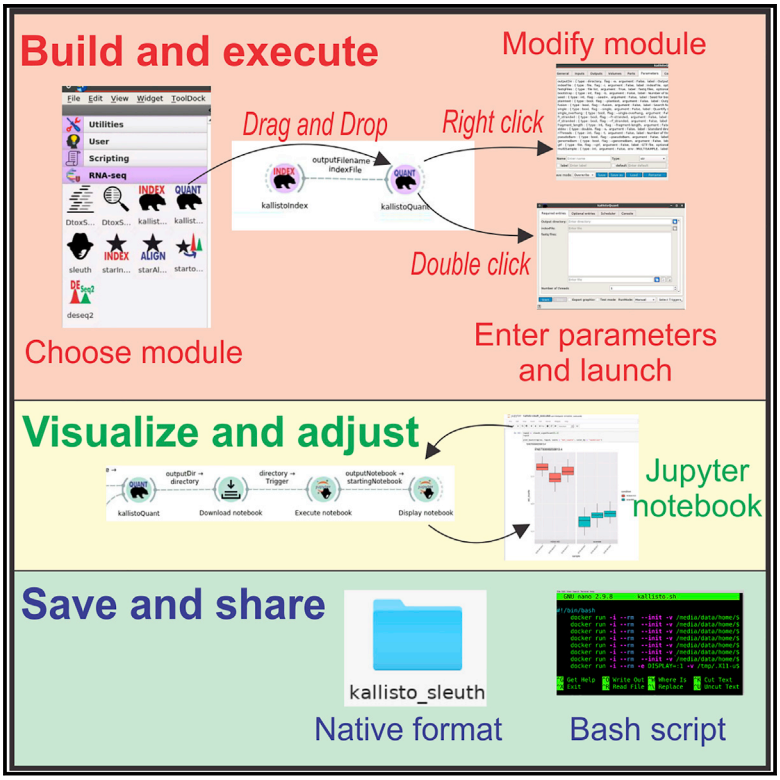
Brief Report

Cell Systems

Building Containerized Workflows Using the BioDepot-Workflow-Builder

Graphical Abstract



Authors

Ling-Hong Hung, Jiaming Hu, Trevor Meiss, ..., Yuguang Xiong, Eric Sobie, Ka Yee Yeung

Correspondence

kayee@uw.edu

In Brief

The BioDepot-workflow-builder (Bwb) is a graphical tool for creating and executing bioinformatics workflows. Icons representing commands executed inside Docker containers are connected to construct analytical pipelines. Bwb provides forms for parameter entry and for customization of workflow modules. Workflows are saved using Bwb's native format or exported as shell script.

Highlights

- An open-source graphical tool for constructing and executing bioinformatics workflows
- Uses Docker containers to ensure reproducibility and facilitate workflow installation
- Interactive graphical modules such as Jupyter and Cytoscape are supported
- Workflows can be saved in Bwb's native format or exported as portable shell scripts

Hung et al., 2019, Cell Systems 9, 508–514
November 27, 2019
https://doi.org/10.1016/j.cels.2019.08.007

CellPress

OPEN ACCESS
CellPress

Cell Systems
Brief Report

Building Containerized Workflows Using the BioDepot-Workflow-Builder

Ling-Hong Hung,[1] Jiaming Hu,[1] Trevor Meiss,[1] Alyssa Ingersoll,[1] Wes Lloyd,[1] Daniel Kristiyanto,[1] Yuguang Xiong,[2] Eric Sobie,[2] and Ka Yee Yeung[1,3,*]
[1]School of Engineering and Technology, University of Washington, Tacoma, WA 98402, USA
[2]Icahn School of Medicine at Mount Sinai, 1468 Madison Ave, New York, NY 10029, USA
[3]Lead Contact
*Correspondence: kayee@uw.edu
https://doi.org/10.1016/j.cels.2019.08.007

SUMMARY

We present the BioDepot-workflow-builder (Bwb), a software tool that allows users to create and execute reproducible bioinformatics workflows using a drag-and-drop interface. Graphical widgets represent Docker containers executing a modular task. Widgets are linked graphically to build bioinformatics workflows that can be reproducibly deployed across different local and cloud platforms. Each widget contains a form-based user interface to facilitate parameter entry and a console to display intermediate results. Bwb provides tools for rapid customization of widgets, containers, and workflows. Saved workflows can be shared using Bwb's native format or exported as shell scripts.

INTRODUCTION

One of the key challenges for biomedical science is the rapidly increasing number and complexity of analytical methods. Reproducing the results of a bioinformatics workflow can be challenging given the number of components, each having its own set of parameters, dependencies, supporting files, and installation requirements. We present the BioDepot-workflow-builder (Bwb), a graphical utility that builds and executes workflows using reproducible Docker containers as modules. Bwb draws workflows as graphical nodes (widgets) that represent executable modules. The nodes are connected by links to form a directed acyclic graph indicating the order of execution and flow of data. Users drag and drop widgets onto the screen and connect them to construct a workflow, which can be executed locally, saved, or exported as a shell script.

Reproducible Execution and Customization of Workflows for Non-programmers

The Bwb graphical user interface (GUI), shown in Figure 1, is designed for non-programmers who want to (1) reproducibly apply a workflow to data without worrying about installation of tools in the workflow, (2) adjust parameters and interact with graphical output from modules for visualization, and (3) incorporate new modules or tools into a workflow.

Bwb constructs workflows out of Docker containers, enhancing reproducibility and portability by encapsulating the operating system and software dependencies with the software. Any publicly available Docker containers, such as those from Dockstore (O'Connor et al., 2017), can be used in Bwb. Docker containers are automatically downloaded if they do not exist, essentially automatically installing the workflow. Bwb itself is a container hosted on our DockerHub repository that is installed with a Docker command. Upon launching Bwb, users also choose one or more host directories to be mounted to the Bwb file system. The user can then use the GUI to navigate and choose host files to be used in the workflows. Bwb automatically handles the mapping of path names, allowing the workflow to be easily applied to local data. The Bwb GUI allows a user to install, customize, and reproducibly apply a workflow to their data with minimum effort.

Most Docker workflow tools are currently designed to be used for large numbers of analyses in a non-interactive batch mode. Users with fewer datasets and less experience may need to tweak parameters and interactively monitor and assess workflows. In Bwb, double clicking on the widget reveals a series of forms, boxes, buttons, and checkboxes for entering flags, parameters and filenames, and a console for monitoring intermediate results. Furthermore, Bwb supports modules that use their own GUI, which include tools for quality assessment and visualization such as Cytoscape (Lau et al., 2017) and Gnumeric (the Linux version of Excel). Jupyter notebooks (Kluyver et al., 2016) can also be used to interactively execute, visualize, and document part of the pipeline while remaining within the Bwb sandbox. This functionality is achieved using methodologies derived from our GUIdock-X11 (Hung et al., 2016) and GUIdock-VNC (Mittal et al., 2017) to directly export graphics created by a module to the browser window.

Finally, one of the purposes of the Bwb GUI is to make it possible for non-programmers to explore new software to improve their pipelines and share their optimal workflows with others. With Bwb, changing a module is as simple as downloading a new workflow, dragging the new module onto the canvas, and adjusting the input and output connections and parameters. Workflows are organized in a single directory with sub-directories of widgets consisting of JavaScript Object Notation (JSON), Extensible Markup Language (XML), and Portable Network Graphics (png) files that can be easily shared. Alternatively, a workflow can be exported as a shell script of Docker commands that can be executed outside of Bwb or integrated into other scripts.

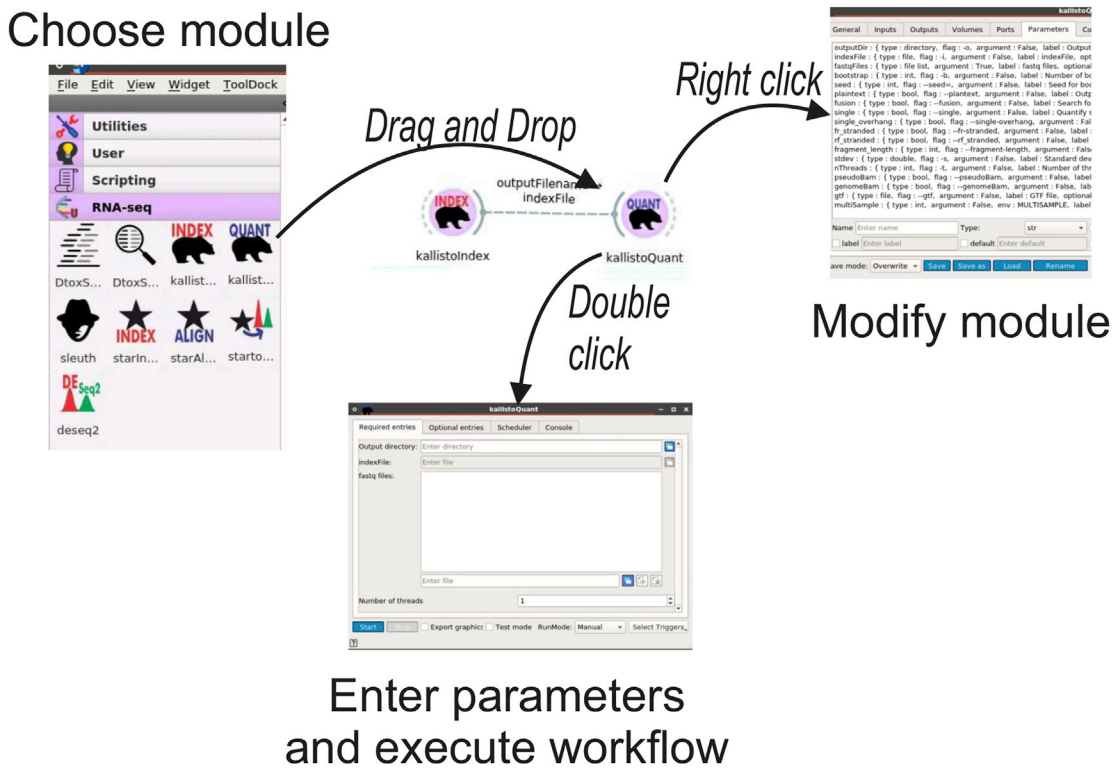


Figure 1. Bwb Interface

The basic elements of the Bwb interface are shown. When the Bwb application is launched and the user connects to the application using a browser, a window consisting of a canvas and a Tool Dock appears. Modules (widgets) are dragged from the Tool Dock and dropped onto the canvas and linked together to form a workflow, in this case, the kallisto-sleuth (Pimentel et al., 2017) workflow. Right-clicking on a module brings up a set of forms that allow the user to edit the properties of the widget such as the parameters to be queried and the command(s) to be executed. Double clicking allows the user to enter parameter values and execute the workflow.

Tools to Facilitate Integration of New Modules and Workflows into Bwb

Bwb is inclusive for users with different skill sets: the Bwb interface is user friendly for non-programmers and Bwb's tools are designed for bioinformaticians by (1) minimizing or eliminating the effort needed to create or modify a widget, (2) facilitating the construction of containers, and (3) easing the integration of existing scripts with Bwb.

Modifying the widget GUI is accomplished by right-clicking on the widget and filling out a series of forms specifying the inputs and outputs, ports that need to be opened, and the parameters and their types. Bwb automatically determines the GUI element to display from the data type that needs to be queried. For example, a parameter of type "file" will result in Bwb drawing a box to enter the filename and a button the user can press to access a graphical file explorer to find and choose the file. The forms also ask for the container to be used and command to be executed. Often, the container will already be available from public repositories. If a developer does need to build a custom container, we provide a tool called BiocImageBuilder (Almugbel et al., 2018) to facilitate this process for containers that use R and Bioconductor. Bwb additionally supports binding Dockerfiles (text recipes specifying containers) to widgets, allowing for inclusion of tools to modify and create containers directly from existing widgets.

Bwb also provides widgets for Bash, Java, Python, Perl, R scripts, and Jupyter notebooks with support for Bioconductor. These enable existing scripts to be integrated into Bwb workflows. An example video (see Additional Resources) demonstrates how to insert a Python script into a kallisto (Bray et al., 2016) workflow for RNA sequencing (RNA-seq) data (see Figure 2). Finally, Bwb supports exporting workflows as Bash scripts that can be incorporated into any scripted workflows.

Related Work

Container repositories such as Dockstore, BioBoxes (Belmann et al., 2015), BioShaDock (Moreews et al., 2015), and BioContainers (da Veiga Leprevost et al., 2017) have been developed for bioinformatics applications. Workflow descriptors such as Common Workflow Language (CWL) (Amstutz et al., 2016), Workflow Description Language (WDL), and YAML Ain't Markup Language (YAML) are also hosted with the containers to provide workflow scripts that can be reproducibly executed with a compatible scheduler on the cloud or local cluster.

The two most similar applications to Bwb are Seven Bridges, which powers the Cancer Genomics Cloud (CGC) (Lau et al., 2017), and Galaxy (Afgan et al., 2016), which is a web server. Seven Bridges is a commercial service using Amazon Web Services (AWS) or Google Cloud for bioinformatics analyses. Users drag multiple "apps" and parameters onto a canvas and

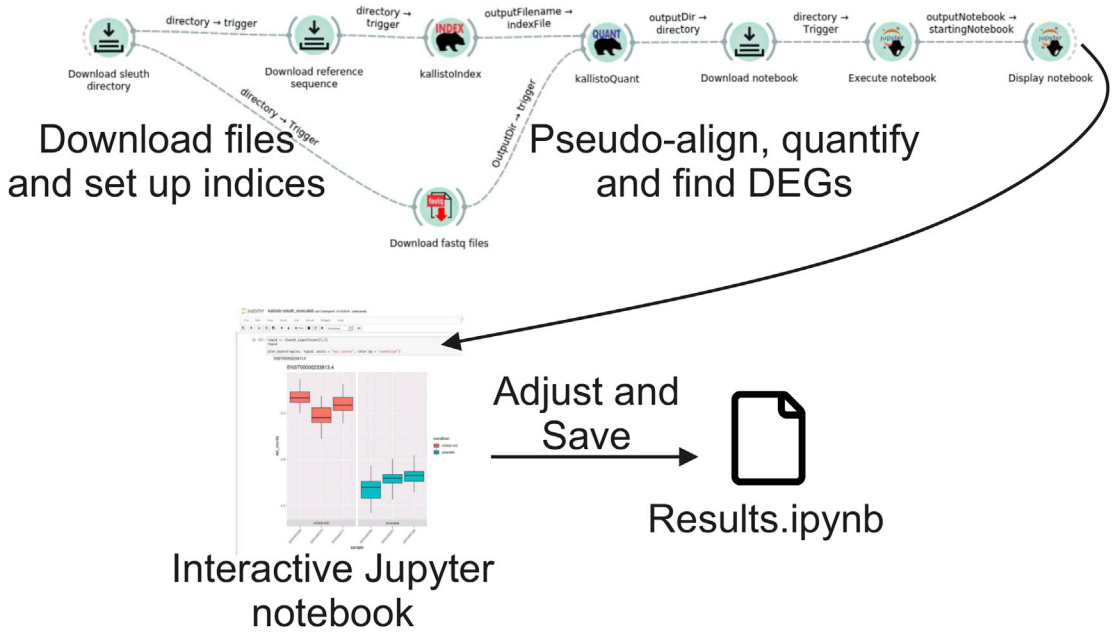


Figure 2. Kallisto-Sleuth Workflow

This workflow illustrates how Bwb is used to execute the kallisto-sleuth workflow. The leftmost widgets in the workflow download support files and set up the indices, whereas the later downstream widgets are responsible for pseudo-alignment and determination of differentially expressed genes (DEGs). When a widget completes its task, it sends signals to downstream connected widgets to trigger their execution. Execution can be started at any widget. For example, after downloading support files and generating indices once, the user can start at the pipeline at the kallisto pseudo-alignment if he or she wishes to analyze a new set of reads. In this workflow, the final widget displays a Firefox window inside Bwb to allow interaction with the Jupyter notebook containing the R script that calls sleuth. The user can re-run the code cells, adjust parameters, and visualize results without leaving the Bwb sandbox. When the user is satisfied, the notebook can be saved to the host computer.

connect them to define an executable workflow. The GUI is also available as an open-source tool, Rabix (Kaushik et al., 2017), for construction of CWL scripts that can be executed on CGC or another CWL compatible platform. In contrast, Bwb is open source and available as a Docker container that can be deployed on any local or cloud platform. Most importantly, Bwb supports interaction with workflow modules that have a GUI. Additionally, Bwb can produce shell scripts that can be modified to run on job schedulers such as Sun Grid Engine (Gentzsch, 2001) and Simple Linux Utility for Resource Management (SLURM) (Yoo et al., 2003). Instead of attaching parameter nodes to a widget, double clicking reveals a set of checkboxes, buttons, forms, and other graphical elements for entry of parameters for the widget.

Galaxy is a web server that provides a common web interface for users to create and execute workflows in a consistent hardware and software environment on a server or cluster. While most Galaxy workflows are not containerized, Galaxy can provide similar functionality to Bwb by using Bio-Docklets (Kim et al., 2017) to execute Docker workflows. However, importing tools and containers from non-Galaxy sources is not trivial and requires modifying a set of configuration files and scripts, whereas Bwb provides specific GUI tools for customizing existing workflows. Bwb can export workflows as Bash scripts of Docker commands that can be run without Bwb and easily incorporated into other scripts. In contrast, execution of Galaxy pipelines requires Galaxy.

Use Cases for Bwb

A key objective of Bwb is to enable researchers with varying levels of technical skill to deploy, reproducibly execute, and test alternative algorithms with confidence. In this manner, workflows and analytical results are made accessible to a large and varied set of users. Using Bwb, the entire biomedical community can quickly vet, implement, and share new technical advances in data analyses. Bwb widgets and workflows are designed to be easily customizable to encourage adoption by developers and bioinformaticians who support biomedical research. Bwb reduces the effort required of bioinformaticians, as they can auto-install pre-tested workflows and tweak the parameterizations using a consistent interface. There is no need to re-implement a pipeline after a system or package upgrade because all modules in Bwb workflows are containers, and all containers used by Bwb are tightly version controlled with explicit version tags, opposed to the "latest" tag. Bwb is also useful to developers wishing to add a graphical front end to command line executables. In this paper, we demonstrate the utility of Bwb with detailed descriptions of four case studies that highlight application use cases included in the Supplemental Information.

RESULTS AND DISCUSSION

We demonstrate the utility of Bwb using four case studies to highlight potential use cases for Bwb. The first use case in

Figure S1 illustrates the utility of Bwb for documenting and disseminating an existing workflow so that researchers can easily replicate and validate their results. This example demonstrates how one can create a completely executable flowchart that also exposes parameters and data to the user for workflow customization. The second and third use cases in Figures S2 and S3 illustrate the use of Bwb in well-established RNA-seq workflows using kallisto-sleuth (Bray et al., 2016; Pimentel et al., 2017) and STAR (Dobin et al., 2013), which consist of modules requiring different computing environments. The STAR workflow provides an example of Bwb running on a remote server. The fourth use case in Figure 2 demonstrates how Bwb's support for exporting GUIs can be utilized to add Jupyter notebooks to a workflow. The saved workflows and widgets from these four case studies are publicly available from GitHub and are included with the Bwb distribution (in the /workflows directory). After downloading the Docker image of Bwb, users can simply load these case studies into the canvas and execute these workflows with one click.

Case Study 1: Reproducibly Disseminating a Standard Operating Procedure Using Bwb

The first case study is taken from the NIH-funded Library of Integrated Network-Based Cellular Signatures (LINCS) program, which provides large-scale expression data from cell lines in response to genetic and drug perturbations (Keenan et al., 2018). The Drug Toxicity Signature Generation Center (DToxS), one of the LINCS data generation centers, studies the expression of human heart muscle cells under the influence of different drugs. As part of the effort to document all procedures and increase their reproducibility, laboratory and computational protocols for all experiments performed by DToxS are given in detailed standard operating procedures (SOPs) available on their website.

One of these SOPs describes a RNA-seq data analysis workflow consisting of two major steps—alignment using the Burrows-Wheeler Aligner (BWA) (Li and Durbin, 2009) and quantification of differential gene expression using the R package edgeR (Robinson et al., 2010). We use this DToxS RNA-seq pipeline as a test case for the creation and dissemination of reproducible workflows using Bwb. This RNA-seq workflow provides an example of an SOP represented as a Bwb workflow and is a completely reproducible, executable, and sharable representation of the pipeline.

Figure S1 depicts a demonstration of the DToxS RNA-seq workflow that can be run in a few minutes. This workflow consists of five widgets. The first widget (downloadURL) downloads the data and parameter files and sets up the directory structure. Note that the widget has been renamed to provide more clarity as to the role in the workflow. Renaming does not affect the program logic. The widget runs a custom Bash script that calls the Linux curl utility to download gzip or bzip2 to decompress the files. The Bash script also properly recognizes Google-Drive URLs. The Docker image uses the biodepot/bash-utils:alpine-3.7 container, which is a lightweight Alpine Linux system with Bash, wget, and curl. This widget connects the output to the trigger of the next widget so that the second widget will start once the download is complete.

The second widget DToxSAlignment, also based on Alpine Linux, provides a container that aggregates the BWA aligner with Python 2.7 for use in the original alignment script. The output is connected to the trigger of the third widget to signal it to start once the alignment step completes.

The third widget is another downloadURL widget. Normally, we would connect the DToxSAlignment widget to the DToxSAnalysis widget. However, executing this pipeline on complete files requires over 29 h while consuming 350 GB of disk space. For the demonstration, we use partial input files with fewer reads to enable the alignment step to complete in a few minutes. This leads to an error in the subsequent analysis steps, as there are too few counts. Consequently, for demonstration purposes, we download the complete results of the alignment using this widget. Here, again, the output is connected to the trigger of the fourth widget. The fourth widget analyzes the gene counts obtained from the alignment step to determine which genes are differentially expressed. The script runs under R and uses Bioconductor. The script and supporting packages are incorporated into an Ubuntu-based container image. In this case, when the widget completes, it not only transfers control to the last widget but also provides the name of the output file so it knows which file to open.

The final widget is Gnumeric, a fully functional open-source spreadsheet that duplicates many of the functions of Excel. It is inside an Ubuntu container and exports its own window and graphics. Note that the export graphics box in the widget options panel is checked to enable this functionality. Gnumeric displays the genes that are most likely to be differentially expressed (lowest p values).

To start the demonstration, the user can double click on the first downloadURL widget and the start button. Clicking on the console tab reveals the output. As each widget finishes and triggers the next widget, intermediate progress can be followed by double clicking on widgets and examining the console. After about 5 min, the final results automatically pop up in the gnumeric spreadsheet, which is shown below the workflow in Figure S1. To save the workflow as a Bash script, the user checks off the "test mode" check box in the downloadURL widget before pressing start. Instead of executing the Docker commands generated by the workflow, Bwb prints the Docker commands on the console of the widgets and optionally saves the set of commands as a script.

Case Study 2: Kallisto-Sleuth Pipeline

In this second example, we illustrate the use of Bwb with a widely used kallisto-sleuth RNA-seq data processing pipeline. The kallisto pipeline is shown in Figures 1, 2, S2, and S4. Kallisto takes the reads in fastq files and quantifies them using a rapid pseudo-alignment technique (Bray et al., 2016). Sleuth then processes the pseudocounts to find the genes that are differentially expressed (Pimentel et al., 2017). This workflow is adapted from the introductory sleuth walkthrough from the Pachter group.

The first widget again is a downloadURL widget that sets up the base directory and downloads two configuration files. One file defines which files are from the control and which files are from the treatment groups. The other file is used to map transcript names to gene names. The method of downloading configuration files rather than including them inside the

containers makes the logic clearer and the workflow more easily customizable.

This workflow diverges into two branches: one responsible for creating the indices and the other for downloading the fastq input files. The lower branch uses the widget created using the fastq-dump utility from the Sequence Read Archive (SRA) toolkit to download the fastq files from the SRA repository (Kodama et al., 2012). By default, the download is set to 10,000 spots, instead of the complete files, allowing the demonstration to complete in ~20 min. The upper branch of the workflow consists of a download widget that fetches the human reference sequence and passes it to the kallisto index widget to produce the indices. The two branches merge at the kallisto quant widget, which waits for both the indices to be made and the files to be downloaded before it can start. Sleuth then analyzes the counts created by kallisto quant and determines which genes are differentially expressed. The table of differentially expressed genes (DEGs) is then passed to the gnumeric widget, which pops up an interactive spreadsheet with the final results. The kallisto widgets are based on the executable compiled from the source in the GitHub repository. We could have used one widget for both the index and quant functions, but the workflow logic is clearer with two different widgets, and the kallisto quant widget required a wrapper script to handle multiple samples. Sleuth also uses a shell wrapper to pass parameters to an R script. The other widgets used containers that were straightforward installations of existing software, where the widget builder is leveraged to pass the flags and options to the user.

This workflow illustrates one of the key advantages of using Bwb—reproducible and automatic installation. Sleuth can be challenging to install, requiring different (undocumented) supporting libraries and packages depending on the version of sleuth, the operating system, and the version of R and whether R is being run with Jupyter. Using Bwb, the containers with all the necessary dependencies are automatically downloaded and run identically regardless of the user's host setup. Although the installation details are masked, they are accessible in the accompanying Dockerfile.

Case Study 3: STAR Pipeline on a Remote Server

The STAR-DESeq2 (Anders and Huber, 2010; Dobin et al., 2013) pipeline in Figure S3 is another well-established workflow for identifying DEGs from RNA-seq experiments. The structure of the pipeline is similar to the kallisto pipeline, and the construction of widgets is similar, with a wrapper around STAR quant to handle multiple samples. Again, the workflow is self-explanatory. With STAR, literally pages of flags and options are easily accessed, highlighting the utility of Bwb's use of a form-based interface to modify parameters. STAR requires a large amount of random-access memory (RAM) (a minimum of 32 GB is recommended), and the generation of indices and alignment steps are difficult to run on local hardware. This example was run on a remote firewalled server that is accessed securely through an ssh tunnel. The Gnumeric spreadsheet pops up and works exactly the same as the other examples that run locally on a laptop. However, even without a larger server, one can start the workflow at later stages if the necessary files are in place. The rest of the pipeline with parameters in place for documentation is executable if the user wishes to use a large local server or cloud server at some point.

Case Study 4: Using Jupyter Notebooks and Customizing a Workflow

In this example, we illustrate the integration of Jupyter notebooks into Bwb workflows by using a Jupyter notebook to analyze and visualize the data produced by kallisto. The notebook is a simplified version of the walkthrough from the Pachter lab (Bray et al., 2016; Pimentel et al., 2017) and uses sleuth. One widget runs the nbconvert function to execute the Jupyter script. It then connects a second widget, which opens the notebook automatically so that users can edit, interactively run the code, and visualize results using the native Jupyter GUI. This demonstration is illustrated in Figure 2, where the notebook displays a boxplot of the expression of the top DEG in the control and treatment samples. Using Bwb to incorporate Jupyter notebooks has many advantages. First, a Jupyter notebook can contain dynamic code for a single programming language, whereas a Bwb workflow can be constructed with many modules consisting of notebooks using different languages. Second, the use of containers ensures that the correct version of R is used with sleuth (3.4.4 and not 3.5.1.) and that the dependencies are pre-installed, saving long installation times when running notebooks.

Using Jupyter notebooks is an example method to customize Bwb workflows by allowing users to interactively modify the code and visualize results. We have included two videos in the Additional Resources involving this workflow. The first video illustrates the steps involved in loading and executing this workflow. The second video demonstrates how to create a new widget that adds a custom Python script to trim the fastq files before analyses and how to incorporate this new widget into the workflow.

Limitations of Bwb

This section details present limitations of the Bwb. First, iteration through a set of parameters (e.g., a set of fastq files) and multithreaded execution are currently implemented using wrapper scripts inside the container. Shell scripts exported by Bwb can then be modified for use in batch scripts for schedulers such as the Sun Grid Engine and SLURM. We are actively developing an automated scheduler system for the Bwb to support deployment and management of workflows to remote cloud and private clusters. Second, GUI support for the Bash scripts depends on having the Bwb application open on the desktop or having either native X11 or X11 emulation through GUIdock. Furthermore, the Bash scripts currently use the directory structure of the host machine that launches Bwb. We are working to develop support to facilitate mapping filenames for use across different machines. Currently, Bwb does not export to CWL or WDL used by other workflow execution engines such as Seven Bridges and the Broad Institute's Cromwell. However, the existing Bash script export tool and workflow XML file representation greatly facilitates the development of future tools for conversion to these workflow languages. Finally, we have done as much as possible to containerize the pipelines and Bwb itself to isolate it from the host platform. Presently, Bwb only requires the host to have an available installation of Docker and a web browser to facilitate connecting to the Bwb to render graphical displays. Of the major current web browsers (Chrome, Firefox, Safari, Opera, and Edge), only Edge for Microsoft Windows is not presently supported by the Bwb. This results from the way Edge handles local URLs. We include a workaround in

our manual for Edge that is sufficient for some versions of Windows, and Edge should work with a remote installation of Bwb.

STAR★METHODS

Detailed methods are provided in the online version of this paper and include the following:

- KEY RESOURCES TABLE
- LEAD CONTACT AND MATERIALS AVAILABILITY
- METHOD DETAILS
  - Bwb Windowing Environment
  - Drag-and-Drop User Interface: Widgets, Tool Dock and Canvas
  - Widget UI and Definition Windows
  - Workflow Storage and Execution
  - Customizing Workflows and Containers
  - Bwb Code Organization Summary
- DATA AND CODE AVAILABILITY
- ADDITIONAL RESOURCES

SUPPLEMENTAL INFORMATION

Supplemental Information can be found online at https://doi.org/10.1016/j.cels.2019.08.007.

ACKNOWLEDGMENTS

L.-H.H, W.L., E.S., and K.Y.Y. are supported by National Institutes of Health (NIH) grant R01GM126019. L.-H.H. and K.Y.Y. are also supported by NIH grant U54HL127624. Y.X. and E.S. are supported by NIH grant U54HG008098. We would also like to thank Microsoft Azure (L.-H.H.), Google Cloud Platform (K.Y.Y.), Amazon Web Services (L.-H.H., W.L., and K.Y.Y.) for computing resources. We would like to thank Jayant Keswani and students from TCSS 592 at the University of Washington Tacoma for contributing widgets and testing effort for earlier versions of the Bwb project. We would also like to acknowledge the Student High Performance Computing Club and the eScience Institute, both at the University of Washington, for providing technical assistance and computing resources to J.H.

AUTHOR CONTRIBUTIONS

L.-H.H is the primary developer for the Bwb project, wrote the existing code for the widgets and containers, adapted the OrangeML code for the drag-and-drop interface and toolbox, and implemented the Bwb container. K.Y.Y. coordinated the manuscript preparation. L.-H.H. and K.Y.Y. drafted the manuscript. L.-H.H. designed the framework of Bwb and the case studies. J.H. and T.M. contributed code to early Docker-py implementations of Bwb. J.H. designed and implemented the image building tool used by Bwb. L.-H.H., J.H., T.M., D.K., and A.I. contributed to testing and case studies of the project. L.-H.H., D.K., J.H., and K.Y.Y. contributed to the writing of the user manual. J.H., D.K., and L.-H.H. made the videos in the Additional Data files. L.-H.H. and W.L. provided technical guidance to all the students. Y.X., and E.A.S., developed the computational analysis pipeline at DToxS. All authors tested Bwb and read and approved the final manuscript.

DECLARATION OF INTERESTS

K.Y.Y. is also affiliated with the Department of Microbiology, University of Washington.

Received: December 29, 2018
Revised: May 21, 2019
Accepted: August 16, 2019
Published: September 11, 2019

REFERENCES

Afgan, E., Baker, D., van den Beek, M., Blankenberg, D., Bouvier, D., Čech, M., Chilton, J., Clements, D., Coraor, N., Eberhard, C., et al. (2016). The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. Nucleic Acids Res. *44*, W3–W10.

Almugbel, R., Hung, L.H., Hu, J., Almutairy, A., Ortogero, N., Tamta, Y., and Yeung, K.Y. (2018). Reproducible Bioconductor workflows using browser-based interactive notebooks and containers. J. Am. Med. Inform. Assoc. *25*, 4–12.

Amstutz, P., Crusoe, M.R., Tijanić, N., Chapman, B., Chilton, J., Heuer, M., Kartashov, A., Leehr, D., Ménager, H., Nedeljkovich, M., et al. (2016). Common Workflow Language, v1.0. Specification, Common Workflow Language working group.

Anders, S., and Huber, W. (2010). Differential expression analysis for sequence count data. Genome Biol. *11*, R106.

Belmann, P., Dröge, J., Bremges, A., McHardy, A.C., Sczyrba, A., and Barton, M.D. (2015). Bioboxes: standardised containers for interchangeable bioinformatics software. GigaScience *4*, 47.

Bray, N.L., Pimentel, H., Melsted, P., and Pachter, L. (2016). Near-optimal probabilistic RNA-seq quantification. Nat. Biotechnol. *34*, 525–527.

da Veiga Leprevost, F., Grüning, B.A., Alves Aflitos, S., Röst, H.L., Uszkoreit, J., Barsnes, H., Vaudel, M., Moreno, P., Gatto, L., Weber, J., et al. (2017). BioContainers: an open-source and community-driven framework for software standardization. Bioinformatics *33*, 2580–2582.

Dobin, A., Davis, C.A., Schlesinger, F., Drenkow, J., Zaleski, C., Jha, S., Batut, P., Chaisson, M., and Gingeras, T.R. (2013). STAR: ultrafast universal RNA-seq aligner. Bioinformatics *29*, 15–21.

Gentzsch, W. (2001). Sun grid engine: towards creating a compute power Grid. In Proceedings of the 1st International Symposium on Cluster Computing and the Grid (IEEE Computer Society), p. 35.

Hung, L.H., Kristiyanto, D., Lee, S.B., and Yeung, K.Y. (2016). GUIdock: using docker containers with a common graphics user interface to address the reproducibility of research. PLoS One *11*, e0152686.

Kaushik, G., Ivkovic, S., Simonovic, J., Tijanic, N., Davis-Dusenbery, B., and Kural, D. (2017). Rabix: an open-source workflow executor supporting recomputability and interoperability of workflow descriptions. Pac. Symp. Biocomput. *22*, 154–165.

Keenan, A.B., Jenkins, S.L., Jagodnik, K.M., Koplev, S., He, E., Torre, D., Wang, Z., Dohlman, A.B., Silverstein, M.C., Lachmann, A., et al. (2018). The Library of Integrated Network-Based Cellular Signatures NIH program: system-level cataloging of human cells response to perturbations. Cell Syst. *6*, 13–24.

Kim, B., Ali, T., Lijeron, C., Afgan, E., and Krampis, K. (2017). Bio-Docklets: virtualization containers for single-step execution of NGS pipelines. GigaScience *6*, 1–7.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., et al. (2016). Jupyter Notebooks – a publishing format for reproducible computational workflows. In Positioning and Power in Academic Publishing: Players, Agents and Agendas, F. Loizides and B. Schmidt, eds., pp. 87–90.

Kodama, Y., Shumway, M., and Leinonen, R.; International Nucleotide Sequence Database Collaboration (2012). The Sequence Read Archive: explosive growth of sequencing data. Nucleic Acids Res. *40*, D54–D56.

Lau, J.W., Lehnert, E., Sethi, A., Malhotra, R., Kaushik, G., Onder, Z., Groves-Kirkby, N., Mihajlovic, A., DiGiovanna, J., Srdic, M., et al. (2017). The cancer genomics cloud: collaborative, reproducible, and democratized-A new paradigm in large-scale computational research. Cancer Res. *77*, e3–e6.

Li, H., and Durbin, R. (2009). Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics *25*, 1754–1760.

Mittal, V., Hung, L.H., Keswani, J., Kristiyanto, D., Lee, S.B., and Yeung, K.Y. (2017). GUIdock-VNC: using a graphical desktop sharing system to provide a browser-based interface for containerized software. GigaScience *6*, 1–6.

Moreews, F., Sallou, O., Ménager, H., Le Bras, Y., Monjeaud, C., Blanchet, C., and Collin, O. (2015). BioShaDock: a community driven bioinformatics shared Docker-based tools registry. F1000Res. *4*, 1443.

O'Connor, B.D., Yuen, D., Chung, V., Duncan, A.G., Liu, X.K., Patricia, J., Paten, B., Stein, L., and Ferretti, V. (2017). The Dockstore: enabling modular, community-focused sharing of Docker-based genomics tools and workflows. F1000Res. *6*, 52.

Pimentel, H., Bray, N.L., Puente, S., Melsted, P., and Pachter, L. (2017). Differential analysis of RNA-seq incorporating quantification uncertainty. Nat. Methods *14*, 687–690.

Robinson, M.D., McCarthy, D.J., and Smyth, G.K. (2010). edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. Bioinformatics *26*, 139–140.

Yoo, A.B., Jette, M.A., and Grondona, M. (2003). SLURM: simple Linux utility for resource management. Paper presented at Job Scheduling Strategies for Parallel Processing (Springer), pp. 44–60.

STAR★METHODS

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Software and Algorithms		
Orange ML	University of Ljubljana	https://orange.biolab.si/
Docker	Docker, Inc	https://www.docker.com/; RRID: SCR_016445
Kallisto and sleuth walkthrough	Bray et al., 2016; Pimentel et al., 2017	https://github.com/pimentel/bears_iplant/blob/master/README.md; RRID: SCR_016582 (Kallisto) and SCR_016883 (sleuth)

LEAD CONTACT AND MATERIALS AVAILABILITY

This study did not generate new unique reagents.

Further information and requests for resources should be directed to and will be fulfilled by the Lead Contact, kayee@uw.edu.

METHOD DETAILS

Bwb Windowing Environment

The Bwb container launches a mini-webserver on the host that is accessed using the browser. The server uses fluxbox, a compact windows manager to provide a graphical user interface similar to Windows or the MacOS. Fluxbox provides full graphical support using X11 to render the graphics internally on the server. Bwb uses the GUIdock-X11 system to allow containerized applications (such as Jupyter or gnumeric) to export their graphics and GUI to the server's internal screen. The noVNC protocol is then used to transfer the internally rendered screen to the user's browser which draws the graphics in the browser window using HTML5.

Minimizing or closing the startup Bwb window reveals the background screen. Right clicking on the background brings up an application menu. For the basic Bwb container, there are three menu options, the Bwb app, a terminal to enter system commands, and the quit container option. Fluxbox provides four separate workspaces that are available which act as independent screens. Multiple instances of Bwb can be launched simultaneously. Windows can be resized, minimized, or maximized as in other windowing systems. Cut and paste is supported between windows inside the container. Support will be added to allow cut and paste with the host system.

Drag-and-Drop User Interface: Widgets, Tool Dock and Canvas

When Bwb is started, the Bwb application window pops up. On the left-hand side of the application window is a tool box (Tool Dock) with multiple tabs (drawers) containing different collections of widgets. Clicking on the tab expands the Tool Dock drawer to reveal the contents. Drawers are organized by function. Bwb comes with a set of 24 ready-to-use widgets. These are all linked to containers available on our BioDepot repository. Workflows constructed with these widgets require no installation as they will automatically download the necessary containers the first time they are run. Users can also create their own drawers. A new drawer is created whenever a workflow is loaded. Widgets can be added and removed, and drawers removed using the Tool Dock Editor available from the menu bar. To interact with a widget and include it in a workflow, the widget is dragged onto the canvas. Multiple copies of the same widget definition can exist in a workflow with different parameters. For example, the downloadURL widget is used twice in the kallisto demonstration workflows to download different files at different stages in the pipeline. Widgets on the canvas are then connected by dragging from the right side of the source widget to the left side of the sink widget. This will transfer the output of the source widget to the input of the sink widget during workflow execution.

The basic implementation of the Tool Dock and Canvas drag and drop UI is from OrangeML using the python PyQT5 Quicktime (QT) library. Orange workflows were connections of widgets that are given in the Tool Dock, which once loaded, do not usually change. Bwb extends the Tool Dock to have drawers for each workflow to support custom widgets. Bwb provides the Tool Dock editor to allow additional dynamic manipulations to the widgets in the Tool Dock. These additions are implemented using PyQT5 and Python.

Widget UI and Definition Windows

After being dragged to the Canvas, double clicking on the widget brings up a tabbed widget UI window, with tabs for entry of required, and optional parameter values. A third tab reveals the console which displays intermediate results from execution. At the bottom of the window, a series of buttons and drop-down menus are available to control the execution of the widget.

Right clicking on the widget and choosing "edit", brings up the tabbed widget definition window. Tabs are available for entry of values to define the UI window, choose port and volume mappings, inputs, outputs, the command to be executed, and the container to be used. In addition, the user can access tools to build and manage containers using the Docker tab.

Bwb widgets are loosely based on widgets found in Orange. The functionality of Orange widgets resides in a Python script associated with the widget. With Bwb, the functionality is standardized and resides in the command, parameter sets, and Docker container bound to the widget, and not in the Python script. Bwb stores these objects as JSON files. A small Python script is required to maintain compatibility with Orange routines such as the signal manager and widget manager that expect individual Python scripts. This stub Python script is auto-generated by Bwb's widget builder module. The implementation of the forms in the UI and definition window is accomplished using Python and PyQT5.

Workflow Storage and Execution

The workflow is stored in a single directory. This directory contains widgets specific to the workflow, the icon, and a Python script used to load the workflow into Bwb. An XML file saves the connections between the widgets, and all the parameter values or settings. This provides different information than the JSON files for widgets that do not store parameter values, but store information defining which parameters are queried, and what and how the widgets execute based on these parameters.

Bwb takes the values from the widgets forms and generates a Docker command (or set of commands when there are multiple commands) for each widget. When the widget begins execution, a QT QProcess is launched. The QProcess runs the Docker command, prints the output to the console, and checks for any signal sent by the user (through the 'stop' button) to abort the process. If the Docker command requires a container that is not present, Docker will automatically search through the public repositories and download the container if it is available (all containers used by widgets included with Bwb are available from the BioDepot repository). The QProcess then sends a signal when the Docker command has finished and a code to indicate whether the command successfully executed. If the Docker command was successful, the widget then sends out signals to downstream connected widgets. Upon receiving an input signal, the downstream widget checks that all necessary parameters are set, and if execution is also triggered (or is automatic once parameters are set), execution starts. In this manner, execution progresses iteratively across connected widgets in the workflow.

The Bwb signals are derived from Orange ML signals and are managed by the Orange signal manager. The XML connection files are also from OrangeML. The rest of the workflow storage and execution process is new to Bwb.

Customizing Workflows and Containers

Customization of workflows can occur at different levels. The simplest is at the level of parameters. We have discussed how Bwb facilitates this through the use of forms and the separation of parameters from the Docker containers and through the use of interactive widgets such as Jupyter.

Customization can also be accomplished by replacing a module with another, or by adding an additional module such as a script. This would be accomplished by inserting a new widget into a workflow. Most modules do not have identical inputs and outputs and require additional customization of widgets. We have previously discussed how Bwb facilitates widget construction through its form-based interface and provide a tutorial with a simple example.

Occasionally, the need will arise to customize Docker containers as well. Most bioinformatics scripts can be run within off-the-shelf Bash, Python, Perl, R or Java containers that we have provided with Bwb. However, it is not uncommon to require additional libraries, and while installation can be incorporated in runtime scripts, this practice is time consuming as library installation for utilities such as biocLite or install.packages is then executed each time the pipeline is run. To avoid this, we have provided our BiocImageBuilder tool (Almugbel et al., 2018) to enable users to specify packages required from Bioconductor and CRAN. BiocImageBuilder will build the container or provide a Dockerfile for the user to modify. In the future, we intend to expand the tool to include other common installation methodologies based on Conda and pip. Bwb also supplies files used to build its containers with its widgets in the Dockerfiles directory, to allow users to easily customize the images if they wish to do so without having to start from scratch. Our tutorial and video demonstrate how these tools can be used within a new container, and also for a widget with an existing script.

Bwb Code Organization Summary

The key components from OrangeML used by Bwb are the signal manager, Tool Dock, and Canvas routines. The signal manager queues and manages signals between widgets in workflows. The Tool Dock and Canvas modules handle the Tool Dock and the drag and drop interface. The OrangeML code has been forked and stored in a directory in the BioDepot-workflow-builder repository. Any modified Orange routines are kept in a separate directory.

The major Bwb modules are the BwBase, WidgetBuilder, ImageBuilder, DockerClient, ToolDockEdit classes and WorkflowTools and CreateWidget packages. The BwBase class handles the widget UI window. The WidgetBuilder class is responsible for the widget definition window. The ImageBuilder class runs the BiocImageBuilder tool for building Docker containers. The ToolDockEdit class adds tools to edit the Tool Dock. These are all subclasses of the original orange widget class, largely to ensure that they interact correctly with the other OrangeML routines. Finally, there are two new packages: workflowTools and createWidget. WorkflowTools handles loading and saving of Bwb workflows, and the createWidget package creates JSON files and python files for the widgets.

DATA AND CODE AVAILABILITY

The source code is publicly available on GitHub at https://github.com/BioDepot/BioDepot-workflow-builder.
The Docker containers used by Bwb are freely available at https://hub.docker.com/u/biodepot/

ADDITIONAL RESOURCES

An introductory tutorial on using Bwb is publicly available at https://www.youtube.com/watch?v=jtu-jCU2DU0.
A tutorial on adding a Python script to a Bwb workflow is publicly available at https://youtu.be/r_03_UG1mBg.
User Manual: https://github.com/BioDepot/BioDepot-workflow-builder#manual.
FAQ: https://github.com/BioDepot/BioDepot-workflow-builder#faq.

Cell Systems, Volume *9*

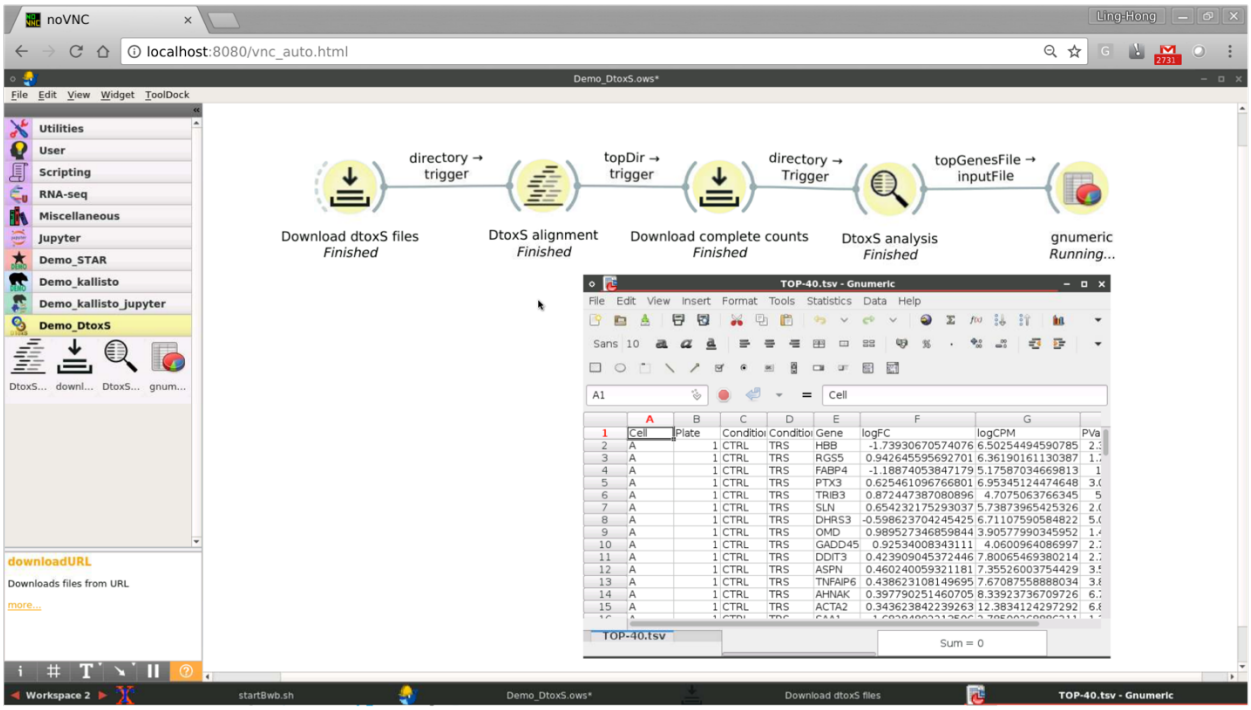
Supplemental Information

Building Containerized Workflows

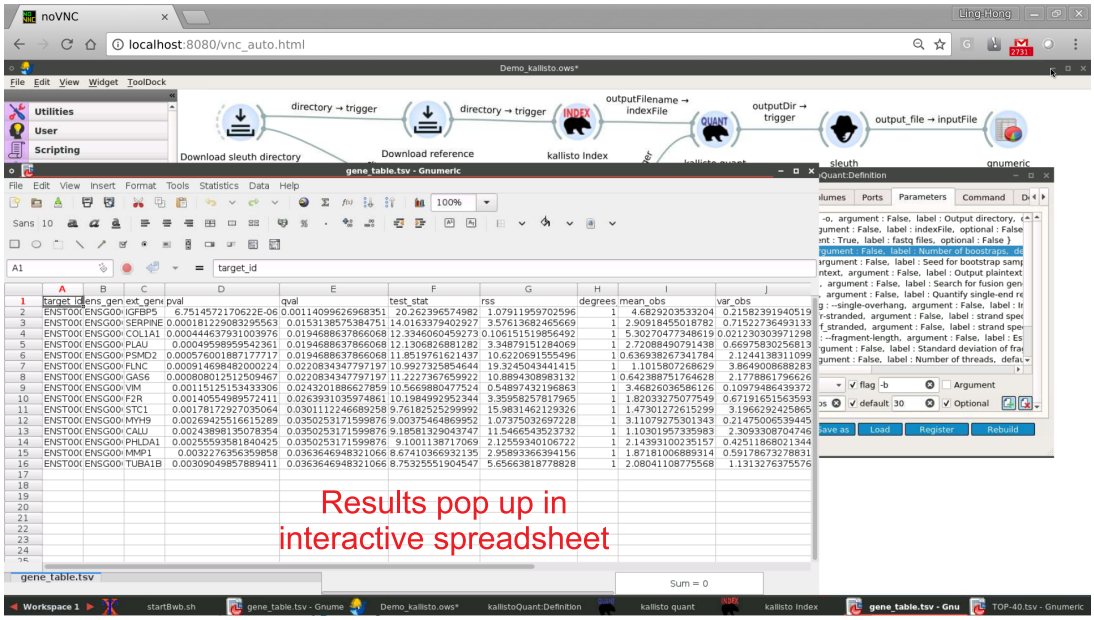
Using the BioDepot-Workflow-Builder

Ling-Hong Hung, Jiaming Hu, Trevor Meiss, Alyssa Ingersoll, Wes Lloyd, Daniel Kristiyanto, Yuguang Xiong, Eric Sobie, and Ka Yee Yeung

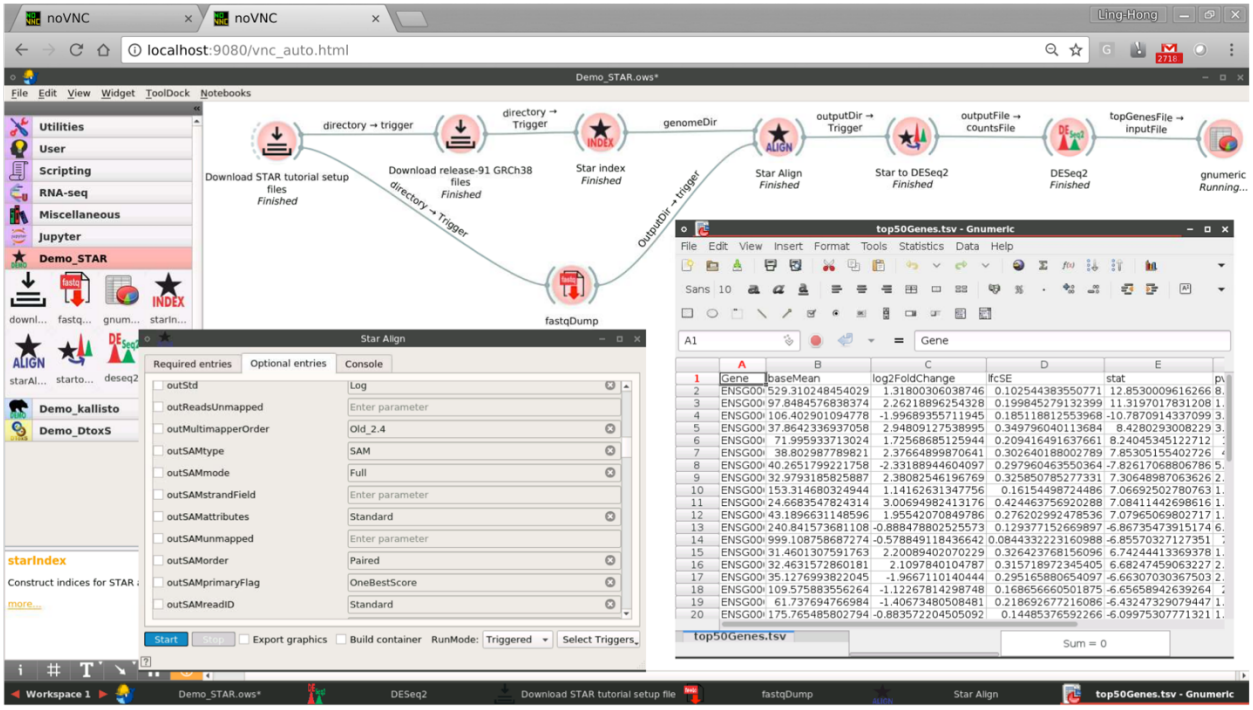
Supplemental Figure S1. Related to Figure 1 and the STAR Methods. Screenshot of DToxS workflow demo: The DToxS RNA-seq workflow is implemented as a demonstration of the Bwb. The connected workflow consists of five connected widgets, and processes RNA-seq fastq files to obtain a list of differentially expressed genes. This list is displayed using the gnumeric spreadsheet.



Supplemental Figure S2. Related to Figure 2 and the STAR Methods. Graphical output of kallisto-sleuth workflow. Sleuth has its output linked to the trigger of the gnumeric spreadsheet. When sleuth is finished processing it sends the output to the trigger which prompts the gnumeric application to read the output CSV file and display the results. The popup window is a normal gnumeric window enabling the user to interact with the process to visualize the data while saving results to a host file.



Supplemental Figure S3. Related to the STAR Methods. STAR/DESeq2 RNA-seq workflow on remote servers: The Bwb workflow consists of nine widgets and implements a RNA-seq differential gene expression pipeline. This is very similar in structure to the kallisto-sleuth pipeline. However, the STAR aligner requires more RAM (32 GB) than available on our laptop and the screenshot image depicts a browser connecting to the Bwb application running on a remote local firewalled server. The connection is established using SSH tunneling. The gnumeric application displays a popup window (lower right window) to view the final results and works identically when run remotely. The scrollable lower left window pops up upon clicking the STAR align widget. It shows some of the many parameters that the STAR aligner uses, that are carried along with the Bwb workflow that are easily customized.



Supplemental Figure S4. Related to Figure 1, Figure 2 and the STAR Methods. Widget panels: each widget has two tabbed panels. Double-clicking on the kallisto-quant widget in the kallisto-sleuth pipeline launches a data entry panel with a series of tabs. The numerous optional parameters that control the way that kallisto performs its quantifications are exposed by clicking on the 'optional' tab. Right-clicking the kallisto-quant widget, and choosing edit brings up the widget definition panel. This reveals the settings that define the widget itself. The blue highlighted selection in the parameters tab of the kallisto:definition window shows the parameters defining the number of bootstrap options. The user can enter new values into the definition:window to change the default number of bootstraps, for example. Finally, the black background window on the left is the console of the kallisto index. It displays the messages being printed by the widget as it processes the data. The data remains in the window for review until cleared.