Common Workflow Language Execution on the I8-Testbed

Thomas Dietrich, Sebastian Gallenmüller*, Manuel Simon*
*Chair of Network Architectures and Services
School of Computation, Information and Technology, Technical University of Munich, Germany
Email: thomas.dietrich@tum.de, gallenmu@net.in.tum.de, simonm@net.in.tum.de

***Abstract*—The Common Workflow Language (CWL) is an open standard to describe workflows in a portable way. Data analysis using workflows has increased significantly in science. Nevertheless, there has been no coordinated way to express them, resulting in multiple diverse workflow systems that are complicated to exchange. Exchange and verification are fundamental principles in research. Therefore, it is essential to share the workflows behind the findings in a standardised way so that the results are reproducible and can be confirmed. This article describes the necessary steps to install and execute CWL workflows on a testbed that uses the pos framework to manage nodes. For the execution of CWL, we chose an implementation called StreamFlow because it provides SSH Connectors. A CWL execution on this testbed utilises a combination of StreamFlow, the pos framework and bash scripts. This automation framework can execute new CWL workflows or existing ones from other researchers without a high expense. It enables data analysis with CWL and its verification. An example implementation is available at https://gitlab.lrz.de/netintum/teaching/iitm/repos/2023ss-bs/u838/-/tree/main/resources/experiment.**

***Index Terms*—experiment workflows, common workflow language, streamflow, testbed**

1. Introduction

Scientific experiments are capable of analysing data to acquire knowledge. These data analyses can consist of several steps following a fixed sequence to generate insights and meaningful output data from the input material. Computers carry out these workflows when dealing with large amounts of data or complex tasks. For this, scientists need applications that execute these workflows based on input data. Competing solutions exist from various companies that perform workflows in different ways. Baker and van Hemert's research [1] concluded that workflows become the dominant technology in describing scientific processes. As a result of this growth, there are over 300 different computational data analysis workflow systems compiled by Amstutz et al. [2].
Exchange with other researchers, mutual control, peer review and transparency are the fundamentals of scientific research. For this, workflows used in scientific methods must be traceable. However, this is complex, with multiple different systems. Repeating experiments with other systems requires a high expense and is impossible if two systems are incompatible. A new standard called the Common Workflow Language (CWL) was developed by Amstutz et al. [3] to ensure that academic principles can also be maintained when using workflows. A multi-faceted project created by Crusoe et al. [4] around the mere language strives for portably exchangeable workflows between systems with various environments, thus enabling scientists to reproduce data analyses for more transparency and better control over results. Due to these reasons, Leipzig [5] identified the CWL standard as a rising trend in his review of bioinformatic pipeline frameworks.
In addition, the CWL project provides tools to facilitate editing and viewing workflows, converting existing languages into CWL through converters and enabling execution through frameworks. The creators of CWL also provide a reference implementation called cwltool that supports local execution on Linux, Mac and Windows. Besides the reference solution, production-ready implementations with other features exist, such as the StreamFlow implementation by Colonnelli et al. [6] with various connectors. These connectors allow StreamFlow to connect to hybrid workflows using cloud computing and high-performance computers or multi-container environments like Docker containers by Merkel [7].
This paper describes the necessary steps to install and execute the CWL. It uses an implementation of StreamFlow with SSH Connectors to combine CWL workflow steps to nodes on the testbed of the Chair of Network Architectures and Services from the Technical University of Munich (I8), which Haden et al. introduced in [8]. In addition, bash scripts automate the execution of StreamFlow projects on the testbed of the I8 using the pos framework from Gallenmüller et al. [9].
The content of this paper is structured as follows. Sec. 2 briefly describes how to write a workflow in CWL, followed by Sec. 3, explaining how StreamFlow is structured and how to use it to execute the CWL. Sec. 4 uses this information to combine CWL and StreamFlow with bash scripts and the pos framework to create an experiment runnable on the testbed. Sec. 5 concludes the paper and provides an outlook on further contents of the CWL project.

2. Common Workflow Language

CWL is an open and free standard. It describes workflows in a human-readable way. CWL focuses on its community and aims to be interoperable, vendor-neutral and portable across different platforms. Reusable workflows and reproducible results can improve transparency in research. There are tools, libraries and editor plugins that already support CWL. Furthermore, you can parallelize

 doi: 10.2313/NET-2023-11-1_04

workflows with the scatter feature [4] to make them scalable. However, this paper will only deal with the basic functionalities of CWL because additions like tools or the scatter feature would exceed its scope.
CWL workflow files are written in YAML and have the ending `.cwl`. There are two essential class types: the `CommandLineTool` and the `Workflow` class. The `CommandLineTool` class is a wrapper for software tools executed with the command line. It stores the command, describes input and output and assigns them an id defined in the `Workflow` class [3].

```
1  cwlVersion: v1.2
2  class: CommandLineTool
3  baseCommand: echo
4  stdout: output.txt
5  inputs:
6    step_in:
7      type: string
8      inputBinding:
9        position: 1
10 outputs:
11   step_out:
12     type: stdout
```

Listing 1: echo.cwl

The functionality of the file above is to repeat an input message captured in a text file called `output.txt`. After specifying the `cwlVersion` and the `class` type, a `baseCommand` has to be declared. The expression in Line 4 of Listing 1 captures the messages from `stdout` in a file. The class assigns input and output names and types. The `inputBinding` describes the position of arguments after the command [3].

```
1  cwlVersion: v1.2
2  class: Workflow
3  inputs:
4    message: string
5  outputs:
6    out:
7      type: File
8      outputSource:
           echo/step_out
9  steps:
10   echo:
11     run: echo.cwl
12     in:
13       step_in: message
14     out: [step_out]
```

Listing 2: workflow.cwl

To define the workflow, the `Workflow` class uses the `CommandLineTool` classes. It specifies the input and output of the whole workflow and assigns intermediate products to steps. Inputs of workflows are typically stored in JSON or YAML files with the same name as the workflow file followed by a `"-job"` suffix. CWL is limited by the input given once the workflow has been started. Since it is not possible to add input later, care must be taken to add all the required data before starting the workflow to avoid later problems. The `workflow.cwl` file uses the `CommandLineTool` from Listing 1 to build an example workflow with one step. A complex workflow can be created, in a human-readable manner, by adding other work steps. CWL also supports additional requirements like using JavaScript with the `InlineJavaScriptRequirement` for added functionality. Installing the required packages for all extra functionalities in the setup is necessary. To evaluate, e. g. inline JavaScript expressions, `Node.js` has to be installed in addition to the basic packages, as shown in Section 4.2 [3].

3. StreamFlow

StreamFlow is a workflow manager capable of executing CWL workflows on different architectures. As shown in Figure 1, this workflow manager executes a StreamFlow file. This file is written in YAML and conventionally is called `streamflow.yml`. It consists of two parts. The first one is called "Model description files" and contains a YAML description of infrastructures that should execute the workflow steps, e. g., different servers or different nodes of a testbed.

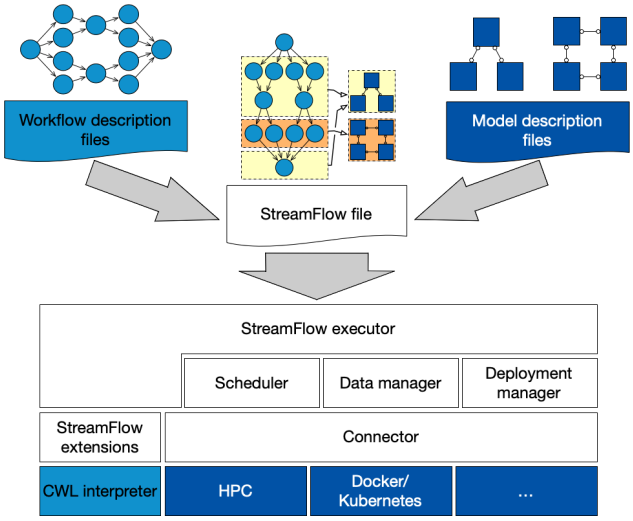

Figure 1: StreamFlow Model from [6]

The deployment diagrams in the upper right corner show that there can be multiple models with different structures within a single model description file. A model has a `type` section containing the chosen connector and a `config` field with connector-specific configuration parameters. The second part, called "Workflow description files", references a workflow file like `workflow.cwl` in Listing 2, or, as indicated by the activity diagram in the upper left corner of Figure 1, can be a complex workflow that runs in parallel steps. Input files mentioned in Section 2 and a `bindings` section are also part of these description files. The binding section describes which workflow step is deployed on which model. This section can, e. g. specify that the echo step in Listing 2, Line 10 should be executed on a specific node from the testbed mentioned in the introduction. This binding is visualised by the arrows in the illustration in the middle of Figure 1 that combine the deployment diagram and the activity diagram. As a whole, combined by the `bindings` section, these two description files result in a StreamFlow file. With this file, the StreamFlow executor can interpret CWL, schedule

tasks and deploy the workflow to connected environments while managing the data by itself [6].

This paper uses an `SSHConnector` to connect to different nodes. The names of the used nodes listed in an array called `nodes`, the username in a field `username`, and a path to the SSH key file on the nodes in `sshKey` are configuration parameters for this connector. As depicted in Figure 1, there also exist other connectors for high-performance computing (HPC) and container technologies. As mentioned in the introduction, there is, e. g. a `DockerConnector` that requires a docker image as his `config` or a `SlurmConnector` that is capable of connecting to an HPC facility orchestrated by the Slurm queue manager from Yoo et al. [10]. This `SlurmConnector` requires a hostname of an HPC facility and a username for the SSH connection to this facility. While container connectors would also be feasible on the I8-testbed, HPC connectors require an HPC facility [6].

4. Implementation on the Testbed

Once the workflow is described by CWL and the execution is managed with StreamFlow, a Bash script called `experiment.sh` is executed on the management node. It automatically orchestrates nodes from the testbed [8] using the pos framework, simultaneously installs the needed packages, prepares SSH Connections for the connectors mentioned at the end of Section 3, copies required files to the nodes and executes the StreamFlow file.

4.1. Orchestration of Nodes

The testbed consists of nodes orchestrated by a management node through the pos framework [9]. The framework can reserve selected nodes by creating calendar entries for them. It allocates the nodes for the experiment and boots them with specified images. Furthermore, pos can copy files from the management node to other nodes and remotely execute commands or executables on selected nodes. For the execution of StreamFlow, a master node is set up that uses a selectable amount of worker nodes, as shown in Figure 2. A Debian 11 (bullseye) image, e. g. is needed, since StreamFlow requires `python >= 3.8` [6].

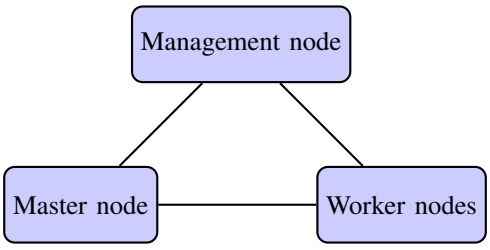

Figure 2: Testbed typology

4.2. Basic Nodes Setup

The `experiment.sh` script simultaneously executes a setup script called `setup.sh` on all selected nodes using the pos launch feature. The script displayed in Listing 3 installs the needed packages for CWL and the StreamFlow execution [9].

,

```
1 #!/bin/bash
2 apt-get update
3 apt-get install -y python3-pip
4 pip install streamflow
5 pip install --force-reinstall
      "cwltool==3.1.20220802125926"
6 curl -fsSL
      https://deb.nodesource.com/setup_19.x
      | bash - &&\
7 apt-get install -y nodejs
8
9 #add needed packages here
```

Listing 3: The setup.sh script

After updating, the script installs the Python package installer `pip` because it is used to install StreamFlow. Once `pip` is available, it installs StreamFlow. Installing StreamFlow downloads a cwltool version missing a file [11]. Therefore, it is necessary to reinstall the latest version, `cwltool==3.1.20220802125926`, without this bug. Users can add other packages required for the experiment to this list. Inline Javascript expressions, e. g. , can be used in the CWL. To evaluate these expressions, NodeSource, a distribution of `Node.js` for Debian [12], must be installed. Once all packages are installed, the master and worker nodes are processed differently. Only the master node receives the `streamflow` folder in the `experiment` directory displayed in Figure 3.

4.3. Preparation of SSH Connections

From now on, it would be possible to execute CWL files on nodes locally with the `cwltool` mentioned in Section 4.2. So it would also be possible to build further preparation steps as a workflow. However, since this would generate a lot of overhead and exceed the concepts presented in section 2, bash scripts also perform further preparations. By default, the management node can communicate with other nodes. To achieve connectivity - as shown in Figure 2 - the master node should also be able to connect to the worker nodes. Therefore, the experiment script prepares the master and worker nodes for SSH connections. The `experiment` directory contains a pre-generated SSH key pair with a private key called `worker` and a public key called `worker.pub`, as shown in Figure 3. The framework script `experiment.sh` executed on the management node remotely copies the public SSH key to all worker nodes with the pos copy feature. Then a script called `keyexchange.sh` is remotely executed on all workers with the pos remote command execution feature from the management node. This setup script appends the copied public key to the `authorized_keys` in the `.ssh` folders to approve the new SSH connection. The `streamflow` folder copied to the master node, as described

in Section 4.2, contains the private key `worker`. To copy the private key, `pos` needs read permission on this file. Therefore the private key has reading permissions for everyone (`644`). Since this would be a security gap, the main script repeals the read permissions by executing the `safeKey.sh` script remotely on the master node. As a result, the private key has the default permissions `600`. Besides the private key, the master node needs all worker nodes mentioned in his `known_hosts`. The framework script uses the command `ssh-keyscan` to gather all public keys from the worker nodes in a manually created `known_hosts file`. This file is remotely stored on the master node by the management node and replaces the existing `known_hosts` file to verify all worker nodes to the master. After this preparation, the master node can connect to all worker nodes through an SSH connection to delegate work.

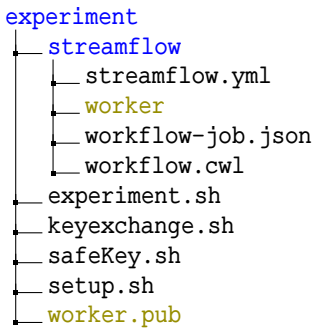

Figure 3: The experiment directory

4.4. Execution of the Experiment

The directory structure of a StreamFlow experiment could be set up as depicted in Figure 3. The `experiment` directory contains a `streamflow` subfolder that is copied to the master node from the management node. It contains the workflow files from Section 2, the StreamFlow file from Section 3 and the private SSH key. Section 4.3 explained this key with the public SSH key, the key-scripts `safeKey.sh` and `keyexchange.sh`. The main script `experiment.sh` explained in Section 4 and the `setup.sh` script shown in Listing 3 complete the experiment.
An example implementation can be downloaded from a GitLab Repository [13]. To execute the workflow, the `experiment` directory in Figure 3 can be copied to the management node of the testbed using the secure copy command `scp`. The main bash script that starts the whole experiment, "`bash experiment/experiment.sh`", is executed from the parent directory of `experiment` as the working directory. It takes a master node as the first argument and at least one worker node as the following arguments. For the workflow to function, the workers listed in the `nodes` array mentioned in Section 3 have to be passed as parameters after the freely selectable master node. When using this framework with larger projects, the hardcoded waiting time for the preparation to finish in `experimant.sh` has to be adjusted. Through the freely adjustable number of worker nodes, the script can scale with larger projects until all nodes from the testbed work to capacity if the workflows are built in a parallelisable way, e. g. , by using the scatter feature mentioned in Section 2. Additional tools required for workflow steps, can be added to the `setup.sh` script, as shown in Section 4.2. Consequently, this proof of concept can be used for further workflow experiments.

5. Conclusion

Instead of using proprietary workflow systems, complex data analysis flows can be described in the CWL standard. The StreamFlow manager can execute workflow descriptions in CWL. This manager interprets CWL and uses SSHConnectors to connect the master nodes with workers. In addition, it independently schedules deployment to workers and accomplishes data management. To automatically run the experiment on the I8-Testbed, bash scripts use features from the pos framework to orchestrate nodes, set them up, prepare SSH connections and execute the StreamFlow file. This way, data analysis workflows are portable and can be exchanged with other researchers to reuse the workflows and reproduce results. The exchangeable and verifiable CWL standard improves two necessary principles of modern research using automated data analysis and enhancing the transparency of scientific findings.
This paper is a proof of concept for a testbed framework of the CWL standard and does not cover the whole standard. There are further undiscussed CWL topics regarding CWL because it would have been too much for the scope of this paper. CWL offers, e. g. , a scatter feature to parallelise workflows to improve their scalability.

References

[1] A. Barker and J. van Hemert, "Scientific workflow: A survey and research directions," in *Parallel Processing and Applied Mathematics*, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewski, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 746–753.

[2] P. Amstutz, M. Mikheev, M. R. Crusoe, N. Tijanić, and S. Lampa, "Existing workflow systems," 2022, updated 2023-03-10, accessed 2023-03-28]. [Online]. Available: https://s.apache.org/existing-workflow-systems

[3] P. Amstutz, M. R. Crusoe, N. Tijanić, B. Chapman, J. Chilton, M. Heuer, A. Kartashov, D. Leehr, H. Ménager, M. Nedeljkovich, M. Scales, S. Soiland-Reyes, and L. Stojanovic, "Common Workflow Language, v1.0," 7 2016. [Online]. Available: https://figshare.com/articles/dataset/Common_Workflow_Language_draft_3/3115156

[4] M. R. Crusoe, S. Abeln, A. Iosup, P. Amstutz, J. Chilton, N. Tijanić, H. Ménager, S. Soiland-Reyes, B. Gavrilović, C. Goble, and T. C. Community, "Methods included: Standardizing computational reuse and portability with the common workflow language," *Commun. ACM*, vol. 65, no. 6, p. 54–63, may 2022.

[5] J. Leipzig, "A review of bioinformatic pipeline frameworks," *Briefings in bioinformatics*, vol. 18, no. 3, pp. 530–536, 2017.

[6] I. Colonnelli, B. Cantalupo, I. Merelli, and M. Aldinucci, "Streamflow: Cross-breeding cloud with hpc," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 1723–1737, 2021.

[7] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[8] M. Haden, "I8-testbed: Introduction," in *Proceedings of the Seminar Innovative Internet Technologies and Mobile Communications (IITM), Summer Semester 2020*, ser. Network Architectures and Services (NET), G. Carle, S. Günther, and B. Jaeger, Eds., vol. NET-2020-11-1. Munich, Germany: Chair of Network Architectures and Services, Department of Computer Science, Technical University of Munich, Nov. 2020, pp. 61–65.

[9] S. Gallenmüller, D. Scholz, H. Stubbe, and G. Carle, "The pos framework: A methodology and toolchain for reproducible network experiments," in *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 259–266.

[10] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60.

[11] Updated 2023-03-10, accessed 2023-03-28]. [Online]. Available: https://github.com/alpha-unito/streamflow/issues/15

[12] Updated 2023-03-10, accessed 2023-03-28]. [Online]. Available: https://github.com/nodesource/distributions

[13] T. Dietrich, accessed 2023-03-28]. [Online]. Available: https://gitlab.lrz.de/netintum/teaching/iitm/repos/2023ss-bs/u838.git