SOFTWARE TOOL ARTICLE

Sapporo: A workflow execution service that encourages the reuse of workflows in various languages in bioinformatics
[version 1; peer review: awaiting peer review]

Hirotaka Suetake1, Tomoya Tanjo2, Manabu Ishii3, Bruno P. Kinoshita4, Takeshi Fujino5, Tsuyoshi Hachiya3, Yuichi Kodama1,2, Takatomo Fujisawa2, Osamu Ogasawara2, Atsushi Shimizu2, Masanori Arita2, Tsukasa Fukusato6, Takeo Igarashi1, Tazro Ohta7

1Department of Creative Informatics, Graduate School of Information Science and Technology, The University of Tokyo, Bunkyo, Tokyo, Japan
2Bioinformation and DDBJ Center, National Institute of Genetics, Mishima, Shizuoka, Japan
3Genome Analytics Japan Inc, Shinjuku, Tokyo, Japan
4Curii Corporation, Sommerville, MA, USA
5Department of Computational Biology and Medical Sciences, Graduate School of Frontier Sciences, The University of Tokyo, Bunkyo, Tokyo, Japan
6Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo, Bunkyo, Tokyo, Japan
7Database Center for Life Science, Joint Support-Center for Data Science Research, Research Organization of Information and Systems, Mishima, Shizuoka, Japan

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

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 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 Bioinformatics gateway.

**Corresponding author:** Tazro Ohta (t.ohta@dbcls.rois.ac.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:** © 2022 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 et al. Sapporo: A workflow execution service that encourages the reuse of workflows in various languages in bioinformatics [version 1; peer review: awaiting peer review] F1000Research 2022, 11:889 https://doi.org/10.12688/f1000research.122924.1

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

Modern experimental instruments that convert biological samples into digital data have lower costs and higher throughput than conventional ones. This has 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. DNA sequencing technology highlighted the data science aspect of biology, sparking the demand for computation in biology.

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.

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. 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. 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.

Open-source workflow systems help the research community work efficiently by reusing published workflows. 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. 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. 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. 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.

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. 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. 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.

The source code, test code, and documentation for Sapporo-service and Sapporo-web are available from GitHub and archived in Zenodo.

**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. 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 execution layer executes the workflow system according to the request. It has a simple shell script named “run.sh” as its easily extendable core component. The number of workflow systems increased from just one at the beginning of the project to seven in the current version (Table 2). 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.

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
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.

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. 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.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engine selector</td>
<td>Select engine from available implementations</td>
</tr>
<tr>
<td>Remote URL as attachment</td>
<td>Fetch remote file and attach to the run</td>
</tr>
<tr>
<td>Output downloader</td>
<td>Direct download of workflow results</td>
</tr>
<tr>
<td>Registered-only mode</td>
<td>Restrict workflows by allowed-list</td>
</tr>
<tr>
<td>Workflow parser</td>
<td>Return parsed workflow information</td>
</tr>
</tbody>
</table>
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. 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.22

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

---

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

<table>
<thead>
<tr>
<th>Engine</th>
<th>Supported language(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cwltool15</td>
<td>CWL</td>
</tr>
<tr>
<td>Nextflow9</td>
<td>Nextflow</td>
</tr>
<tr>
<td>Toil16</td>
<td>CWL</td>
</tr>
<tr>
<td>Cromwell17</td>
<td>WDL, CWL</td>
</tr>
<tr>
<td>Snakemake18</td>
<td>Snakemake</td>
</tr>
<tr>
<td>ep319</td>
<td>CWL</td>
</tr>
<tr>
<td>StreamFlow20</td>
<td>CWL</td>
</tr>
</tbody>
</table>

---

29109b85-7935-4e13-8773-9def402c7775
   └── run_request.json
   │   └── workflow_engine_params.txt
   │   └── exe
   │       └── <input_files>
   │           └── workflow_params.json
   │       └── outputs
   │           └── <output_files>
   │               └── output.json
   │               └── stdout.log
   │               └── stderr.log
   │               └── state.txt
   │               └── run.pid
   │               └── cmd.txt
   │               └── start_time.txt
   │               └── end_time.txt
   │               └── exit_code.txt

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.

---

15 Elixir-GA4GH WES, another WES implementation, to confirm Sapporo’s GA4GH WES specification compliance.22
16 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 workflow described in Common Workflow Language (CWL) generates a typed input form per parameter because CWL specifies input parameters with a structured text form.23 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.20

To evaluate the usability with real use cases, we used Sapporo to run the public workflows that researchers frequently use. We successfully executed the following three workflows: Mitochondrial short variant discovery from the GATK best practice workflows (language: WDL), RNA-Seq workflow from the nf-core repository (language: Nextflow), and a GATK best practice-compatible germline short variant discovery workflow, which is used to process whole-genome sequencing data of the Japanese Genotype-phenotype Archive (language: CWL).24 We published the description of the test procedures for these workflows on GitHub,25 and the results of the test runs on Zenodo.26–28

Discussion
In the big-data era in biology, the demand for efficient data processing will never stop increasing.29 There are countless painful tasks in data processing, and researchers have developed methods to solve each of them, resulting in many different workflow systems.30 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.31 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.

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.9,18 However, this flexibility in describing the procedures often makes the required input parameters unparsable by other applications. It means that users need to learn how to edit the parameters for each...
Figures 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.

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. 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**

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:


**Extended data**

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](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.](https://doi.org/10.5281/zenodo.6609570)
- 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](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.](https://doi.org/10.5281/zenodo.6462809)
- 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

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