the workshop goals.
- Mylene Simon (NIST) presented an overview of a draft manifest for containers as plugins to computational workflows.
- Nathan Hotaling (NIH) presented container-based plugin development at NCATS NIH.

2.2. Topics discussed in breakout sessions

All participants of workshop breakout sessions were asked to discuss answers to the following posed questions and provide comments and suggestions:

1. What is your experience with generic data organization types?
2. What is your experience with ontologies for data organization and file types (availability and usability)?
3. What is your experience with data storage mounted to a container?

2.3. Summary of breakout sessions

Based on the inputs from the scribes and moderators of all breakout sessions, the breakout summaries were partitioned into the following categories:

1. FAIR-CCS Format considerations
2. Data organization types
3. Use of ontologies
4. Data storage

These categories naturally present variabilities in file format, file locations, file content organization, semantic meaning of file content representation, and storage. The data organization types refer not only to file content organization on disk (e.g., images tiled into blocks, striped along rows, or forming pyramids) but also to file locations (e.g., in Amazon S3 bucket, on a networked server, and inside of a container). While the variabilities must be overcome to achieve shareable and reusable software containers, the workshop attendees also

discussed the pros and cons of each solution for a particulate variable, for example, the pros and cons of ontology implementations.

The summaries of notes from multiple breakout sessions are provided next after clustering them according to the categories. The actual notes are provided in Appendix A.

2.3.1. FAIR-CCS Format considerations

To streamline the utilization of the specification and enhance interoperability across diverse systems, our overarching goals for this topic focused on what was needed to establish robust data/algorithm specifications and what supporting information/tools/formats were needed. We prioritized getting information about the format of the specification, what was used now, how the configuration of the specification was currently and would ideally be implemented, and the validation process for the specification.

The breakout discussions emphasized the importance of adhering to specifications over the choice of format in data handling. The discussion suggested that while various common formats like YAML, JSON, and XML can be used, the focus should be on the supporting tooling rather than the format itself. Tools that can automate processes such as validation, configuration, and population are deemed crucial. Additionally, validating flat files can be time-consuming, highlighting the importance of ensuring the accuracy of variables before submission. Moreover, the discussion underscored the necessity of supporting dated or historical formats, acknowledging the ongoing value and utilization of legacy tools.

2.3.2. Data organization types:

Data that are being either loaded or created by the tools described in the specification can be saved in various file structures. These structures can be semantically meaningful (i.e., a file called “Tiff” for all .tiff files) or not; tools can expect an input folder directory location or a list of files or to be serially fed images one at a time from a data serving service, etc. The goals for these discussions were to determine which of the above was prevalent, which was preferred, and what was needed to enable tools built by disparate labs to load data in a typical organizational structure.

The breakout sessions outlined various methods and technologies used for sharing and managing files across different storage solutions, including Network Attached Storage (NAS), High-Performance Computing (HPC) parallel file systems, and Cloud Storage, specifically S3 buckets. The discussions described how containers interact with files through various means such as disk storage, Hypertext Transfer Protocol Secure (HTTPS) services, persistent volumes, file shares on Kubernetes clusters, and direct drive mounting. Additionally, The discussions included the process of specifying container manifest inputs, which involves listing paths to input files that the pipeline then locates and retrieves, whether they are stored locally or in the cloud.

The breakout sessions also covered the types of I/O code libraries and tools suitable for handling images in formats like Tiff, Zarr, OME-NGFF, and NIfTI, alongside common

photography image file formats. Furthermore, the discussions touched on the use of metadata for organizing data, emphasizing the shift away from traditional file system organization towards database queries, subsetting, and filtering for identifying and fetching files. This approach suggested a move towards more dynamic and flexible methods of data management that rely less on the physical organization of files on storage systems.

2.3.3. Use of ontologies:

Ontologies were found to be needed in several aspects of the specification, including at the file format level, the hardware level, and the description of UI types at each level. Each level needed an ontology, taxonomy, or controlled vocabulary for various reasons. File formats need an ontology, taxonomy, or controlled vocabulary to guarantee that a file saved by one step in a workflow can be successfully loaded, processed, and saved by the next step. Without having unique identifiers that specify the exact data format (and its associated metadata), these guarantees cannot be made, and, therefore, a specification for interoperable tools cannot exist.

The breakout sessions also discussed software tools that need to be reusable; an ontology, taxonomy, or controlled vocabulary of the hardware executing a given software tool is required. Here, due to the complex nature of modern hardware accelerators and their rapidly evolving ecosystem (Graphical Processing Units (GPUs), Tensor Processing Units (TPUs), etc.), a unique identifier to specify the exact hardware and device drivers interfacing the hardware for a given software tool is needed to ensure that a given software tool can be reused appropriately and without errors.

Finally, for a software tool to be accessible to users who are not programmers, graphical user interfaces (GUIs) are needed. However, an ontology, taxonomy, or controlled vocabulary is necessary for these UI elements for one system to be able to appropriately load and render a GUI without explicitly being designed to render that particular UI. This is further complicated because the software tool must run in a priori unknown runtime environment (i.e., web browsers, Python applications, Java applications, etc.). Therefore, a detailed ontology of UI elements and their corresponding representation in various languages is needed to make these tools user-friendly to users without programming skills.

2.3.4. Data storage:

Data storage is integral to a FAIR tool specification because, depending on how data are stored in each system, it dramatically changes how efficiently data are loaded from the software tool. For example, parallel reading and writing from a file are possible in parallel file systems available on high-performance computing (HPC) clusters. In contrast, this is only possible in the bucket type of cloud data storage if very modern file formats are used. Understanding how the community deals with these different formats of file loading (and, by extension, the underlying file storage) is critical to understanding what a FAIR tool must do to be interoperable in all data storage environments.

The breakout sessions discussed the evolving landscape of data management and storage, as well as a diverse array of solutions to address the challenges of heterogeneity in storage and

access control. The adoption of object stores, such as S3 buckets and traditional file systems behind firewalls, provide versatile options for data storage. Moreover, Data Version Control (DVC) and Role-Based Access Control systems are instrumental in managing data changes and ensuring secure access. The conversion of heterogeneous input files into a unified file format has become crucial as well. Several long-term storage options, such as Glacier, and partial downloading solutions, like the International Business Machines Corporation (IBM) Storage Fusion, have been mentioned to manage storage costs effectively.

The integration with cloud storage platforms was discussed as well. Solutions such as Amazon Web Services (AWS) Mountpoint facilitate the use of S3 as a file system, with innovative applications like Artificial Intelligence (AI) for Life developing servers that interface directly with S3. These servers not only mount S3 but also enforce user-specific policies and generate pre-signed Uniform Resource Locators (URLs) for secure file transfer. Emerging repositories and cloud solutions, such as Rembi (supported by Riken in Japan) and the European Molecular Biology Laboratory (EMBL) Embassy cloud (which utilizes an OpenStack-based architecture with Kubernetes cluster deployment and Ceph Storage backend) are showcasing the potential of specialized storage solutions.

3. Theme: Security of Container Execution

This theme aimed at specifying how the security of containerized software should be described in a manifest file so that each of the containerized software algorithms in a workflow is executed according to its security constraints.

3.1. Introductory presentation

The theme focused on the security of container execution, which was introduced by a NIST member of the cybersecurity division.

- Michael Bartock (NIST) presented Hardware-Enabled Security: Policy-Based Governance in Trusted Container Platforms

Note: The slides from all speakers are available from the workshop web page.

3.2. Topics discussed in breakout sessions

All participants of workshop breakout sessions were asked to discuss answers to the following posed questions and provide comments and suggestions:

1. What is your experience with the benefits of including security-related metadata?
2. What is your experience with security-related metadata to verify the integrity of container content?
3. What is your experience with encrypting container content?
4. What is your experience protecting a code execution within a container with a passphrase or a license key?

3.3. Summary of breakout sessions:

Based on the inputs from the scribes and moderators of all breakout sessions, the breakout summaries were partitioned into the following categories:

1. Security in practice
2. Lessons Learned from Others

These two categories span a narrow range of possible security topics since the audience was not very concerned about security. Most of the workshop attendees were scientists concerned with reproducibility and reusability of software. However, this topic was recognized as a necessary one.

The summaries of notes from multiple breakout sessions are provided next after clustering them according to the categories. The actual notes are provided in Appendix B.

3.3.1. Security in practice

Executing arbitrary code (adhering to the FAIR-CCS specification) provided by diverse developers who are not part of the organization implementing the code is a daunting security

task. Concerns range from bad actors hiding malicious code to take control over computational resources owned by others to hidden bitcoin/crypto miners trying to utilize HPCs for personal gain and to unintentionally poorly written code utilizing all available resources and crashing a computer cluster. The line between what the responsibility of the software tool maker is, what the responsibility of the system administrator running those tools is, and what responsibilities are taken on by the platform developers where the software tool is registered needs to be better defined. Due to this, and the fact that the majority of the community who came to the sessions were software tool makers, not system administrators for supercomputing centers or registry administrators for software tools, much of the discussion was centered around trying to clarify what could/should be taken on by tool makers. The consensus was as little as possible.

The breakout sessions discussed managing the security and licensing of containers in air-gapped or regulated environments. The discussion suggested using region-specific licenses and employing hashes and signatures for easier validation. Protecting credentials/licenses is critical, especially when tracking lot provenance. The recommendations were about pulling images from authentic sources, maintaining a separate registry for security metadata, and avoiding connections or data transmissions to the internet. Finally, the workshop participants advocated for role/permission-based access for securing containerized environments.

3.3.2. Lessons Learned from Others

While direct experience from users on security was low, there were some suggestions for software tool registries that we could look at how security was handled on a larger scale than the scale of a local institutional registry. Additionally, a few accepted ideas seemed manageable for software tool makers to take on, such as a container digest and custom hash.

Participants discussed Amazon Elastic Container Registry and Bio Containers as registry examples. They also agreed on the need for a container digest to verify image integrity and identify its version. Finally, participants also emphasized the need for a custom hash algorithm.

4. Theme: Graphical User Interface

This theme aims to specify how to describe graphical user interfaces (GUI) for entering parameters required by the containerized software. In scientific computational workflows, scientists with backgrounds other than information technology prefer to use GUI instead of command line interfaces (CLI) to enter parameters. The description of GUI types and fields is needed to facilitate GUI design and on-the-fly GUI creation in workflow editors as computational workflows are configured for execution. The discussions focused on definitions and representations of GUI components in a manifest file associated with each containerized software algorithm.

4.1. Introductory presentations

The introductory presentations overviewed the previous workshop day, the current GUI fields used in a manifest file managed by NIST to enable chaining containers into workflows, the needs for GUI at NCATS NIH, and the NIST investments into managing a registry of manifest files.

- Peter Bajcsy (NIST) presented a quick summary of the previous workshop day and the goals for the user interface-related discussions.
- Michael Majurski (NIST) illustrated user interface specifications in the draft manifest and the need for user interfaces in microscopy stitching tools.
- Sunny Yu (NIH) described the need for user interfaces at NCATS NIH.
- Ben Long (NIST) presented a registry for manifest files and the considerations for finding, accessing, and retrieving the manifest files.

4.2. Topics discussed in breakout sessions

All participants of workshop breakout sessions were asked to discuss answers to the following posed questions and provide comments and suggestions:

1. What is your experience with file type or service for containerized computational workflows?
2. What are the basic types of algorithmic parameters?
3. What is your experience with the languages and libraries used for on-the-fly (on-demand, dynamic) GUI creation?
4. What is your experience with tools to automate the generation of manifest sections describing conditional GUIs for encoding complex logic into static web forms for conditional user interfaces?
5. What command line interfaces (CLI) are of interest in creating GUIs?
6. What web application programming interfaces (web API) are of interest in creating GUIs?

4.3. Summary of breakout sessions

Based on the inputs from the scribes and moderators of all breakout sessions, the breakout summaries were partitioned into the following categories:

1. General considerations
2. Lessons Learned from Others
3. Use Cases

The first category, labeled “General considerations,” covered discussions about whether GUI should be included since many workshop participants work in high-performance computing (HPC) environments that prefer a command line interface. For scientific workflows, the GUI is essential, and general requirements and GUI representations were discussed.

The second category, “Lessons Learned from Others“, included discussions about specific libraries, existing scientific workflows and their GUI, possibilities of converting library-based code to Web GUI, and the on-the-fly execution of GUI creation based on the GUI entries in a manifest file.

The third category, “Use Cases,” covered examples of GUI needed for status updates.

The summaries of notes from multiple breakout sessions are provided next after clustering them according to the categories. The actual notes are provided in Appendix C.

4.3.1. General considerations:

Overall, GUI elements in a FAIR-CCS specification were seen as needed to increase the accessibility of such tools (the A in FAIR). However, the community was relatively split on the necessity of such elements. Depending on the use case, for example, discovery and experimentation, comprehensive parameter scans in experimentation, production processing runs, or headless submission of workflows to an HPC, the necessity of a GUI (or not) was debatable. Therefore, we wanted to hear from the community what their thoughts were on (a) whether GUI elements should be mandatory, (b) if so, how complex these elements should be, and (c) what type of ontology/taxonomy/controlled vocabulary was needed for these GUIs to be interoperable in a variety of computational environments and programming languages.

The breakout sessions discussed an integration of GUI specs into computational tool manifests, and the opinions varied. Some argued that GUI considerations should be separate from the specification, while others advocated for including GUI elements. A consensus among HPC users favored optional GUIs, while computer cloud and scientific users found GUIs invaluable. Basic GUI primitives like dropdowns and radio buttons could be universally adoptable, while complex GUIs could reside in optional sections of the manifest. Raw files were proposed for passing non-trivial parameters for sophisticated GUI requirements. Recommendations for GUI integration included the adoption of open frameworks and APIs, offline functionality for HPC resources, and tooling support for translating workflows into GUI instances.

4.3.2. Lessons Learned from Others:

Various tools and platforms have developed simple GUI specifications for their specific, non-interoperable computational workflow platforms. Specifications for these GUIs abound and were used by one workshop attendee or another. Each workshop attendee had its pros and cons, and none was found to be a standard that was ready out of the box. However, various tools were discussed and named from which the FAIR-CCS specification could take inspiration.

The breakout sessions discussed tools and methodologies related to software development, with an emphasis on building and deploying interactive graphical user interfaces, web services, and ML pipelines. The discussion mentioned several languages, libraries, and frameworks, including tools like Swagger Docs, FastAPI, and Node.js for microservices and web development and frameworks like KNIME, Flyte, and Argo for workflow and pipeline management. Participants also discussed issues like security risks and converting library-based code to web interfaces. Overall, the discussion offered a comprehensive overview of tools and technologies for developers, catering to a wide range of professionals from biologists to web developers.

4.3.3. Use Cases:

Due to the large number of current solutions listed above, use cases were also mainly specific to each of the above platforms, i.e., each solution was designed to handle a particular set of use cases. In addition to the use of case-driven design of solutions, the browser and, by extension, HTML, CSS, PHP, JavaScript/Typescript, and WebAssembly (WASM) could be considered the largest and most successful implementation of GUI standards defined by the World Wide Web Consortium. In this vein, a minimal additional discussion was devoted to the use case of creating a progress report GUI for large-scale simulations. The web page would be parsing outputs, and with a template, it would inform end users about the progress (like the elastic search platform).

5. Theme: Execution Hardware Requirements

This theme aimed at specifying what hardware dependencies are required for executing each containerized software algorithm and how to describe the hardware dependencies in a manifest file. The focus was on compatibility between hardware and software (failed vs. successful execution), optimal execution (max speed of execution), and testing execution (benchmark execution).

5.1. Introductory presentations

The introductory presentations overviewed the workshop goals, the current draft of a manifest file managed by NIST to enable chaining containers into workflows, and the experiences at NCATS NIH with the current draft during the development of many image analytics workflows.

- Peter Bajcsy (NIST) summarized the previous workshop day and the goals for the hardware-related discussions.
- Tim Blattner (NIST) overviewed hardware specifications in a draft manifest.
- Nick Schaub (NIH) presented hardware heterogeneity at NCATS NIH that poses compatibility challenges for container executions.
- Derek Juba (NIST) reviewed hardware heterogeneity at NIST, which requires matching hardware specifications with built software containers.

5.2. Topics discussed in breakout sessions

All participants of workshop breakout sessions were asked to discuss answers to the following posed questions and provide comments and suggestions:

1. What is your experience with the benefits of including hardware requirements?
2. What is your experience with ontologies for computer hardware (existence, coverage, access)?
3. What metadata fields describing hardware can be used for job schedulers to request hardware on computer nodes in the cloud or HPC resources (e.g., Simple Linux Utility for Resource Management (SLURM), Kubernetes, CWL command line tool)?
4. What is your experience with metadata fields describing hardware that can be automatically detected using hardware discovery tools in the cloud or on HPC resources by job schedulers (e.g., SLURM, Kubernetes, CWL command line tool)?
5. What metadata fields describing hardware should be included about the hardware on which a container was tested/validated?
6. What metadata fields describing hardware should be included about all hardware types on which a container has been benchmarked?
7. How would you like this specification to be governed? What structure for changes/updates/feature extensions?

5.3. Summary of breakout sessions

Based on the inputs from the scribes and moderators of all breakout sessions, the breakout summaries were partitioned into the following categories:

1. Purpose of including hardware specifications
2. Purpose of ontologies for hardware specifications
3. Suggested hardware information to capture.
4. Hardware discovery
5. Hardware requirements for tests and validations
6. Metadata about hardware requirements
7. Governance of manifest specifications

These categories map directly to the seven topics prepared for the workshop breakout sessions and formulated as questions to be discussed. The discussions confirmed the benefits of including hardware specifications since, although the software container technology targeted the independence of software execution from its hardware platforms, the evolution of hardware environments has caused incompatibilities between hardware and software. The hardware specifications must be included with containers for the reusability of computational workflows formed by containerized software in the long term.

Another set of discussion topics revolved around specific hardware characteristics to capture, their corresponding metadata fields in a manifest file, and their semantic mappings. These discussions included utilizing hardware specifications for hardware discovery, as well as container testing and validation.

The final category was about the governance of such hardware specifications. As the workshop attendees came from many different application areas, the challenge of governing hardware specifications became a separate discussion topic to understand methods for keeping the hardware specifications up to date with the needs of many users.

The summaries of notes from multiple breakout sessions are provided next after clustering them according to the categories. The actual notes are provided in Appendix D.

5.3.1. Purpose of including hardware specifications:

Generally, before diving into the specifics of requiring hardware specifications in a FAIR-CCS specification, the question of whether the hardware specifications belonged to the container manifest specification was needed. An extensive debate was around the utility of actually requiring any hardware specifications at all due to (a) hardware heterogeneity across users and use cases, (b) the unregulated use of vocabularies and ontologies to describe hardware, and (c) the relatively few resources available to researchers to test diverse hardware systematically and cheaply. The underlying question was asked: “If we can’t do it well or systematically, should we do it at all? “.

The breakout sessions discussed that capturing and detailing hardware requirements are crucial in cloud and HPC computing to ensure compatibility, optimize performance, and enhance reproducibility. Recording specific hardware configurations on which a container was validated to run and using a standardized hardware ontology or controlled vocabulary can facilitate efficient resource management and deployment. Learning from existing solutions, such as Kubernetes with Argo workflow, Hewlett Packard Enterprise's compute capability metadata, and Amazon's ECS and spot fleet models, can provide valuable insights into cost-effective and dynamic resource allocation.

5.3.2. Purpose of ontologies for hardware specifications:

A significant takeaway from the above discussion was that a substantial limitation for researchers is that there is no standard hardware ontology/taxonomy/controlled vocabulary they can quickly draw from to describe what hardware their software tool was tested on or compatible with. If an ontology, taxonomy, or controlled vocabulary existed, then mapping compatibility between concepts/terms within it could be undertaken to develop reasonable assurances of portability across defined concepts (and the levels to which that compatibility spanned). However, without first having that ontology, taxonomy, or controlled vocabulary, creating the compatibility inheritance tree was impossible or extremely resource intensive.

The breakout sessions discussed challenges in translating metadata across infrastructures and highlighted the need for controlled vocabulary and ontologies for complex descriptions. The discussion provided insights into current practices and solutions, such as using Kubernetes and SLURM for hardware specifications and adopting hardware tiers and Visual Effects (VFX) reference platforms for benchmarking. The discussion also mentioned the importance of standardized frameworks, such as the FAIR Principles and mOSAIC ontology, for ensuring clarity and interoperability in hardware specifications.

5.3.3. Suggested hardware information to capture:

Even if hardware ontologies, taxonomies, or controlled vocabularies existed, how much information should be included in the specification? How detailed was a description of the hardware needed to ensure interoperability? How could this information be obtained and confirmed in a standardized way that could be relied upon by a large community of users who did not know or talk to each other? These questions were critical to understanding and defining the hardware specifications.

The breakout sessions discussed different perspectives on computing resource specifications, with HPC enthusiasts wanting more detailed hardware specifications for optimal performance. At the same time, general users prefer software that works out of the box. Participants agreed on the necessity of simple descriptors like CPU, GPU, and TPU, with more advanced specifications as optional. Orchestration solutions can use hardware information to impact cost management, workload decisions, and performance estimations. For instance, Argo workflow and TOIL can specify resource requirements, showing the practical implications of accurate definitions of computing needs in various environments.

5.3.4. Hardware discovery:

If (1) hardware definitions were included in the specification, (2) an ontology, taxonomy, or controlled vocabulary existed to describe what a software tool was tested on, and (3) the minimum reporting requirements above were met, a further challenge existed in that the computational hardware resource running a software tool would rarely be using the same hardware that a given software tool was tested with. Given this hardware portability challenge, how platform administrators can utilize the reported hardware information provided with software tools in their computer clusters was a question for our audience.

Hardware "impedance" matching, or the translatability of the knowledge of what an algorithm running in one hardware environment meant for running the same algorithm in a different hardware environment, is an unsolved problem. The debate about this unsolved problem was another consideration when defining hardware specifications in a container manifest specification. Here, the question was more focused on the value proposition: "Even if a software tool maker defines everything perfectly, is there a value-added if that knowledge is not generalizable to other computational hardware resources? If generalizability is not provided, then why are those hardware definitions and specifications required at all?"

The breakout sessions discussed the challenges in creating platform-specific solutions that integrate software and hardware components. The discussion emphasized the need for tailored impedance matching and compatibility between software applications and specific hardware devices. The discussion also highlighted the importance of interacting with job schedulers and identifying available resources at runtime, which depend on the configuration of the workflow manager. These complexities require a deep understanding of both software and hardware components.

5.3.5. Hardware requirements about tests and validations:

This discussion stemmed naturally from the hardware Ontology and Information to Capture sections 5.3.2 and 5.3.3. Essentially, tests and validations can provide the information reported by the software tool creators and described in the Information to Capture section 5.3.3. An essential discussion in the workshop was the ontology, taxonomy, or controlled vocabulary needed to capture such information. This discussion covered not only compatibility (portability of software tool execution across hardware) but also "expectations" of execution performance and outputs from a given software tool. How fast and resource-intensive is a given software tool in each computational hardware and software environment? What would be the expected costs given a set of inputs in cloud computing environments? Here, standards for hardware environments and ontologies/taxonomy/controlled vocabularies are necessary, as well as datasets to process so that apples-to-apples comparisons could be made between tools.

The breakout sessions discussed strategies for optimal resource allocation in computational environments. The discussion emphasized planning and adjusting resources based on past usage and execution profiles. Key points included testing and validation for hardware requirements, container specificity to architectures, and the level of detail to expose for developers and sysadmins. The discussion also highlighted the need for openly available

computing platforms to test tools for compatibility. Participants concluded with remarks about lessons from existing practices and the community's role in ensuring software performance across different hardware setups.

5.3.6. Metadata about hardware requirements for benchmarking:

In line with section 5.3.3, if data and hardware environments are set, then what information about the software tool, the hardware, and the data is necessary to make the results truly reusable? This question was discussed in the context of reusability across the community in a way that empowered users to “know” something about software tools that they had never used before.

The breakout sessions discussed insights and recommendations for benchmarking code and predicting its performance on various systems. The discussion suggested categorizing hardware specs into three detail levels and using metadata to standardize benchmarks. Participants also highlighted Nextflow as an example of community collaboration in the field of workflow languages.

5.3.7. Governance of manifest:

The final topic discussed was how all the above decisions and implementation paths could be made, ratified, and implemented for a potential standard (or at least a community consensus). How would community engagement occur, who would decide when to do one implementation path vs another, who had a voice, how often, and how would conflicts be mitigated? These questions were discussed at a very high level. Only a few conference participants had experience in dealing with this level of organization/governance.

The breakout sessions discussed challenges with ontologies and controlled vocabularies, highlighting the need for validation, governance protocols, and generic guidelines. Participants also noted their minimal experience in these areas.

6. Workshop Statistics

The workshop was attended by 111 attendees out of 160 registered participants over the period of three days. The workshop participants came from various affiliations, including Academia, the U.S. Federal Government, U.S. Government Research labs, Industry, Law Firms, non-U.S. government, non-profit research organizations, Research-Performing Organizations, and Individuals. Note that the categories were self-reported by workshop attendees. Research-Performing Organizations differ from non-profit research organizations in that they are based on a profit or non-profit business model. "Non-U.S. Government" refers to the State, Local, Territorial, Tribal, or national government of another country than the United States. A histogram of workshop participants is shown in Figure 3.

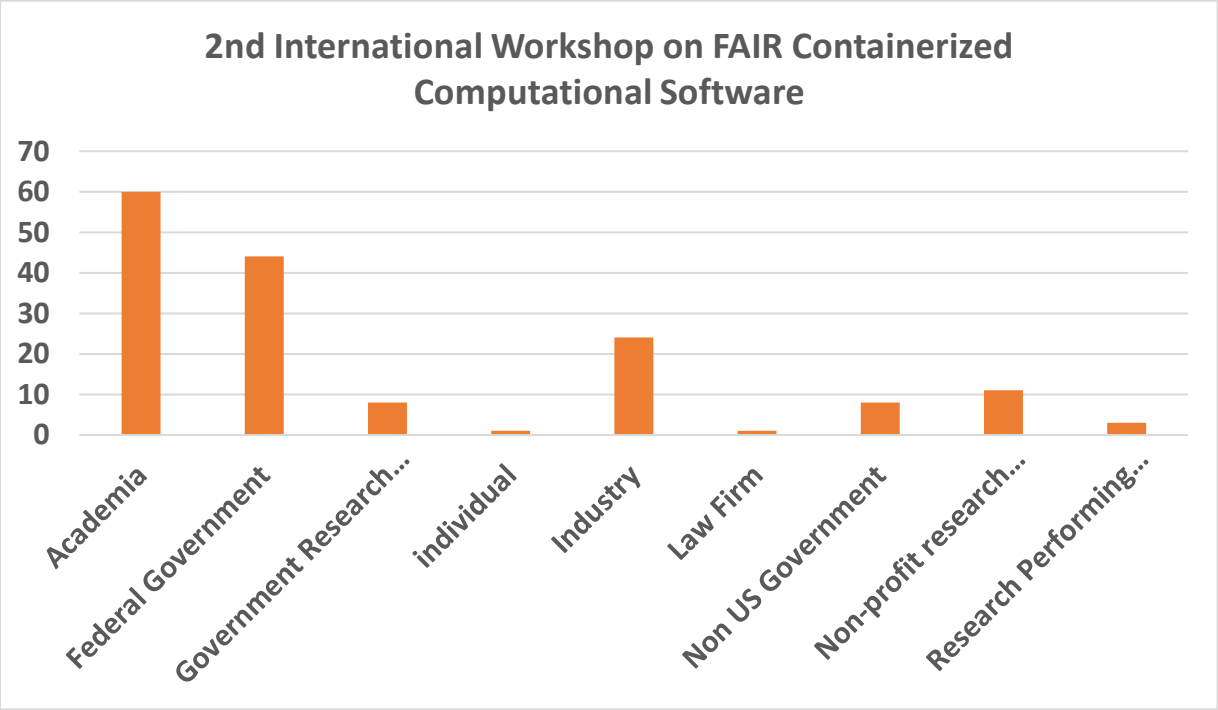

Figure 3: Distribution of registered participants

7. Additional Resources

The GitHub repository with the draft specification of container manifests is
https://github.com/usnistgov/fair-chain-compute-container.

Registries of manifests adhering to the draft specification of container manifests:

https://wipp-plugins.nist.gov/

https://wipp-registry.ci.ncats.io/

Workshop event URL:

https://www.nist.gov/news-events/events/2023/12/2nd-international-workshop-fair-containerized-computational-software

Previous workshop event URL:

https://www.nist.gov/news-events/events/2019/12/interoperability-web-computational-plugins-large-microscopy-image

The workshop report from the 1st workshop:

https://www.nist.gov/publications/interoperability-web-computational-plugins-large-microscopy-image-analyses

Appendix A. Notes from Breakout Session on Container Inputs and Outputs

FAIR-CCS Format considerations

- The specification is more important than the format itself. All common formats could/should be supported as long as they comply with the specification.
- The specific format (e.g., yet another markup language (YAML), JavaScript Object Notation (JSON), and Extensible Markup Language (XML)) is not as important as the tooling around it. Tools that perform automated validation, configuration, population, etc., are much more important than the format itself.
- Flat files can take longer to validate, so we want to confirm that variables are correct before submission.
- Dated/historical formats must be supported because legacy tools still have high value/utilization.

Data organization types

- **File Locations:** NAS, parallel file systems on HPC, or Cloud Storage - S3 buckets.
- **Container ←→ File:** Share files between nodes by storing them on disk, serving them by HTTP(S), by persistent volumes or file shares on the cluster through Kubernetes, or by direct drive mounting.
- **Container Manifest Inputs:** List of paths to input files - Pipeline finds the actual files and brings them (can be a local or cloud path).
- **Container I/O code:** Libraries/Tools for images stored in Tiff, Zarr, OME-NGFF, and Neuroimaging Informatics Technology Initiative (NIfTI) file format in addition to typical image file formats used in photography.
- **Metadata-defined data organization:** Database query, subset, and filtering used to identify and retrieve files. File organization on file systems, storage systems (e.g., S3, NAS, Parallel file systems), etc., are less relevant here.

Use of ontologies

- **Pros of ontologies**
    - Institutes have many different types of users with different roles, and ontology could reduce training time for users.
    - Ontologies could be critical for sharing our containerized components with the outside world in an unsupervised manner.
    - Ontologies allow for more straightforward and less ambiguous communication of concepts, leading to faster/easier adoption of new tools.
    - Discovery of relevant tools is made dramatically simpler/faster with the utilization of an ontology.

- **Cons of ontologies**
    - It is not apparent whether a complete ontology would be needed. It may be needed for big workflows, very large institutions with many user types, or mixing and matching algorithms. However, a complete ontology may be overkill for what is needed.
    - "AI for Life" tried using pre-defined tags. It had a problem that ontology did not support every tag needed. It is not clear if arbitrary tags should be allowed.
    - Ontologies are challenging to sustainably fund – many current community-supported ontologies have received funding and are not well updated/supported after the initial grant(s) ran out.
    - Ontology can make it more complicated for users (especially novice users) to annotate correctly and set up their workflows.
    - Ontologies require training and good tooling to be used appropriately by tool creators and, therefore, outside of the "average" researcher trying to solve their domain-specific problem.
    - Ontologies need a mechanism for versioning and version control.
- **Many pointers to ontologies, file catalogs, and frameworks:**
    - HPC Ontology: https://hpc-fair.github.io/ontology/
    - Nf-core: community-driven standard- https://nf-co.re/
    - S3 Quilt file browser: https://docs.quiltdata.com/catalog/filebrowser
    - Bioimage Analysis - visualize & analyze complex images: https://qupath.github.io/
    - List of ontologies used in IDR: https://idr.openmicroscopy.org/about/linked-resources.html
    - OxO - service for finding mappings (or cross-references) between terms from ontologies, vocabularies, and coding standards: https://www.ebi.ac.uk/spot/oxo/
    - Examples sites: https://www.ebi.ac.uk/bioimage-archive/rembi-help-overview/; https://www.ebi.ac.uk/empiar/deposition/manual/
    - Cryo-EM image processing framework: https://scipion.i2pc.es/
    - Pub2tools, RO-Crate: https://www.researchobject.org/ro-crate/
    - EDAM and EDAM bioimaging ontology: https://bioportal.bioontology.org/ontologies/EDAM-BIOIMAGING; https://edamontology.org/poster-bioimaging.pdf
    - Fast Healthcare Interoperability Resources (FHIR) Ontology/OWL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8367140/

- Brain Imaging Data Structure (BIDS) format: https://bids.neuroimaging.io/
- Huggingface for bioimage models: https://huggingface.co/models?other=biology
- DICOM file ontology: https://www.ebi.ac.uk/ols4/ontologies/dicom

Data Storage

- **Heterogeneity of storage and access control:**
  - premise-premises High-Performance Computing (HPC)
  - Object store like S3 buckets
  - File system behind the firewall
  - DVC - Data Version Control
  - Role-Based Access Control
- **Conversions and storage costs:**
  - Conversion of heterogeneous input files into a coherent file format while using cheap storage for the original files.
  - Many fields did not have a common, open, well-defined file format to convert to/from.
  - Glacier - slow long-term storage: https://aws.amazon.com/pm/s3-glacier
  - IBM Storage Fusion - partial downloading: https://www.ibm.com/products/storage-fusion
- **Interfaces with cloud storage:**
  - Protocols: AWS Mountpoint and s3fs for S3 as filesystem. AI for Life built a server that interfaces with S3 and Mounts S3 with the open-source server. Creates a policy for each user's home folder. Generates pre-signed URLs to send and receive files in AWS S3.
  - Nextflow handles mounting cloud storage as file systems into the container.
  - Rembi (images repository supported by Riken in Japan; https://ssbd.riken.jp/database/)
  - EMBL Embassy cloud – Open stack-based (Kubernetes cluster deployment and Ceph Storage backend - https://docs.embassy.ebi.ac.uk/userguide/Embassyv4.html
  - Pegasus - Data available on the host OS via NAS or a local filesystem; https://pegasus.isi.edu/documentation/user-guide/containers.html#containers-symlinking; Data from S3 buckets are downloaded to the compute nodes during the data staging phase.

Appendix B. Notes from Breakout Session on Security of Container Execution

Security in Practice

- Most participants reported no hands-on experience.
- Licenses – impose regional, country, and file-based licenses. Security is combined with licensing.
- Use hashes and signatures against public-private keys to enable facile validation of downloaded containers in air-gapped infrastructure.
- Protect credentials/licenses, especially when doing lot provenance tracking and logging.
- Security is very relevant when institutions host their container registries.
    - Pull images from authentic sources.
    - Contain a separate registry to check security metadata.
    - Don’t want a container to connect or send information back to the Internet.
- Encryption of input data is often more critical than container encryption.
- Instead of encrypting containers, use control access to location - hardware/systems.
- Recommended role/permission-based access/execution.

Lessons Learned from Others

- Amazon Elastic Container Registry: https://aws.amazon.com/ecr/
- Bio Containers: https://github.com/BioContainers
- Container digest: Multiple participants agreed on the usefulness of having a container digest to pinpoint the container image's exact version and check its integrity.
- Custom hash: They also agreed that an option for using a custom digest/hash algorithm should be present to avoid relying only on Docker tools.

Appendix C. Notes from Breakout Session on Graphical User Interface

General considerations:

- **Presence of GUI specs in a manifest?**
    - *Variety of opinions*: GUI considerations should not appear in the specification versus GUI for parameters, outputs, and intermediates.
        - Human in-the-loop is the boundary between workflows.
        - Making GUI optional was preferred by many HPC users, but many cloud/scientific users found the GUI to be invaluable.
        - Most participants agreed that complex GUIs were out of scope for the manifest but could be defined in optional manifest sections or as a field for referencing external raw files. However, these cannot prevent the tool from completing its function(s). In other words., all tasks/ parameters/settings should be programmatically accessible via the command line.
        - Simple/basic GUI primitives – drop downs, radio buttons, etc., could be required for all plugins. Ideally, they can also be ignored if desired by the user.
    - *GUI Complexity via raw files:* Non-trivial parameters passed via raw files since we cannot process every possible input. However, these "advanced" GUIs should not stop a containerized algorithm from running only from the command line/headless if needed/desired.
- **Requirements:**
    - *Open frameworks and APIs* to help with adoption.
    - *Needs to work offline* for high-performance computing (HPC) resources.
    - *Creation of GUI by using libraries and services:*
        - Ability to submit through a queue or a service.
        - TensorFlow's visualization toolkit: https://www.tensorflow.org/tensorboard
        - The AI developer platform: Weights & Biases - https://wandb.ai/site
        - There is a tool that will take GraphQL schema and create a frontend for it.
        - Airtable Connected Apps Platform can create apps for non-technical users.
    - *Forward-looking support: Large Language Models (LLMs) can be used to generate web API* or as user interfaces themselves.
    - *Tooling support:*

        - Translate the directed acyclic graph (DAG) to JSON format (or other formats) and start from the JSON to generate the GUI instances.
        - GUI tools need to be able to handle conditional logic.
        - Manifest should not be too specific and tied to a particular GUI.

- **GUI representation considerations:**
    - *Use a markup language* to define the supported basic types and show a preview.
    - *Use GUI Ontologies:* The GUI should use ontologies to describe each step's content better.
    - A simple Graphic Tool that allows the user to run a workflow online.
    - Need for validation following a standard ontology (RO-Crate, EDAM, Bioschemas, GA4GH WES)

Lessons Learned from Others

- **Languages/Libraries:**
    - Galaxy workflow GUI types (Galaksio's GUI for biologists)
    - Swagger Docs, FastAPI microservices in Python, JavaScript server in Node.js, TypeScript/Django, PyQT, JSX, Vega-Light; ReactJS
        - Swagger does not support conditional parameter input logic yet.
    - OpenAPI: https://github.com/OAI/OpenAPI-Specification/issues/256
    - Label studio - annotation library, XML → HTML; https://labelstud.io/playground/
    - Flask with Jinja2 (Web Server Gateway Interface with templates)
    - Pixme from Google Apps (Android)
    - streamlit-jupyter – create interactive GUI inside Jupyter Notebook (alternative to voila).
    - *Warning:* Users can inject JS in the manifests and introduce a security issue.
- **Workflows with GUI**
    - KNIME (server version, local version), build custom GUI from primitives.
    - Flyte, Build & deploy data & ML pipelines, https://flyte.org/
    - Argo, Kubernetes native workflow engine, https://argoproj.github.io/workflows/
    - Polus: Workflow builder https://github.com/PolusAI/workflow-builder
    - Napari allows the user to create modules that can describe the GUI.
    - In Matlab, you can create pop-ups using CLIs if you define a GUI.
    - Textualize: they have tools https://github.com/Textualize

  - TxTk, WxWidgets: Allow for testing before launching workflows in the production environment.
  - Python: argparse, Airflow, Dagster, Prefect, Kubeflow pipelines, Parsl, SnakeMake - https://snakemake.readthedocs.io/en/stable/
  - Galaxy, https://galaxyproject.org/learn/advanced-workflow/
  - Nextflow - If a task fails in CLI, inspect the log to triage the issue.
- **From library-based code to Web:**
  - Galaxy interprets the XML.
  - From JSON to UI. (JS, Angular, Node.js) -> It uses an Angular NGX Schema (NGX = Angular + x (redefined/modern/new/next-gen) update) for a web form.
  - R to DashApp to HTML.
  - R to Shiny App
  - Wanted: Python Notebooks to Typescript app.
- **On-the-fly execution:**
  - Specify library-based code location in the manifest and how to launch it.

Appendix D. Notes from Breakout Session on Execution Hardware Requirements

Purpose of including hardware specifications:

- Cloud computing: Capturing the hardware requirements is very useful.
- Ensure interoperability by stating what hardware a tool can run on or was validated to run on. Improve speed/full utilization of hardware capabilities.
- HPC computing: Specifying the hardware is a must. Hardware requirements impact reproducibility. (Example - different generations of GPU cards).
- **Suggestions:**
    - Record what the container was validated to run on. Metadata contains only hardware specifications that the developer validated.
        - To do this, we need hardware ontology or controlled vocabulary.
    - Define how the computation resource capabilities are exposed and picked up by the scheduler.
- **Lesson Learned from Others**:
    - Kubernetes + Argo workflow for orchestration.
    - Hewlett Packard Enterprise (HPE) provides the compute capability metadata inside a container.
    - Open Container Initiative: https://opencontainers.org/
    - Distributed Cell Profiler: https://github.com/DistributedScience/Distributed-CellProfiler
        - Amazon's ECS and spot fleet need to ask for AWS resources dynamically and cost-effectively.

Purpose of ontologies for hardware specifications:

- Problem with translating metadata across different infrastructures.
- Controlled vocabulary with translation to each platform should be sufficient (or our best effort).
- If the descriptions are simple (CPU yes/no, GPU yes/no, TPU yes/no), then an ontology is overkill.
- If the descriptions are complex and include Compute Unified Device Architecture (CUDA) compute capability, AVX level, etc., then an ontology is needed to ensure a typical frame of reference when using hardware specification terms.
- **Lessons Learned from Others:**
    - The Web Image Processing Pipeline (WIPP) system is using Kubernetes, SLURM, to define the "compute" hardware requirement specification:
        - https://github.com/usnistgov/fair-chain-compute-container

- SNMP (Simple Network Management Protocol) also has something for node description:
    - https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol
- Hardware tiers (for example, Amazon T1, T2, ...) or VFX reference platforms for execution benchmarks:
    - https://vfxplatform.com/
- FAIR Principles for Research Hardware:
    - https://www.rd-alliance.org/groups/fair-principles-research-hardware
- Cloud computing community to define mOSAIC ontology as one part of IEEE 23002
    - https://groups.oasis-open.org/higherlogic/ws/public/document?document_id=46205:
    - https://www.sciencedirect.com/science/article/pii/S0167739X17330467?via%3Dihub#sec3

Suggested hardware information to capture:

- Min resource usage constraints (CPU, memory).
- Type of storage.
- Specific requirements (GPU, specific architecture or instruction sets, page size…).
- Topology information (colocation of workloads).
- Latency requirements.
- Range of values (SLURM doesn't allow stating a range, only a minimum; Command workflow language might work).
- Workshop participants highly varied on what they thought should be captured. Generally, there is a split between HPC users/maintainers and "common" users.
    - HPC is about maximizing the performance of hardware; therefore, the more specified hardware, the better performance.
    - General users want the software to "just work" and, therefore, have minimal expectations of definitions; therefore, whether the software runs optimally is irrelevant.
- Many workshop participants believed that "very simple" descriptors, e.g., x86 yes/no, GPU yes/no, and TPU yes/no, should be required with all other "more advanced" specifications as optional.
- **Utilization of the hardware information:**

- Some orchestration solutions can use the hardware information to manage cost, decide workloads (stop, pause, and reschedule when appropriate), and report estimated execution time.
- Argo workflow allows specifying resources when requesting containers. A user can set limits and hardware requests.
- The manifest definition of computing requirements must be translated into the specific scheduler used to orchestrate and launch the containers. TOIL workflow engine: https://toil.readthedocs.io/en/latest/index.html

Hardware discovery:

- Platform specific. Current solutions involve homebrewed impedance matching requirements to specific hardware devices.
- Necessity to query into the job scheduler and update metadata field.
    - Example: snake_make with SLURM and Prefect on AWS.
- It is about defining the container runtime touch points with the OS.
    - For example, glibc needs specific versions for runtime impedance matching.
- **Challenges of hardware discovery:**
    - How can you detect what the available resources are at runtime?
    - Depends on the workflow manager configuration.

Hardware requirements for tests and validations:

- Optimal resource allocation:
    - Ahead of time, how many resources do we need?
        - Allow resource optimization by tracking and comparing executions against past usage and executions. Next, adjust the resources allocated. Bin input data for each size. Various strategies to monitor an execution profile.
- Testing and validation:
    - Given a set of inputs, what anticipated hardware requirements can we rely on?
        - Containers should be hardware architecture-specific.
        - How much detail should be exposed? This raises the question of where the burden should be put on the developer or the sysadmin.
    - Given a software test that passed in one hardware environment, does the test translate to my environment?
        - Conda style validation needed/desired here.

        - Needed openly available diverse computing platforms for anyone to test their tool to "guarantee" when/where/which platforms and to what extent a container can run.

- **Lessons Learned from Others:**
    - Conda and automated build tests to offer a community level of assurance for given libraries.
    - Open Container Initiative (OCI) only has two fields included for hardware: arch and OS; the rest is optional and up to the vendor.
        - https://github.com/opencontainers/runtime-spec
        - https://github.com/opencontainers/image-spec
    - AIRFLOW, at some point, orchestrated both resource allocation and job scheduling (complex configuration).
    - Common Workflow Language (CWL) and Workflow Description Language (WDL) have only elementary computational fields exposed with minimal/no hardware acceleration supported.

Metadata about hardware requirements for benchmarking:

- Benchmarking code and then predicting how it will behave on another system are unreliable steps and hard to do.
- Three different levels of details for each category of the hardware spec (required hardware - lower bound limits, tested hardware specs, benchmark hardware specs).
- **Metadata recommendations:**
    - Run pre-existing benchmarks and pre-run them for each container. Use existing standard measurements.
    - Standardize the benchmarks running on different environments and have a portable ready-to-go virtual machine (VM) that can be a standard when performing the benchmarks.
- **Lessons Learned from Others:**
    - Nextflow is a workflow language that hosts community pipelines at https://nf-co.re/. The hosting of community pipelines is an example of a community effort to collect analysis pipelines built using Nextflow.

Governance of manifest:

- Ontologies and controlled vocabularies:
    - Dealing with reproducibility
    - Dealing with the constant evolution of technology
- Validation of manifest specifications (even if the specification is just for a controlled vocabulary instead of a full ontology):

    - https://www.commonwl.org/v1.2/CommandLineTool.html#ResourceRequirement
    - https://www.commonwl.org/user_guide/introduction/basic-concepts.html
- Having “too many” specifications can be an issue, too; describing generic guidelines could be more useful.
- **Governance protocol:**
    - At least an RFC with a comment period so interested parties can express an opinion.
    - Minimal experience among the workshop participants.