



## SOFTWARE TOOL ARTICLE

# **REVISED** Sapporo: A workflow execution service that encourages the reuse of workflows in various languages in bioinformatics [version 2; peer review: 2 approved, 2 approved with reservations]

Hirota Suetake <sup>1</sup>, Tomoya Tanjo<sup>2</sup>, Manabu Ishii<sup>3</sup>, Bruno P. Kinoshita<sup>4,5</sup>, Takeshi Fujino<sup>6</sup>, Tsuyoshi Hachiya<sup>3</sup>, Yuichi Kodama <sup>2</sup>, Takatomo Fujisawa<sup>2</sup>, Osamu Ogasawara <sup>2</sup>, Atsushi Shimizu<sup>2</sup>, Masanori Arita<sup>2</sup>, Tsukasa Fukusato<sup>7</sup>, Takeo Igarashi<sup>1</sup>, Tazro Ohta<sup>8-10</sup>

<sup>1</sup>Department of Creative Informatics, Graduate School of Information Science and Technology, The University of Tokyo, Bunkyo, Tokyo, Japan

<sup>2</sup>Bioinformation and DDBJ Center, National Institute of Genetics, Mishima, Shizuoka, Japan

<sup>3</sup>Genome Analytics Japan Inc, Shinjuku, Tokyo, Japan

<sup>4</sup>Barcelona Supercomputing Center (BSC), Barcelona, Spain

<sup>5</sup>Curii Corporation, Somerville, MA, USA

<sup>6</sup>Department of Computational Biology and Medical Sciences, Graduate School of Frontier Sciences, The University of Tokyo, Bunkyo, Tokyo, Japan

<sup>7</sup>Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo, Bunkyo, Tokyo, Japan

<sup>8</sup>Institute for Advanced Academic Research, Chiba University, Chiba, Japan

<sup>9</sup>Database Center for Life Science, Joint Support-Center for Data Science Research, Research Organization of Information and Systems, Mishima, Shizuoka, Japan

<sup>10</sup>Department of Artificial Intelligence Medicine, Graduate School of Medicine, Chiba University, Chiba, Chiba, Japan

**v2** First published: 04 Aug 2022, 11:889  
<https://doi.org/10.12688/f1000research.122924.1>

Latest published: 24 Jun 2024, 11:889  
<https://doi.org/10.12688/f1000research.122924.2>

## Abstract

The increased demand for efficient computation in data analysis encourages researchers in biomedical science to use workflow systems. Workflow systems, or so-called workflow languages, are used for the description and execution of a set of data analysis steps. Workflow systems increase the productivity of researchers, specifically in fields that use high-throughput DNA sequencing applications, where scalable computation is required. As systems have improved the portability of data analysis workflows, research communities are able to share workflows to reduce the cost of building ordinary analysis procedures. However, having multiple workflow systems in a

## Open Peer Review

Approval Status

	1	2	3	4
<b>version 2</b> (revision) 24 Jun 2024		 view	 view	 view
		↑		
<b>version 1</b> 04 Aug 2022	 view	 view		

1. **Justin M. Wozniak** , Argonne National Laboratory, Lemont, USA

research field has resulted in the distribution of efforts across different workflow system communities. As each workflow system has its unique characteristics, it is not feasible to learn every single system in order to use publicly shared workflows. Thus, we developed Sapporo, an application to provide a unified layer of workflow execution upon the differences of various workflow systems. Sapporo has two components: an application programming interface (API) that receives the request of a workflow run and a browser-based client for the API. The API follows the Workflow Execution Service API standard proposed by the Global Alliance for Genomics and Health. The current implementation supports the execution of workflows in four languages: Common Workflow Language, Workflow Description Language, Snakemake, and Nextflow. With its extensible and scalable design, Sapporo can support the research community in utilizing valuable resources for data analysis.

### Keywords



workflow, workflow language, workflow execution service, open science



This article is included in the [Japan Institutional Gateway](#) gateway.



This article is included in the [Bioinformatics gateway](#).

2. **Iacopo Colonnelli** , Università degli Studi di Torino, Turin, Italy
3. **Denis Yuen** , Ontario Institute for Cancer Research, Toronto, Canada
4. **Stephen R. Piccolo**, Brigham Young University, Provo, USA

Any reports and responses or comments on the article can be found at the end of the article.

**Corresponding author:** Tazro Ohta ([tazro.ohta@chiba-u.jp](mailto:tazro.ohta@chiba-u.jp))

**Author roles:** **Suetake H:** Conceptualization, Funding Acquisition, Investigation, Methodology, Software, Writing – Original Draft Preparation; **Tanjo T:** Software; **Ishii M:** Data Curation, Software; **P. Kinoshita B:** Software, Writing – Review & Editing; **Fujino T:** Data Curation; **Hachiya T:** Data Curation, Writing – Review & Editing; **Kodama Y:** Conceptualization; **Fujisawa T:** Conceptualization, Resources; **Ogasawara O:** Conceptualization, Resources; **Shimizu A:** Conceptualization; **Arita M:** Conceptualization, Funding Acquisition; **Fukusato T:** Supervision, Writing – Review & Editing; **Igarashi T:** Funding Acquisition, Supervision, Writing – Review & Editing; **Ohta T:** Conceptualization, Investigation, Methodology, Project Administration, Software, Supervision, Writing – Original Draft Preparation

**Competing interests:** No competing interests were disclosed.

**Grant information:** This study was supported by JSPS KAKENHI (Grant Number 20J22439; assigned to H.S.), the Life Science Database Integration Project, and the National Bioscience Database Center (NBDC) of the Japan Science and Technology Agency (JST). DDBJ is supported by the Research Organization of Information and Systems (ROIS) under the Ministry of Education, Culture, Sports, Science, and Technology (MEXT) of Japan. This study was also supported by the CREST program of the Japan Science and Technology Agency (Grant Number JPMJCR17A1, assigned to T.I.).

*The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.*

**Copyright:** © 2024 Suetake H *et al.* This is an open access article distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

**How to cite this article:** Suetake H, Tanjo T, Ishii M *et al.* **Sapporo: A workflow execution service that encourages the reuse of workflows in various languages in bioinformatics [version 2; peer review: 2 approved, 2 approved with reservations]** F1000Research 2024, 11:889 <https://doi.org/10.12688/f1000research.122924.2>

**First published:** 04 Aug 2022, 11:889 <https://doi.org/10.12688/f1000research.122924.1>

**REVISED Amendments from Version 1**

In response to reviewer comments, the manuscript was updated to emphasize the challenges in creating universal workflow systems due to differences in syntax and engine features, highlighting the need for standardized workflow specifications and support for various engines. The user's procedure for performing analysis with Sapporo was clarified, with a detailed view added as Figure 8. The discussion now addresses the workflow depiction in Figure 7, clarifying the relationship between the user interface and workflow usability. Additionally, the manuscript explains how Sapporo addresses inefficiencies and fragmentation by wrapping multiple systems and using Docker containers for workflow engines, adhering to the GA4GH WES standard for interoperability. Documentation now includes the location of the Docker compose manifest for Sapporo-service and Sapporo-web, and the Methods section details the run.sh function of Sapporo-service, highlighting its modularity, extensibility, and role in managing workflow executions and environment-specific requirements.

**Any further responses from the reviewers can be found at the end of the article**

**Background**

Modern experimental instruments that convert biological samples into digital data have lower costs and higher throughput than conventional ones.<sup>1</sup> Those instruments have made it possible to conduct large-scale data-driven biology, not only in large projects but also in smaller studies. A DNA sequencer is one such technology in biology, which has shown a drastic improvement in throughput since the late 2000s.<sup>1</sup> DNA sequencing technology highlighted the data science aspect of biology, sparking the demand for computation in biology.<sup>2</sup>

Raw data, the fragments of nucleotide sequences for a DNA sequencer, often called “reads,” are not biologically interpretable in their unprocessed form. Researchers need to process the data using computational methods to obtain biological insights from the samples. The data processing includes, for example, estimation of sequence error rates, read alignment to a reference genome sequence, extraction of genomic features from aligned data, and annotation with the information obtained from public databases. Researchers develop and share the command-line tools for each step in an analysis. They use the raw data as the initial input data of the first tool and pass its output on as input for the next tool. This chain of processes, connecting a sequence of tools according to their inputs and outputs, is called a workflow.<sup>3</sup>

Workflow structure can be complicated as various sequencing applications require multiple steps of data processing. Combining many tools to construct a complex workflow that performs as intended is not straightforward. It is also not practical to fully understand the internal processes of all the tools. Thus, ensuring that every individual part of a workflow is working correctly depends heavily on the skills of the workflow developer. Even if a workflow runs successfully once, maintaining it is another issue. The tools in a workflow are often developed as open-source software and are frequently updated to improve performance and fix bugs. It is time-consuming to assess the impact of updates associated with individual tools. The tools in a workflow often work in an unintended manner for many reasons, such as changes in hardware, operating system (OS), software dependencies, or input data. Difficulties in building and maintaining workflows cause portability issues with workflows.<sup>4</sup> Because of this, researchers have to spend a great deal of time building workflows similar to those that others have already created.

To address these issues, researchers have developed many workflow systems in bioinformatics.<sup>5</sup> Each workflow system has unique characteristics, but generally, they all have a language syntax and a workflow engine. Workflow languages define a syntax to describe the inputs and arguments passed to tools and the handling of outputs. Workflow engines often take two arguments to execute a workflow: a workflow definition file that specifies the processes and a job file for input parameters. In many cases, techniques, such as package managers and container virtualization, make it easier to build, maintain, and share complex workflows by pinning down the versions of workflow tools.<sup>6</sup>

Open-source workflow systems help the research community work efficiently by reusing published workflows.<sup>7</sup> However, having multiple systems has resulted in resources distributed across various workflow system communities. For example, the Galaxy community is known for being one of the largest for data analysis in biology.<sup>8</sup> The community maintains a number of workflows and learning materials that users can run on public Galaxy instances. However, as the Galaxy workflows are only runnable on the Galaxy platform, users will face difficulties in running these workflows on other platforms. As another example, Nextflow, one of the most popular command-line-based workflow systems, has a mature community called nf-core to share standard workflows.<sup>9,10</sup> The community has excellent resources, but these are usable only by Nextflow users. It is not reasonable to have a “one-size-fits-all” workflow system in science because various approaches have pros and cons.<sup>3</sup> Learning the different concepts and features of each workflow system has a high cost associated with it. Thus, it is not practical to consider becoming familiar with a large number of workflow systems in order to be able to utilize the workflows shared by their community users.

Workflow systems have different language syntaxes and engines, each designed for specific purposes. For instance, Nextflow aims to boost developer productivity and scalability, while Snakemake focuses on flexibility and simplicity, using Python as its base. In contrast, the Common Workflow Language (CWL) project aims to promote interoperability by creating a standardized syntax that various workflow engines can understand. However, workflows written in different languages cannot be easily converted into each other automatically. The most popular workflow systems used in bioinformatics, such as CWL, WDL, Nextflow, and Snakemake, take a workflow definition and input parameters to produce output result files, while there are differences between these workflow systems in command-line options, workflow description syntax, methods for specifying inputs, and how expected output files are defined.

Creating a universal language converter isn't practical because some languages lack the necessary syntax parsers, or contain features that are not commonly found in other workflow engines (e.g. JavaScript evaluation as in CWL, loops in workflows or cyclic workflows instead of DAG-based systems). To bridge the gap between different workflow systems, we need a standardized way to specify workflows, input parameters, and expected outputs. Additionally, a system that supports various engines and selects the appropriate one for a given workflow is essential for smooth interoperability.

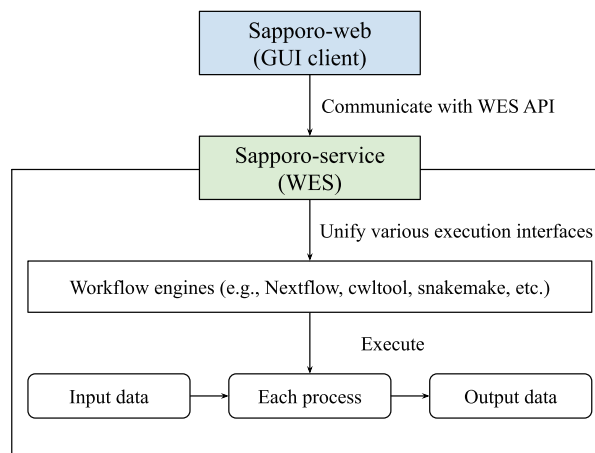
In this paper, we introduce Sapporo, a platform for running multiple workflow systems in the same computing environment. Sapporo wraps the differences in the workflow systems and provides an application programming interface (API) for executing them in a unified way. Sapporo also provides a graphical user interface (GUI) that works as its API client. By enabling users to run multiple workflow systems on the same computing environment, Sapporo gives users the ability to reuse workflows without having to learn a new workflow system.

## Methods

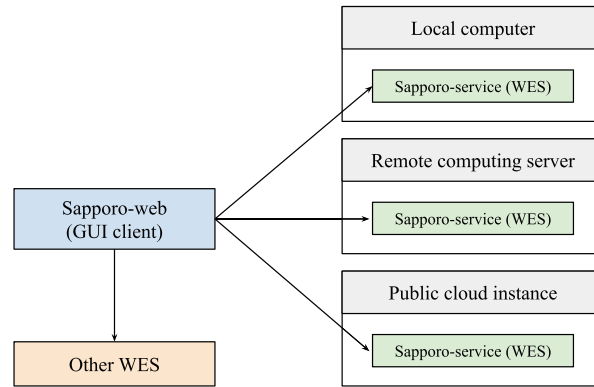
### System overview

Sapporo consists of two components: Sapporo-service and Sapporo-web (Figure 1). Sapporo-service is an API that receives requests for workflow execution from clients, then executes them in a specified manner. Sapporo-service has an API scheme that satisfies the Global Alliance for Genomics and Health (GA4GH) Workflow Execution Service (WES) standard.<sup>11</sup> Sapporo-web is a workflow management client. It is a client of Sapporo-service and other GA4GH WES compatible API servers. The GUI is a browser-based application that does not require user installation.

We designed the Sapporo system based on the concept of microservices architecture.<sup>12</sup> Unlike conventional computation server applications, we expect multiple Sapporo-service instances to be run on servers as independent endpoints on demand. To manage the runs on the different API servers, we separate the implementation of the server and its client, allowing clients to connect to multiple servers (Figure 2). One of the unique features of the Sapporo system is that it has no authentication mechanism on the application layer. Instead of having users' information on the server-side, the user's web browser stores the information, such as workflow execution history. The online documentation "Sapporo: Getting Started", available in *Extended data*, shows the step-by-step procedures to deploy a Sapporo instance on a local computer to test the system.<sup>13,34</sup>



**Figure 1. Overview of the Sapporo system.** The component at the bottom is Sapporo-service, a Global Alliance for Genomics and Health (GA4GH) Workflow Execution Service (WES) standard compatible application programming interface (API) to manage the workflow execution. The box at the top is Sapporo-web, the graphical user interface (GUI) client for WES implementations. Sapporo-service has the open specification of the API endpoints, which users can access programmatically.



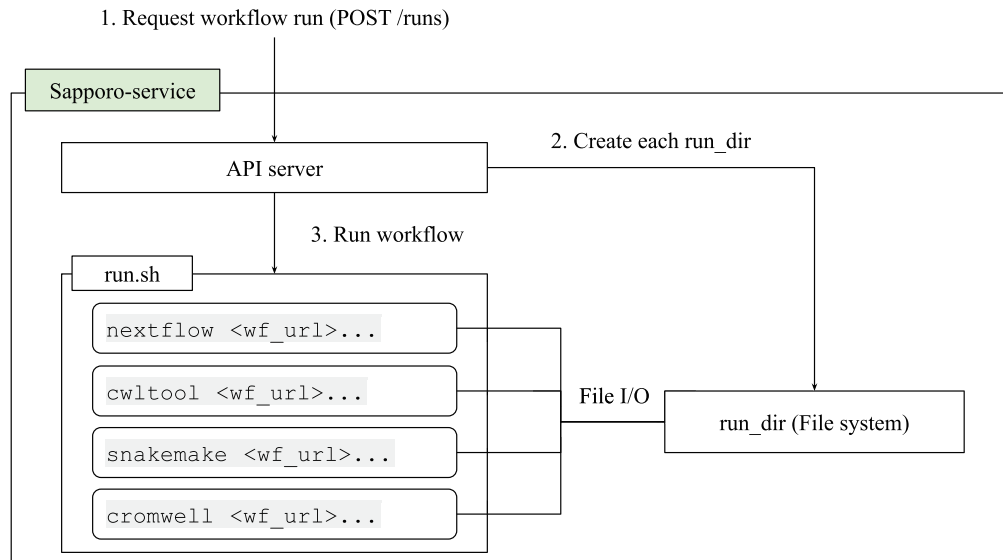
**Figure 2. The distribution model of the Sapporo system.** Researchers often have multiple computing environments for their data analysis. We designed the Sapporo system to work with a distributed computation model. For example, users can deploy the application programming interface (API) on their local computer, remote computing server, and public cloud instances. As long as the API is running, the user can send a request to execute a workflow from a local client computer.

The source code, test code, and documentation for Sapporo-service and Sapporo-web are available from GitHub and archived in Zenodo.<sup>35,36</sup>

### Workflow execution service

The WES has two layers: the API and the execute function (Figure 3). The API structure and the response are compliant with the GA4GH WES standard.<sup>14</sup> The API specification defines the methods to manipulate workflow runs, such as execution, stop, and checking the outputs. In addition, Sapporo-service has its own unique features (Table 1). The key feature that makes Sapporo notable is the workflow engine selection. While the other workflow management systems accept one or a few workflow languages, Sapporo-service can accept any workflow language as long as it has a corresponding workflow engine.

The system is designed to separate the execution layer from the handling of API requests, thereby enhancing modularity and extensibility. The execution layer operates through a well-structured shell script named “run.sh.” Upon receiving an



**Figure 3. The Sapporo-service components.** Sapporo-service’s application programming interface (API) layer implemented in Python works as an API server to receive the request of a workflow run. The system can be deployed easily by using Docker compose manifest provided in the GitHub repository (See Software availability). Once the API receives the request, it creates a directory to store all the related information and execute run.sh. The run.sh script receives the arguments from the API request and runs the workflow with the specified workflow engine.

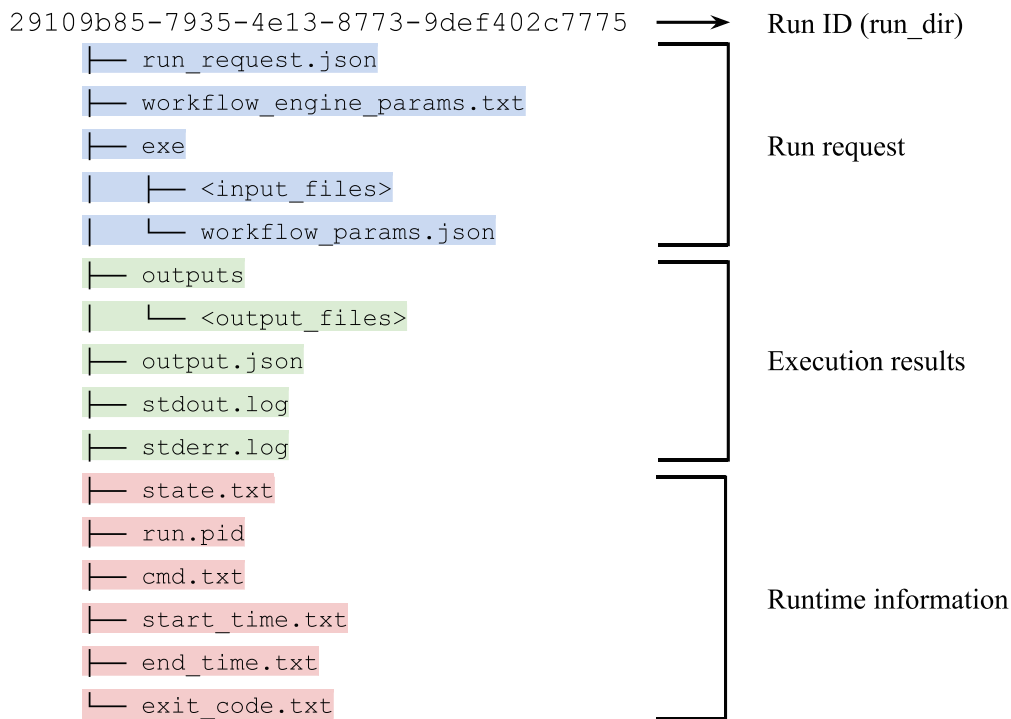
**Table 1. The list of Sapporo-service's features.**

Feature	Description
Engine selector	Select engine from available implementations
Remote URL as attachment	Fetch remote file and attach to the run
Output downloader	Direct download of workflow results
Registered-only mode	Restrict workflows by allowed-list
Workflow parser	Return parsed workflow information

**Table 2. The list of workflow engines available in Sapporo.**

Engine	Supported language(s)
cwltool <sup>15</sup>	CWL
Nextflow <sup>9</sup>	Nextflow
Toil <sup>16</sup>	CWL
Cromwell <sup>17</sup>	WDL, CWL
Snakemake <sup>18</sup>	Snakemake
ep3 <sup>19</sup>	CWL
StreamFlow <sup>20</sup>	CWL

API request, the system forks “run.sh,” which then generates command lines for the workflow system and executes them. This separation enables the addition of new workflow systems without changes to the API server’s code. As a result, adding new workflows becomes straightforward, with the number of systems growing from just one at the beginning of



**Figure 4. The contents in the per-run directory.** Various types of information, such as run requests, execution results, and runtime information, are stored as a bundle of provenance for the workflow run.

the project to seven in the current version (Table 2). The flexibility of the “run.sh” also allows for specific adjustments for each workflow system, supporting pre- and post-execution processes, such as authentication, staging input files, and uploading results. Additionally, it is enabled to manage environment-specific requirements, including executing jobs on grid engines and handling file I/O with S3-like object storage. Once the system receives a workflow run request, it issues a universally unique identifier (UUID) and creates a directory named with the UUID, where the system stores all the necessary files. The workflow definition files, intermediate and final outputs, and the other metadata are stored in that directory. This per-run directory can act as a bundle of provenance for the workflow run (Figure 4).

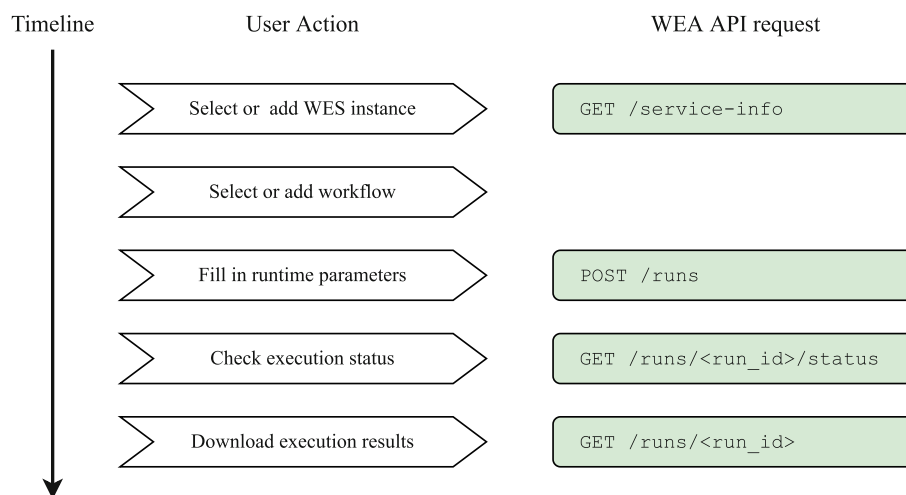
The system has no backend database as it stores all the information in the file system. This architecture allows the system administrators to manage the data as they do for normal server operations. We also provide a Docker image of the application, which can completely separate the system into the application (container image) and data (file system) for better portability and scalability.<sup>21</sup>

Another feature not implemented in a standard WES server is a registered-only mode. By enabling it at the server start-up, users can execute only the workflows in the allowed list specified by the administrator. This function helps the administrators launch a public WES instance while preventing suspicious programs from running on the server. Instead of implementing user authentication on the application, we expect the administrators to do the required authentication to the server on the network layer, such as virtual private network (VPN).

### Workflow management console

We designed Sapporo-web as a browser-based GUI client for GA4GH WES endpoints. Sapporo-web can also be easily deployed by using the Docker compose manifest provided in the GitHub repository (See Software availability). The system is a JavaScript application that runs on a web page, which users do not need to install on their computers. It stores user data in the browser’s local storage, so users do not need to sign up to start running workflows. No information other than the access log is preserved on the server-side. The Sapporo-web system is compliant with the GA4GH WES specification. We used Elixir WES, another WES implementation, to confirm Sapporo’s GA4GH WES specification compliance.<sup>22</sup>

To execute a workflow with Sapporo-web, users take the following five steps (Figure 5). Users can use a WES endpoint either running remotely or locally. Following the user’s connection request, Sapporo-web requests the service-info API of the WES to read the endpoint metadata and display the information (Figure 6). Users can select a workflow to run by entering a published workflow URL, uploading a workflow definition file, or selecting from the workflows registered on the WES server. Sapporo-web also can accept the GA4GH Tool Registry Service (TRS) protocol as a source of published workflows. Sapporo-web retrieves the content of the requested workflow definition file to generate a web form for entering input parameters (Figure 7). The type of web form depends on the workflow language. For example, loading a



**Figure 5. User actions to execute a workflow on Sapporo-web.** Sapporo-web provides a step-by-step user interface to help users set up a workflow run. First, users need to specify where to execute a workflow (Workflow Execution Service (WES) instance). Next, users select what to run (workflow) and then how it should be run (input parameters). The UI allows users to download a set of input parameters, which users can upload to re-run a workflow with the same parameters.



The screenshot shows the Sapporo-web interface for an 'Example Service'. The service endpoint is `http://localhost:1122` and it was added on 2022-04-15 17:22:21. The 'Workflow Engine Versions' section lists the following engines: Cromwell (v72), Cwltool (v3.1.20211107152837), EP3 (experimental) (v1.7.0), Nextflow (v21.10.5), Snakemake (v6.9.1), Streamflow (v0.1.3), and Toil (experimental) (v4.1.0). Below this, a table titled 'Workflows' lists 9 workflows with their names and added/updated dates. The workflows are: `cwltool_attach_all_files`, `cwltool_remote_workflow`, `nextflow_file_input`, `nextflow_params_outdir`, `nextflow_str_input`, `snakemake_tutorial_wf`, `cromwell_bamstats_cwl`, `cromwell_bamstats_wdl`, and `trimming_and_qc_remote`. The table has columns for 'Name' and 'Added / Updated Date'. At the bottom of the table, there are 'ADD' and 'IMPORT' buttons.

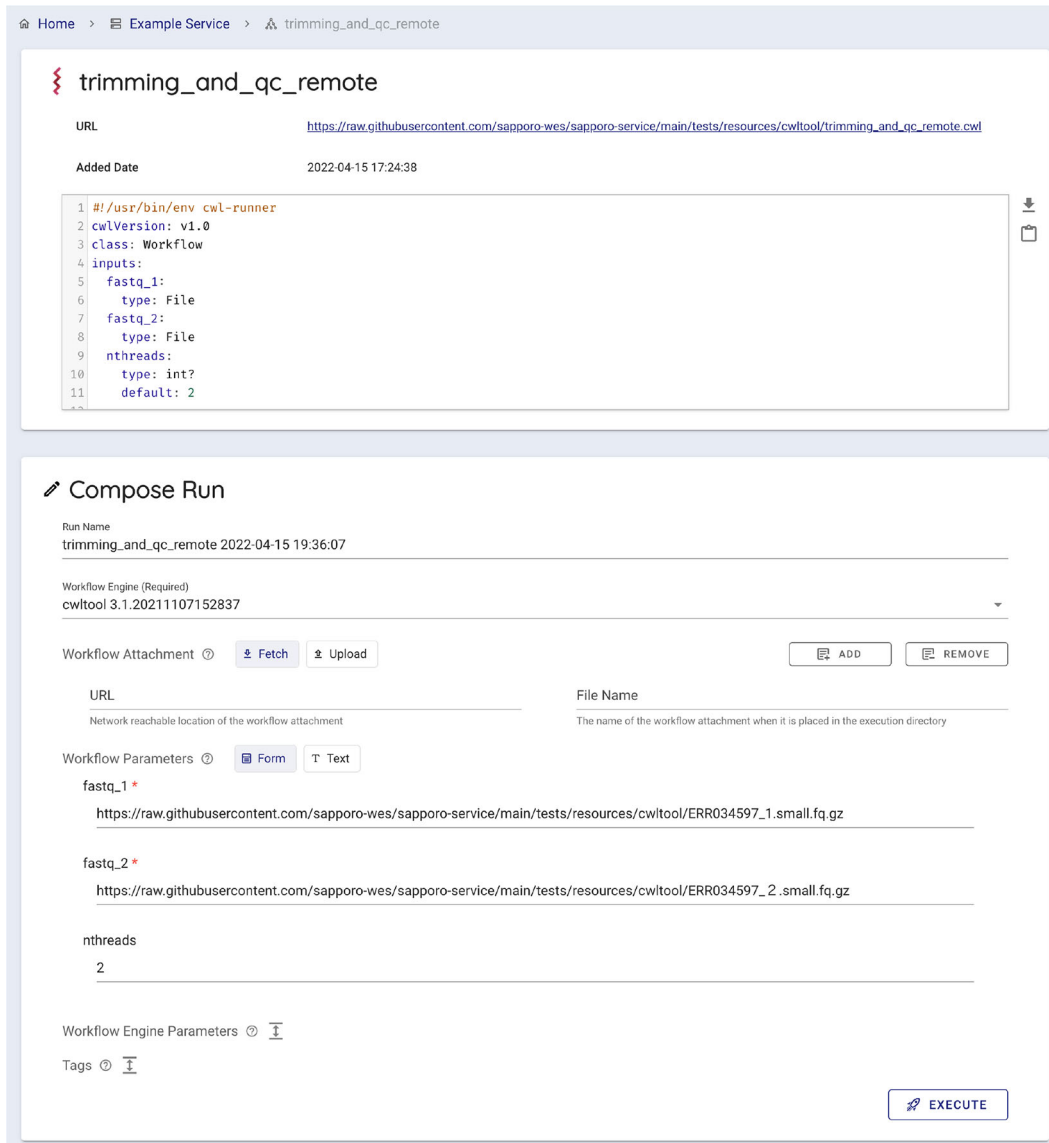
**Figure 6. Sapporo-web displays the metadata of the specified Workflow Execution Service (WES) endpoint.** The Global Alliance for Genomics and Health (GA4GH) WES specification defines the scheme of the response of service-info. It has the basic information of the WES endpoint, such as supported workflow language and workflow engines. Sapporo-web reads the WES metadata and provides the interface to start composing a workflow run.

workflow described in Common Workflow Language (CWL) generates a typed input form per parameter because CWL specifies input parameters with a structured text form.<sup>23</sup> In contrast, loading a workflow described in languages other than CWL generates a text editor to change the parameters in the corresponding format. After the edit, users can click “execute” to request the workflow to run on the server where the WES endpoint is running.

While the workflow is running, users can check the execution log via Sapporo-web. The standard output and the standard error of the workflow run retrieved from the WES endpoint show up in the log history section. The running status becomes “complete” when the execution finishes on the server. Workflow outputs stored in the WES server are downloadable via a link in the Sapporo-web user interface. If the workflow run failed with an error, the status “executor error” would be shown. Users can visualize the error log in Sapporo-web.

## Results

We developed Sapporo as a WES implementation that allows developers to add new workflow systems. Developers only need to implement the command line procedure in the `run.sh` script, which is a simple bash script. The project was hosted on GitHub from its inception, with the intention to have other developers contribute with new workflow systems. A good example of this in practice can be seen in the pull request (<https://github.com/sapporo-wes/sapporo-service/pull/29>) that added a new workflow system called StreamFlow.<sup>20</sup>



**Figure 7. The generated web form to input parameters.** Sapporo-web automatically generates a typed input form for a given workflow. It is possible when the given workflow language has a structured job configuration file, for example, a YAML format file for a Common Workflow Language (CWL) workflow. The form has the type validation function for users' input, such as a file, integer, or array of strings.

To evaluate the practical applicability and robustness of Sapporo, we executed the public workflows that researchers frequently use. Specifically, we chose the Mitochondrial Short Variant Discovery workflow from the GATK best practices (written in WDL), the RNA-seq workflow from the nf-core repository (written in Nextflow), and a Germline Short Variant Discovery workflow for processing whole-genome sequencing data from the Japanese Genotype-phenotype Archive (written in CWL).<sup>24</sup> Users access Sapporo's endpoint specifying the input parameters following the WES specification. The required parameters are workflow\_url, workflow\_type, workflow\_type\_version, and workflow\_params. The workflow\_url argument specifies the location of the workflow definition file (e.g. CWL file) to be executed, typically hosted on a remote server, enabling the API to access and utilize the workflow's instructions. The workflow\_params argument points to a JSON file containing input parameters essential for the workflow execution, facilitating customization and adaptation of the workflow's behavior. The optional arguments workflow\_type and workflow\_type\_version arguments indicate the type and version of the workflow language being employed, ensuring compatibility and proper interpretation of the workflow instructions by engines supported inside Sapporo. Additionally, the workflow\_engine\_name argument specifies the execution engine to be used, while the default engine for the given

workflow language is assigned when it is not specified. Lastly, another optional argument `workflow_engine_parameters` argument allows for the specification of additional parameters tailored to the execution engine, providing fine-grained control over the execution environment and behavior of the workflow engine. We published the detailed description of the test procedures for these workflows on GitHub,<sup>25</sup> and the results of the test runs on Zenodo.<sup>26–28</sup>

Using a simple CWL workflow as an example, we describe the procedures we performed in the evaluation (Figure 8). It is noteworthy that despite changes in workflow languages, the steps remain the same, differing only in the supplied workflow definition file or the runtime parameters specified within the designated files. Firstly, the Sapporo-service is initiated within a computational environment. There are two methods for initiating the service: one involves executing a Python program natively, and the other utilizes our Docker image. If Docker or a Docker-compatible Linux container system is available, using the Docker image is simpler. Once the Service is initiated, by default, the API is available via port 1122. The workflow can be executed by sending a POST request to the `/runs` endpoint of this API. The POST request must include the location of the definition file for the workflow to be executed and the runtime parameters as a part of the URL parameters. Requests to the Sapporo-service can be made by using command-line programs such as `curl`, scripts written in any programming language, or via our developed web UI, Sapporo-web. Here, we explain the method using `curl`. Assuming that the Sapporo-service is running on port 1122 of the localhost, the `curl` command for the request would be as follows:

```
curl -X POST -F "workflow_url=https://raw.githubusercontent.com/pitagora-network/pitagora-cwl/master/workflows/download-fastq/download-fastq.cwl" -F "workflow_type=CWL" -F "workflow_type_version=v1.0" -F "workflow_engine_name=cwltool" -F "workflow_params=<workflow_params.json" -F "workflow_engine_parameters=<workflow_engine_parameters.json" http://localhost:1122/runs
```

In this request, a CWL workflow named `download-fastq`, publicly available on GitHub, is specified. The type of workflow is CWL, with version v1.0, and the workflow engine designated for executing this workflow is `cwltool`. While there are workflow languages like CWL that can be executed by multiple engines, there are also languages like Nextflow that can only be executed by the nextflow program. Therefore, users must choose the appropriate engine here; otherwise, errors will occur. Information on which engines support which languages can be retrieved via API requests. Parameters to be supplied to the `download-fastq` workflow for execution via `workflow_params`, and parameters to be supplied to the workflow engine `cwltool` via `workflow_engine_parameters` are specified. Both are described within JSON files and attached to the request as files. Upon receiving this request, the API server issues a UUID to identify this workflow run and returns it as part of the API response. Using this UUID, users can check the status of the run or download results after execution. This API, compliant with GA4GH WES, is straightforward, allowing for the execution of workflows written in

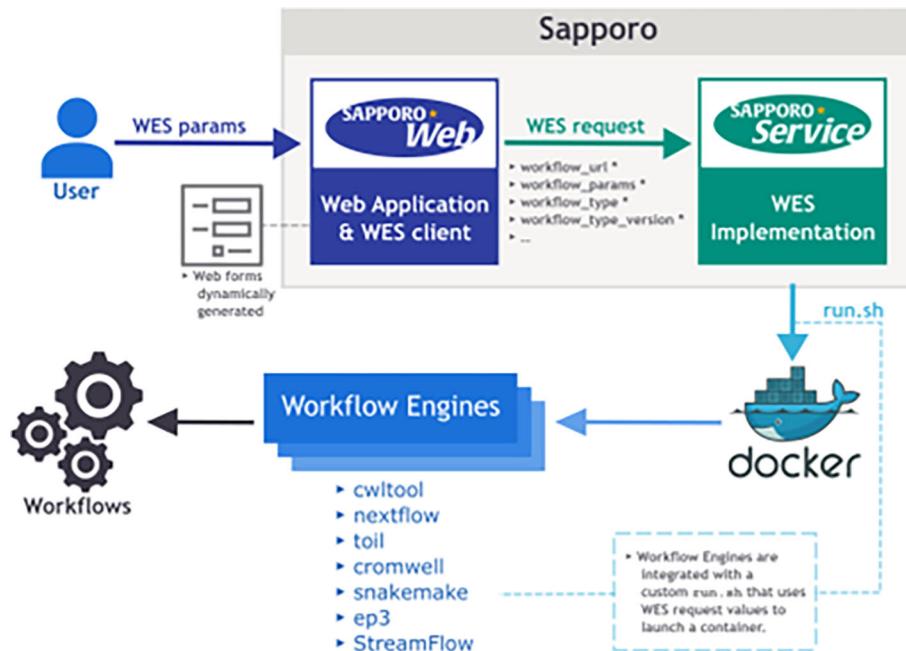


Figure 8. The detailed view of users procedure to run a workflow on Sapporo.

various workflow languages within the same computational environment without needing to rewrite the client based on differences in workflow languages.

## Discussion

In the big-data era in biology, the demand for efficient data processing will never stop increasing.<sup>29</sup> There are countless painful tasks in data processing, and researchers have developed methods to solve each of them, resulting in many different workflow systems.<sup>30</sup> We appreciate that many options are available as open-source so that researchers can choose one for their specific needs. The situation strongly encourages open science: each workflow system community is there so that individuals can help each other by sharing resources.<sup>31</sup> However, as each community grows, the gap between the communities also becomes larger. We developed Sapporo to bridge these gaps by providing a new layer to better utilize resources across communities. As the workflow systems are for the increased productivity of data scientists, improving resource interoperability must not interfere with researchers doing their science. An upper layer, the layer of workflow execution, can be a better solution than proposing a new language convertible to other existing languages. The concept of abstraction of workflow execution, as well as the idea of “bringing the algorithms to the data,” is also proposed by the GA4GH cloud work stream, which resulted in the development of the GA4GH Cloud standards. As of May 2022, the GA4GH WES specification supports only CWL and WDL for their workflow format. There is no official list of WES implementations; however, no other service that allows the addition of workflow systems is available as far as we investigated.

To support workflow developers and researchers conducting data analysis, multiple different workflow management systems have been developed. These systems enhance productivity and reproducibility in data analysis, enabling more effective science. However, the proliferation of multiple systems has revealed inefficiencies, leading to fragmentation within developer and user communities. While it’s crucial to effectively leverage the assets of each system and community, it is not practical to provide the methods for syntax conversion between workflow systems and extending execution engines. Therefore, Sapporo aims to absorb differences between systems by wrapping multiple systems. Specifically, we provide an API that rewrites workflow definitions and runtime parameters into the command lines of each system based on the type of workflow definition received, enabling the execution of different workflows using the same client. The Web API adheres to the internationally defined GA4GH WES standard, ensuring interoperability with other GA4GH WES implementations. By developing and releasing Sapporo Web as an example of a GA4GH WES client, we demonstrate the readiness of our developed API for research use.

Although Sapporo is a flexible system covering many use cases, we recognize that the current implementation has a few technical limitations. The main objective of Sapporo is to absorb the variance of the execution methods per workflow system. We achieved building a unified way to request a workflow run by providing the API and its client. However, there is still a challenge in the user experience with regard to the parameter editing function. This is caused by differences in the workflow system concepts. For example, some workflow systems, such as Nextflow or Snakemake, use Domain Specific Language (DSL) model in their syntax for better productivity, so users can write a workflow as they would write a script in their preferred programming language.<sup>9,18</sup> However, this flexibility in describing the procedures often makes the required input parameters unparseable by other applications. It means that users need to learn how to edit the parameters for each workflow system they are using. Though often this is not too difficult, the workflow system communities need to lower the learning costs to use a workflow. For example, finding a more generic representation of workflow inputs between workflow language systems could alleviate the situation.

Sapporo is a unique WES implementation that accepts multiple workflow languages. Researchers can use the system to utilize community workflows without regarding what language they are written in. One downside of this flexibility is that errors reported by Sapporo from different workflow engines may not look familiar to users. Many well-maintained workflow registries are available, such as nf-core and WorkflowHub, but the quality of the workflows published in these registries relies on each community’s efforts.<sup>10,32,33</sup> A system that validates and verifies the quality of workflows is also required for the sustainability of the resources published in the workflow registries.

Data processing methods vary greatly depending on the type of input data and the computational platform. In bioinformatics, the laboratory equipment and computers available drive changes. New computing applications for efficient data science, and new problems of resource portability may appear if variables such as input data, equipment, and computing resources keep changing in the future. Through its concept of abstraction, Sapporo can be a key player in assisting different communities in sharing and reusing workflows and other computing resources.

## Data availability

All of these projects are licensed under the Apache License 2.0.

## Underlying data

Zenodo: sapporo-wes/test-workflow: 1.0.1. <https://doi.org/10.5281/zenodo.6618935>.<sup>25</sup>

This project contains the following underlying data:

- sapporo-wes/test-workflow-1.0.1.zip (description of the test procedures and results of the workflows described in section *Use cases*).

The results of the test runs are contained in the following projects:

- Zenodo: Sapporo execution results - broadinstitute/gatk/MitochondriaPipeline: 1.0.0. <https://doi.org/10.5281/zenodo.6535083>.<sup>26</sup>
- Zenodo: Sapporo execution results - nf-core/rnaseq: 1.0.0. <https://doi.org/10.5281/zenodo.6534202>.<sup>27</sup>
- Zenodo: Sapporo execution results - JGA analysis - per-sample: 1.0.0. <https://doi.org/10.5281/zenodo.6612737>.<sup>28</sup>

## Extended data

Zenodo: sapporo-wes/sapporo: 1.0.0. <https://doi.org/10.5281/zenodo.6462774>.<sup>34</sup>

This project contains the following extended data:

- Sapporo: Getting Started.md (step-by-step procedures for deploying a Sapporo instance on a local computer and testing the system).

## Software availability

Sapporo-service's source code, test code, and documentation:

- Source code available from: <https://github.com/sapporo-wes/sapporo-service/tree/1.2.4>
- Archived source code at time of publication: <https://doi.org/10.5281/zenodo.6609570>.<sup>35</sup>
- License: Apache License 2.0

Sapporo-web's source code, test code, and documentation:

- Source code available from: <https://github.com/sapporo-wes/sapporo-web/tree/1.1.2>
- Archived source code at time of publication: <https://doi.org/10.5281/zenodo.6462809>.<sup>36</sup>
- License: Apache License 2.0

## Acknowledgements

We acknowledge and thank the following scientific communities and their collaborative events where several of the authors engaged in irreplaceable discussions and development throughout the project: the Pitagora Meetup, Workflow Meetup Japan, NBDC/DBCLS BioHackathon Series, Elixir's BioHackathon Europe Series, GA4GH Cloud WorkStream, Common Workflow Language Community, Nextflow Community, Galaxy Community, and Open Bioinformatics Foundation Bioinformatics Open Source Conference Collaboration Fest. We would like to acknowledge Dr. Alexander Kanitz for his support of the collaboration with WES-ELIXIR. We also would like to thank Dr. Ivan Topolsky for his assistance with the implementation of Sapporo-service. We also acknowledge Prof. Kazuki Yoshizoe for his valuable comments on the project. We also would like to thank Ascade Inc. for their support with the software development. Computations were partially performed on the NIG supercomputer at the ROIS National Institute of Genetics.

## References

1. Goodwin S, McPherson JD, McCombie RW, *et al.*: **Coming of age: Ten years of next-generation sequencing technologies.** *Nat. Rev. Genet.* 2016; **17**(6): 333–351.  
[PubMed Abstract](#) | [Publisher Full Text](#)
2. Stein LD: **The case for cloud computing in genome informatics.** *Genome Biol.* 2010; **11**(5): 207–207.  
[Publisher Full Text](#)
3. Perkel JM: **Workflow systems turn raw data into scientific knowledge.** *Nature.* 2019; **573**(7772): 149–150.  
[PubMed Abstract](#) | [Publisher Full Text](#)
4. da Leprevost FdV, Barbosa VC, Barbosa EL, *et al.*: **On best practices in the development of bioinformatics software.** *Front. Genet.* 2014; **5**: 199.  
[Publisher Full Text](#)
5. Wratten L, Wilm A, Göke J: **Reproducible, scalable, and shareable analysis pipelines with bioinformatics workflow managers.** *Nat. Methods.* 2021; **18**(10): 1161–1168.  
[PubMed Abstract](#) | [Publisher Full Text](#)
6. Leprevost FdV, Grüning BA, Aflitos SA, *et al.*: **Biocontainers: An open-source and community-driven framework for software standardization.** *Bioinformatics.* 2017; **33**(16): 2580–2582.  
[PubMed Abstract](#) | [Publisher Full Text](#)
7. Khan FZ, Soiland-Reyes S, Sinnott RO, *et al.*: **Sharing interoperable workflow provenance: A review of best practices and their practical application in cwlprov.** *GigaScience.* 2019; **8**(11): giz095.  
[Publisher Full Text](#)
8. Batut B, Hiltmann S, Bagnacani A, *et al.*: **Community-driven data analysis training for biology.** *Cell Systems.* 2018; **6**(6): 752–758.e1.  
[PubMed Abstract](#) | [Publisher Full Text](#)
9. Di Tommaso P, Chatzou M, Floden EW, *et al.*: **Nextflow enables reproducible computational workflows.** *Nat. Biotechnol.* 2017; **35**(4): 316–319.  
[Publisher Full Text](#)
10. Ewels PA, Peltzer A, Fillinger S, *et al.*: **The nf-core framework for community-curated bioinformatics pipelines.** *Nat. Biotechnol.* 2020; **38**(3): 276–278.  
[Publisher Full Text](#)
11. Rehm HL, Page AH, Smith L, *et al.*: **GA4GH: International policies and standards for data sharing across genomic research and healthcare.** *Cell Genomics.* 2021; **1**(2): 100029.  
[PubMed Abstract](#) | [Publisher Full Text](#)
12. Cerny T, Donahoo MJ, Trnka M: **Contextual understanding of microservice architecture: Current and future directions.** *ACM SIGAPP Applied Computing Review.* 2018; **17**(4): 29–45.  
[Publisher Full Text](#)
13. Suetake H, Ohta T: **Sapporo: Getting started.** 2021.  
[Reference Source](#)
14. The Global Alliance for Genomics and Health Cloud Work Stream: **Workflow Execution Service (WES) API.** 2017.  
[Reference Source](#)
15. Common Workflow Language: **common-workflow-language/cwltool.** 2015.  
[Reference Source](#)
16. Vivian J, Rao AA, Nothhaft FA, *et al.*: **Toil enables reproducible, open source, big biomedical data analyses.** *Nat. Biotechnol.* 2017; **35**(4): 314–316.  
[PubMed Abstract](#) | [Publisher Full Text](#)
17. Voss K, Van Der Auwera G, Gentry J: **Full-stack genomics pipelining with GATK4 + WDL + Cromwell.** 2017.  
[Reference Source](#)
18. Köster J, Rahmann S: **Snakemake—a scalable bioinformatics workflow engine.** *Bioinformatics.* 2012; **28**(19): 2520–2522.  
[Publisher Full Text](#)
19. Tanjo T: **tom-tan/ep3.** 2019.  
[Reference Source](#)
20. Colonnelli I, Cantalupo B, Merelli I, *et al.*: **Streamflow: Cross-breeding cloud with hpc.** *IEEE Trans. Emerg. Top. Comput.* 2020; **9**(4): 1723–1737.
21. Merkel D: **Docker: Lightweight linux containers for consistent development and deployment.** *Linux Journal.* 2014; **2014**(239): 2.
22. Harrow J, Drysdale R, Smith A, *et al.*: **ELIXIR: Providing a sustainable infrastructure for life science data at European scale.** *Bioinformatics.* 2021; **37**(16): 2506–2511.  
[PubMed Abstract](#) | [Publisher Full Text](#)
23. Crusoe MR, Abeln S, Iosup A, *et al.*: **Methods included: Standardizing computational reuse and portability with the common workflow language.** *arXiv.* 2021.
24. Kodama Y, Mashima J, Kosuge T, *et al.*: **The ddbj japanese genotype-phenotype archive for genetic and phenotypic human data.** *Nucleic Acids Res.* 2015; **43**(D1): D18–D22.  
[PubMed Abstract](#) | [Publisher Full Text](#)
25. Suetake H, Ohta T: **sapporo-wes/test-workflow: 1.0.1.** 2022.  
[Publisher Full Text](#)
26. Suetake H: **Sapporo execution results - broadinstitute/gatk/MitochondriaPipeline.** 2022.  
[Publisher Full Text](#)
27. Suetake H: **Sapporo execution results - nf-core/rnaseq.** 2022.  
[Publisher Full Text](#)
28. Suetake H: **Sapporo execution results - JGA analysis - per - sample.** 2022.  
[Publisher Full Text](#)
29. Prins P, De Ligt J, Tarasov A, *et al.*: **Toward effective software solutions for big biology.** *Nat. Biotechnol.* 2015; **33**(7): 686–687.  
[Publisher Full Text](#)
30. Amstutz P, Mikheev M, Crusoe MR, *et al.*: **Existing workflow systems.** 2021.  
[Reference Source](#)
31. Wilkinson MD, Dumontier M, Aalbersberg IJJ, *et al.*: **The fair guiding principles for scientific data management and stewardship.** *Sci. Data.* 2016; **3**(1): 1–9.
32. Goble C, Soiland-Reyes S, Bacall F, *et al.*: **Implementing FAIR digital objects in the EOSC-life workflow collaboratory.** 2021.
33. O'Connor BD, Yuen D, Chung V, *et al.*: **The dockstore: enabling modular, community-focused sharing of docker-based genomics tools and workflows.** *F1000Res.* 2017; **6**.  
[Publisher Full Text](#)
34. Suetake H, Ohta T: **sapporo-wes/sapporo: 1.0.0.** *Zenodo.* 2022.  
[Publisher Full Text](#)
35. Suetake H, Ohta T, Tanjo T, *et al.*: **sapporo-wes/sapporo-service: 1.2.4.** *Zenodo.* 2022.  
[Publisher Full Text](#)
36. Suetake H, Ohta T: **sapporo-wes/sapporo-web: 1.1.2.** *Zenodo.* 2022.  
[Publisher Full Text](#)

# Open Peer Review

Current Peer Review Status: ? ✓ ✓ ?

---

## Version 2

Reviewer Report 30 October 2024

<https://doi.org/10.5256/f1000research.167905.r294852>

© 2024 Colonnelli I. This is an open access peer review report distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.



**Iacopo Colonnelli** 

Universita degli Studi di Torino, Turin, Piedmont, Italy

The author successfully addresses most of the concerns affecting the first version of this work. For the remaining ones, they provided exhaustive and convincing justifications in the comments. The article is now mature enough to be indexed.

**Competing Interests:** No competing interests were disclosed.

**Reviewer Expertise:** Workflows, High Performance Computing, Parallel Computing, Distributed Computing

**I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.**

Reviewer Report 27 July 2024

<https://doi.org/10.5256/f1000research.167905.r305477>

© 2024 Piccolo S. This is an open access peer review report distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.



**Stephen R. Piccolo**

Department of Biology, Brigham Young University, Provo, UT, USA

This paper describes the Sapporo system, which is designed to be a Web-based wrapper around existing workflow engines. It allows users to specify workflows to be executed on various different engines on the back end. It is compatible with the GA4GH WES standard. The paper clearly describes motivations behind the software's existence, how a researcher might use it, and

limitations of the software. The software is open source, and examples of its use are provided. Personally, I am unsure whether I would use the software because it adds a layer of complexity. I would probably just use the underlying workflow engine if I already had a workflow file. Then again, if I typically used one particular workflow engine and found a better workflow in a different ecosystem, I might use Sapporo to facilitate execution. On the other hand, a challenge is that I would likely need to specify `workflow_engine_parameters` values that are specific that new workflow engine. That would take some time to figure out, so maybe I would just use the underlying workflow engine after all. Having said this, I think it is good that the software exists so that it is an option for researchers to use.

I have one more question/comment. CWL workflows can execute tasks within Docker. (I forget which other workflow systems support this.) However, the Sapporo system also uses Docker. It is tricky to run Docker from within Docker. So I am unsure how that situation is handled.

**Is the rationale for developing the new software tool clearly explained?**

Yes

**Is the description of the software tool technically sound?**

Yes

**Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?**

Yes

**Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?**

Yes

**Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?**

Yes

**Competing Interests:** No competing interests were disclosed.

**Reviewer Expertise:** Bioinformatics, genomics, workflow management systems, education

**I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.**

Reviewer Report 26 July 2024

<https://doi.org/10.5256/f1000research.167905.r305473>



© 2024 Yuen D. This is an open access peer review report distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.



**Denis Yuen** 

Ontario Institute for Cancer Research, Toronto, Ontario, Canada

This article largely covers the technical design decisions behind Sapporo, a suite of tools for facilitating workflow execution, while focusing on sapporo-service and sapporo-web.

Overall, the authors should be commended for their efforts in aiding reproducible science by making it simpler to run workflows using multiple workflow languages. Their efforts also sand-off the rough edges of the GA4GH workflow execution service (WES) standard making it easier for the community as a whole to develop tools to run workflows in a simpler uniform manner. Efforts to write standards and aid reproducible science can often be unsung but are deeply valuable to science as a whole.

Under the header of data and software availability, both GitHub repositories are well laid out and I was able to easily use docker-compose to spin up both components for some simple sanity checks. I also commend the authors for following best practices including well developed documentation for developers, continuous integration, DOIs on Zenodo, and last but not least regular software releases conforming to semantic versioning.

As for the manuscript, I recommend that the 2024 revision of the manuscript be accepted with only a couple minor suggestions for clarification.

There is one reference to 2022 in the discussion, "As of May 2022, the GA4GH WES specification supports only CWL and WDL for their workflow format." Since this revision is in 2024 and In light of the 1.1.0 release of the standard on Sept 14, 2023 it may be useful to update. For example, in table 2, it is clear Sapporo supports more than these two languages yet is built on top of WES in figure 2. Has the WES standard been expanded to more languages or did Sapporo need to make extensions to the standard?

Also in table 2, it should be noted that Toil has WDL support although this may/may not be supported by Sapporo.

**Is the rationale for developing the new software tool clearly explained?**

Yes

**Is the description of the software tool technically sound?**

Yes

**Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?**

Yes

**Is sufficient information provided to allow interpretation of the expected output datasets**

**and any results generated using the tool?**

Yes

**Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?**

Yes

**Competing Interests:** For full disclosure, I am a contributor to a different standard in the GA4GH cloud workstream and to Dockstore. Both are efforts aimed at improving reproducible science. That said, I have not been a collaborator or a co-author with any of the authors and have no financial or non-financial competing interests.

**Reviewer Expertise:** scientific workflows, cloud computing

**I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.**

---

Version 1

Reviewer Report 05 October 2023

<https://doi.org/10.5256/f1000research.134975.r185836>

© 2023 Colonnelli I. This is an open access peer review report distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.



**Iacopo Colonnelli** 

Universita degli Studi di Torino, Turin, Piedmont, Italy

In this article, the authors describe Sapporo, a WES-compatible web-based interface to multiple workflow management systems (WMSs). Sapporo aims to act as a common interface to different underlying WMSs, abstracting product-specific details and complexities to the end users. This way, Sapporo fosters WMSs interoperability and lowers the technical barriers between domain experts and workflow execution.

The description of Sapporo is kept at a high level of abstraction, without many technical details about the implementation. However, it is detailed enough to let the reader capture all the crucial aspects of the software architecture, the frontend and backend structure, and the main limitations of the current version. A link to a Docker Compose manifest for quick evaluation would be a plus.

For implementers, the description is too high-level to understand how to replicate the software development. Additional material published on GitHub and Zenodo contains further details for the developers. However, having at least a high-level description of the `run.sh` script, which constitutes the core of the Sapporo backend, would improve the article's understandability.

The main flaw of the article in its current form is that the description of the experimental evaluation and the obtained results is left to external references. The authors describe three different experiments using three different workflow languages and datasets. Adding a detailed description of one of them directly in the article, together with the steps needed to reproduce it, would allow the reader to better understand the features Sapporo provided.

Also, there is no quantitative measure of achieved results in the article. Some quantitative measures of the Sapporo complexity (e.g., the percentage of product-specific lines of code that users are still forced to write to specify workflow parameters or the average lines of code needed to add support for a new WMS) would enable a more scientifically sound evaluation of the product.

**Is the rationale for developing the new software tool clearly explained?**

Yes

**Is the description of the software tool technically sound?**

Yes

**Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?**

Partly

**Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?**

No

**Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?**

Partly

**Competing Interests:** No competing interests were disclosed.

**Reviewer Expertise:** Workflows, High Performance Computing, Parallel Computing, Distributed Computing

**I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.**

Author Response 13 Jun 2024

**Tazro Ohta**

We deeply thank you for taking the time to review our manuscript.

**1. A link to a Docker Compose manifest for quick evaluation would be a plus.**

Thank you for the great suggestion. We added a line to indicate the location of the docker compose manifest for each of Sapporo-service and Sapporo-web.

**2. For implementers, the description is too high-level to understand how to replicate the software development. Additional material published on GitHub and Zenodo contains further details for the developers. However, having at least a high-level description of the `run.sh` script, which constitutes the core of the Sapporo backend, would improve the article's understandability.**

We agree with your assessment. We modified the paragraphs in the subsection "Workflow execution service" in the Methods section to describe the run.sh function of Sapporo-service:

> *The system is designed to separate the execution layer from the handling of API requests, thereby enhancing modularity and extensibility. The execution layer operates through a well-structured shell script named "run.sh." Upon receiving an API request, the system forks "run.sh," which then generates command lines for the workflow system and executes them. This separation enables the addition of new workflow systems without changes to the API server's code. As a result, adding new workflows becomes straightforward, with the number of systems growing from just one at the beginning of the project to seven in the current version (Table 2). The flexibility of the "run.sh" also allows for specific adjustments for each workflow system, supporting pre- and post-execution processes, such as authentication, staging input files, and uploading results. Additionally, it is enabled to manage environment-specific requirements, including executing jobs on grid engines and handling file I/O with S3-like object storage. Once the system receives a workflow run request, it issues a universally unique identifier (UUID) and creates a directory named with the UUID, where the system stores all the necessary files. The workflow definition files, intermediate and final outputs, and the other metadata are stored in that directory. This per-run directory can act as a bundle of provenance for the workflow run (Figure 4).*

**3. The main flaw of the article in its current form is that the description of the experimental evaluation and the obtained results is left to external references. The authors describe three different experiments using three different workflow languages and datasets. Adding a detailed description of one of them directly in the article, together with the steps needed to reproduce it, would allow the reader to better understand the features Sapporo provided.**

In response to the comment by another reviewer (Dr. Justin M. Wozniak), we added the details of how users can specify the workflow condition in the Result section and the new figure Figure 8. We believe the addition can guide users to understand how they can run a workflow using our implementation.

**4. Also, there is no quantitative measure of achieved results in the article. Some quantitative measures of the Sapporo complexity (e.g., the percentage of product-specific lines of code that users are still forced to write to specify workflow parameters or the average lines of code needed to add support for a new WMS) would enable a more**

***scientifically sound evaluation of the product.***

We agree with the reviewer's feedback. Indeed, quantitatively evaluating how much the adoption of our system reduces the barrier to using different workflow management systems is challenging, which made this article posted as a software tool article. Methods such as user surveys could be considered for evaluation, but attempting to familiarize participants with multiple workflow systems without the assistance of our system or having learners of one language use a system in another language would not be practical. Demonstrating quantitatively that "using workflow systems is inherently more productive than executing workflows built with shell scripts on job queuing systems via the command line" poses a challenge for the entire developer community involved in developing workflow systems. Considering the challenges in the evaluation, with this paper we believe that providing the option of not only multiple different workflow management systems but also a system that can be used across them is our main contribution to the community.

**Competing Interests:** NA

Reviewer Report 26 July 2023

<https://doi.org/10.5256/f1000research.134975.r185842>

© 2023 Wozniak J. This is an open access peer review report distributed under the terms of the [Creative Commons Attribution License](#), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.



**Justin M. Wozniak** 

Data Science and Learning, Argonne National Laboratory, Lemont, IL, USA

This article presents a new abstraction layer (Sapporo) over existing workflow systems to support portability and interoperability. The paper is oriented around a bioinformatics workload. The paper is primarily focused on how Sapporo operates as a web service and runs underlying workflows using the other existing systems.

The paper provides a pretty good high-level state-of-the-art in the workflow ecosystem and the bioinformatics use case.

The paper spends most of its space describing the abstraction over workflow systems and figures that illustrate the corresponding architecture. It does not contain a deep dive into any challenges regarding workflow system interoperability.

From a bioinformatics perspective, the description of support for the application workload is very high-level. There is no deep dive into what is really required to make this workload work. There are results posted for the run that are linked on the Internet, but they are not summarized in the text.

The architecture figures are very spacious and do not provide much technical insight.

There is an illustration of the web form used to run Sapporo, but it seems oversimplified and does not convey what the user would be faced with in a real-world problem.

I downloaded the Sapporo source zip via the links in the paper. The source tree was not very revealing and I am not set up to run a new web service. The README pointed me to web-based docs that seemed quite good and included detailed installation notes.

Overall, this is a good high-level introduction to Sapporo, but does not offer detailed technical insights into the problem space or the Sapporo solution. More details are necessary about these topics to provide insight to the community and allow for a better evaluation of the Sapporo contribution.

**Is the rationale for developing the new software tool clearly explained?**

Yes

**Is the description of the software tool technically sound?**

Partly

**Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?**

No

**Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?**

No

**Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?**

Partly

**Competing Interests:** No competing interests were disclosed.

**Reviewer Expertise:** workflows, hpc, distributed computing

**I confirm that I have read this submission and believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.**

Author Response 13 Jun 2024

**Tazro Ohta**

We deeply thank you for taking the time to review our manuscript.

**1. The paper spends most of its space describing the abstraction over workflow systems and figures that illustrate the corresponding architecture. It does not contain a deep dive**

***into any challenges regarding workflow system interoperability.***

We agree with your assessment. We added the text below in the Background section to emphasize the challenge in making the existing workflow systems:

*> Workflow systems have different language syntaxes and engines, each designed for specific purposes. For instance, Nextflow aims to boost developer productivity and scalability, while Snakemake focuses on flexibility and simplicity, using Python as its base. In contrast, the Common Workflow Language (CWL) project aims to promote interoperability by creating a standardized syntax that various workflow engines can understand. However, workflows written in different languages cannot be easily converted into each other automatically. The most popular workflow systems used in bioinformatics, such as CWL, WDL, Nextflow, and Snakemake, take a workflow definition and input parameters to produce output result files, while there are differences between these workflow systems in command-line options, workflow description syntax, methods for specifying inputs, and how expected output files are defined.*

*> Creating a universal language converter isn't practical because some languages lack the necessary syntax parsers, or contain features that are not commonly found in other workflow engines (e.g. JavaScript evaluation as in CWL, loops in workflows or cyclic workflows instead of DAG-based systems). To bridge the gap between different workflow systems, we need a standardized way to specify workflows, input parameters, and expected outputs. Additionally, a system that supports various engines and selects the appropriate one for a given workflow is essential for smooth interoperability.*

***2. From a bioinformatics perspective, the description of support for the application workload is very high-level. There is no deep dive into what is really required to make this workload work. There are results posted for the run that are linked on the Internet, but they are not summarized in the text.***

We agree again with your assessment. To clarify the user's procedure for performing the analysis, we modified the paragraph in the Result section as follows:

*> To evaluate the practical applicability and robustness of Sapporo, we executed public workflows frequently used by researchers. Specifically, we chose the Mitochondrial Short Variant Discovery workflow from the GATK best practices (written in WDL), the RNA-seq workflow from the nf-core repository (written in Nextflow), and a Germline Short Variant Discovery workflow for processing whole-genome sequencing data from the Japanese Genotype-phenotype Archive (written in CWL). Users access Sapporo's endpoint specifying the input parameters following the WES specification. The required parameters are workflow\_url, workflow\_type, workflow\_type\_version, and workflow\_params. The workflow\_url argument specifies the location of the workflow definition file (e.g. CWL file) to be executed, typically hosted on a remote server, enabling the API to access and utilize the workflow's instructions. The workflow\_params argument points to a JSON file containing input parameters essential for the workflow execution, facilitating customization and adaptation of the workflow's behavior. The arguments workflow\_type and workflow\_type\_version arguments indicate the type and version of the workflow language being employed, ensuring compatibility and proper interpretation of the workflow instructions by engines supported inside Sapporo. Additionally, the workflow\_engine\_name argument specifies the execution engine to be used, while the default engine for the given workflow language is assigned when it is not*

*specified. Lastly, another optional argument workflow\_engine\_parameters argument allows for the specification of additional parameters tailored to the execution engine, providing fine-grained control over the execution environment and behavior of the workflow engine. We published the detailed description of the test procedures for these workflows on GitHub, and the results of the test runs on Zenodo.*

We also added a detailed view of user's procedure to run a workflow with Sapporo as Figure 8.

### **3. The architecture figures are very spacious and do not provide much technical insight.**

The architecture figures presented in the paper aim to illustrate the concept of our approach, which is not a monolithic software but rather divided into multiple layers. The figure aims to focus on the concept itself, because the technologies employed for implementation may change in the future. While we implemented the Sapporo-service providing the Web API in Python Flask and the Web UI accessing the API in Vue.js to enhance our development efficiency, we do not claim technical superiority over them here. As the software stacks evolve, we acknowledge the possibility of changing technology choices in the future. The idea of decomposing components into functionalities, as exemplified by microservices architecture, is common in today's software engineering, and we do not consider it novel. However, in the field of bioinformatics where existing workflow systems tend to be monolithic, easily extensible and layered systems are not as prevalent. Thus, we argue in the paper that such a design is beneficial for solving the problem we raise. Nonetheless, from the series of comments received from the reviewer, we recognized that the main message of the paper might not have been as clear as we intended. We believe the paragraphs added in Background and Discussion according to your comments may help readers to understand the aim of our projects.

### **4. There is an illustration of the web form used to run Sapporo, but it seems oversimplified and does not convey what the user would be faced with in a real-world problem.**

We understand the reviewer's concern about the oversimplified depiction of the workflow in Figure 7, resulting in a very basic UI representation that may differ from what users may encounter when using Sapporo-web in real-world scenarios. The intention behind this illustration was to demonstrate the automatic rendering of the UI based on the input in the workflow definition file. We do not assume or claim here that highly complex workflows with complicated input parameter sets can be executed seamlessly through the generated Web UI. We added the following sentences in the Discussion section to claim the relationship between the user interface and the improvement of workflow usability:

*> As the complexity of the workflow increases, so does the difficulty of executing it. Sapporo's primary goal is to reduce the time and learning costs associated with deployment, parameterization, and execution due to changes in workflow languages or systems, not to reduce the inherent complexity of given workflows. The use of complex workflows requires complex input parameter specifications due to the intricacies of their internal processes, and using them without understanding them would not reflect scientific integrity in data analysis. Of course, some costs*



can be mitigated through UI enhancements, such as optimizing sets of multiple input parameters or streamlining iterations for numerous input files. However, it's not practical for Sapporo-web's default UI to cover all these scenarios, as web UIs are not one-size-fits-all. There is potential to solve this problem by semi-automatically generating a UI for each workflow, a concept we're exploring in another project. However, even in this scenario, the advantage of splitting our UI into standardized APIs remains apparent.

**5. Overall, this is a good high-level introduction to Sapporo, but does not offer detailed technical insights into the problem space or the Sapporo solution. More details are necessary about these topics to provide insight to the community and allow for a better evaluation of the Sapporo contribution.**

In response to your comment; we believe that your previous points stressed this same point, and the content added there addresses this same issue. As outlined in our response to your first comment, we have added descriptions in the updated manuscript about the issues Sapporo aims to address. Additionally, we have incorporated the following passage into the Discussion section:

*> To support workflow developers and researchers conducting data analysis, multiple different workflow management systems have been developed. These systems enhance productivity and reproducibility in data analysis, enabling more effective science. However, the proliferation of multiple systems has revealed inefficiencies, leading to fragmentation within developer and user communities. While it is crucial to effectively leverage the assets of each system and community, it is not practical to provide the methods for syntax conversion between workflow systems and extending execution engines. Therefore, Sapporo aims to absorb differences between systems by wrapping multiple systems. Specifically, we provide an API that rewrites workflow definitions and runtime parameters into the command lines of each system based on the type of workflow definition received, enabling the execution of different workflows using the same client. Inside the API server, we use Docker containers to ensure the usability of different workflow engines. The use of containers also ensures future additions of workflow engines while maintaining the portability of the API server. The Web API adheres to the internationally defined GA4GH WES standard, ensuring interoperability with other GA4GH WES implementations. By developing and releasing Sapporo Web as an example of a GA4GH WES client, we demonstrate the readiness of our developed API for research use.*

**Competing Interests:** NA

The benefits of publishing with F1000Research:

- Your article is published within days, with no editorial bias
- You can publish traditional articles, null/negative results, case reports, data notes and more
- The peer review process is transparent and collaborative
- Your article is indexed in PubMed after passing peer review
- Dedicated customer support at every stage

For pre-submission enquiries, contact [research@f1000.com](mailto:research@f1000.com)

**F1000Research**