Endofday: A Container Workflow Engine for Scalable, Reproducible Computation

Joe Stubbs*, Stephen Talley†, Walter Moreira*, Rion Dooley* and Ann Stapleton†
*Texas Advanced Computing Center
University of Texas, Austin, Austin, TX 78758-4497
Email: jstubbs@tacc.utexas.edu, wmoreira@tacc.utexas.edu, dooley@tacc.utexas.edu
† University of North Carolina, Wilmington, Wilmington, NC 28403-5904
Email: spt6655@alum.uncw.edu, stapletona@uncw.edu

***Abstract*—Container technologies such as Docker [1] are transforming the way distributed systems are deployed onto cloud platforms by providing a simple mechanism for packaging and isolating an application and its dependencies from the host machine on which it is running. The same ideas and technologies can be applied to computational science applications to obtain exceptional ease of installation and reproducibility of results. In this paper, we introduce endofday [2], a workflow engine that orchestrates a directed acyclic graph (DAG) of computational science apps where the nodes of the DAG are Docker containers. The endofday engine enables users to execute entire workflows of science applications without actually installing any of the applications themselves. As an example, we present the Validate [3] system, a suite of software applications for testing the accuracy and precision of Genome Wide Association methods, and illustrate how it can be run using endofday with zero installation. We also show how endofday integrates with the Agave [4] platform's application catalog and compare running Docker containers on cloud systems to running traditional applications on systems like Stampede.**

***Keywords*—workflow, containers, Validate, Agave, Docker.**

I. Introduction

By leveraging features of the Linux kernel such as cgroups and namespaces, containers enable developers to package, deploy, and execute their applications with exceptional independence and isolation from the host system as well as from other processes running therein. The container model differs from the traditional virtual machine model by virtualizing operating system calls instead of hardware interfaces. Containers run as processes in userspace and share the underlying host's kernel, but each container runs within a rooted file system and isolated network stack. In particular, out of design and necessity, containers include all file dependencies needed to execute the application they contain. The result is an independent, executable package that relies only on the kernel and container "runtime." If resource isolation is needed, the kernel can limit the CPU, memory and network available to a given container. The model also makes containers much more lightweight than traditional virtual machines: modest commodity servers can easily run hundreds of containers simultaneously. In fact, a single Raspberry Pi 2 with a HypriotOS was recently shown to be able to run over 2,300 web server containers at once [5].

While the first container technologies were notoriously difficult to work with, the Docker platform [1] has revolutionized distributed systems and cloud computing over the last two years by providing a system that greatly simplifies container creation, execution, and management. In the same way, Docker containers can be used to simplify installation, deployment, and execution of scientific applications, leading to greater reproducibility of the scientific computations themselves.

Typically, computational experiments involve multiple steps with different scientific applications at each step working together to perform some larger task. Often times there are dependencies between steps implying a certain order of execution. Workflow engines accommodate such needs: given a definition of tasks, their inputs, and their outputs, a workflow engine executes the tasks in the correct order, scheduling applications and data dependencies as needed to ensure correctness while additionally providing some level of re-runnability. A given workflow can be associated with a directed acyclic graph (DAG) where the nodes on the graph correspond to steps (or application invocations) in the workflow and the edges correspond to dependencies between the steps. It is common for inputs and outputs to be defined in terms of files or directories on a file system, but some workflow engines enable more general notions such as records in a database.

In this paper we introduce endofday [2], a workflow engine designed to accommodate workflows of Docker containers, and examine its use with the Validate [3] system, a set of applications for genome wide association studies. Using endofday, we show how users can execute entire workflows of Validate applications on any Linux machine that has Docker installed. No additional software is required, including the Validate applications themselves.

Dependencies and outputs in endofday workflows are defined in terms of files or directories and are managed between steps by the engine through the use of Docker volume mounts. An endofday workflow is described using YAML [6] syntax, a human readability data serialization standard. Thus, an endofday workflow definition file doubles as documentation of the experiment itself and can include arbitrary comments to that end.

In addition to running arbitrary Docker containers, endofday also has support for executing applications defined in the Agave [4] platform's application catalog. Agave is a hosted science-as-a-service platform developed at the Texas

Advanced Computing Center for hybrid-cloud, HPC and HTC computing. Agave provides backend services for several science gateways including the iPlant collaborative [7], a large NSF project to develop cyberinfrastructure for life sciences research. Agave provides a set of restful APIs for registering and leveraging storage and execution servers available on the public internet. Once systems are registered with Agave, users can move data, register applications and launch jobs against remote schedulers with simple http requests. Through it's integration with Agave, endofoday can leverage applications living on virtually any server connected to the public internet. Thus, workflows comprised of applications from any such server can be built, executed and managed from any computer without installing any software other than the endofday binary itself. Moreover, the entire workflow can be shared and reproduced simply by sharing the workflow definition file.

The rest of the paper is organized as follows: Section II gives a review of container technology and how it helps solve the reproducibility problem in computational science. It then discusses workflow engines in general and compares some of the more popular solutions to endofday.

In Section III we give a detailed overview of the endofday system, its support for local, hybrid and remote executions, as well as its support for executing Agave applications.

In Section IV, we introduce the Validate software system for testing the accuracy and precision of Genome Wide Association Studies, and compare two approaches to using endofday to run Validate workflows. We close the paper with some final remarks and areas for future development in Section V.

II. Background and Related Work

A. Containers and Reproducibility

While the surge in prominence for Docker is relatively recent, containers trace their roots back to technologies such as Unix chroot, first introduced in Unix 7 in 1979. More recent technologies such as OpenVZ [8], LXC [9] and BSD Jails [10] have been around for at least a decade and provide more advanced container features. What seems to set Docker apart from its predecessors is its exceptional ease of use, not only in executing containers and managing their run time, but also in creating, distributing and sharing Docker images, which are a container template for the Docker ecosystem. Docker images are described using a single text file, referred to as a Dockerfile, which includes simple commands for adding files and metadata to an initially empty rooted file system. A built image is essentially just that—a tar archive representing an independent file system for an application together with a small amount of metadata used by the Docker runtime for determining configuration options of the containers that are created from it. By including all necessary dependencies in the image, a Docker container can run an application on any Linux host with no additional software installation.

Reproducibility in computational science has long been a top priority for obvious reasons. Nevertheless, scientific codes are notoriously difficult to build and maintain, and independent users are often stymied when trying to install applications on new systems to reproduce experiments. Different versions of operating system distributions, libraries, packages, or compilers can lead to repeating the build process from scratch. Obtaining the original source code for the application can also sometimes be challenging. With Docker, all of these dependencies can be described in the Dockerfile and packaged into an image. Often times the Dockerfile takes only a few minutes to write, but once written, allows anyone to rebuild the image with a single command. Additionally, using the public Docker hub or a number of other commercial registries, the image can be distributed to the rest of the world with ease.

B. Workflow Engines

Docker goes a long way towards solving the reproducibility problem for single applications, however most computational experiments are comprised of several steps involving multiple applications. It is therefore desirable to be able to reproduce an entire workflow of computations, where each step in the workflow is given by the execution of a Docker container.

Like container technologies, workflow engines have been around for decades (for background, see [11] and [12]). Generally speaking, workflow engines give practitioners the ability to specify and execute multiple steps comprising a computational experiment within a collection of definition documents. Some are command-line based, some live in a graphical interface such as a workbench or web browser, and others are integrated directly into a science gateway. Workflow engines can generally be subdivided into two categories: engines catering to specific scientific domains and engines that are domain agnostic. Engines built for specific subdomains of science often target the actual scientist as the end user with abstractions that represent specific tools or notions in the discipline. These kinds of engines are easiest for scientist to use directly but often have the disadvantage of being difficult to update or extend. As a result, maintenance becomes a significant challenge over time, as new updates must be made to the tools to keep up with advances in the domain.

On the other end of the spectrum are domain-agnostic engines. These tools are generally more powerful and flexible but often require significant understanding of the underlying cyberinfrastructure to operate. Often times users must be familiar with HPC clusters, grids, clouds, web service technologies such as REST and WSDL, databases, or other technologies from computer science. Additionally, some of the more robust engines are difficult to install, requiring complex configuration to enable interactions with HPC systems. These requirements make it difficult for actual scientists to begin using the tool.

With endofday, our goal was to develop a domain-agnostic system that was simple enough to be used directly by scientists possessing even a modest familiarity with Docker and the Unix command line. The key to endofday's approach is the Docker container abstraction; using containers, the endofday engine is able to manage the installation of tools and interaction with underlying cyberinfrastructure. When traditional HPC applications are needed, endofday leverages the Agave platform to

again keep the cyberinfrastructure details hidden from the end user. Moreover, endofday is trivial to install and only requires minimal configuration when interacting with Agave.

C. Existing Tools

A complete survey of workflow engines is beyond the scope of this article, but we highlight some of the most popular solutions in use today and compare them to endofday.

Pegasus [13] is a mature and robust workflow management system capable of scaling executions across a distributed and heterogeneous compute environment. Originally developed in 2001, Pegasus takes an end user-provided abstract workflow definition, written in XML, and translates it into an executable workflow using information about the compute resources available in the environment. This information is configured in Pegasus through a variety of catalogs such as the Replica Catalog (logical file lookups), Translation Catalog (locations of user-installed executables), and Site Catalog (computational and storage resources). Pegasus includes advanced features such as compile time and runtime optimization, deep provenance, failed workflow recovery, and the Pegasus Dashboard, a rich web interface for monitoring and debugging workflow executions. While abstract workflow definitions provide a clean separation of concerns between the computational science being performed and the available compute environment, the challenge comes in configuring and maintaining the environment. Storage systems, execution systems, schedulers and application binaries all must be installed and configured within Pegasus before they can be utilized.

The Taverna [14] tool suite combines local executables with remote web services calls in REST or WSDL format to form complex workflows. Like Pegasus, Taverna workflows are described in XML, and the execution engine can leverage local, cloud, and traditional HPC resources. The Taverna workbench provides a graphical user interface for building workflows. www.myexperiment.org provides a public repository of workflows primarily for the life-sciences community. Taverna applications (called tools) are described in XML and must be installed on the machines where they will be executed. Tool descriptions can be imported from remote registries using the workbench.

Galaxy (see [15], [16] and [17]) is an open-source web-based software platform for bioinformatics. The platform itself can be installed and run locally, or users can take advantage of the many hosted instances, including "Galaxy Main" hosted at the Texas Advanced Computing Center. The platform includes support for building and executing workflows comprised of Galaxy "tools" as long as the software and all dependencies have been previously installed into the given Galaxy instance by the system administrator. Virtually any piece of software can be used to create a Galaxy tool, but it must first be described to Galaxy through a series of XML configuration files. By default, Galaxy executes tools on the local system. With additional configuration it can launch jobs on a cluster with a scheduler such as TORQUE or PBS, though a shared file system between the Galaxy server instance and the cluster is required. Additionally, tools can be installed from public Galaxy toolsheds, catalogs of bioinformatics applications together with their Galaxy configurations. While Galaxy can automate the installation of some dependencies, others cannot be provided and must be manually installed. By default, Galaxy executes tools on the local system, but with additional configuration it can launch jobs on a cluster with a scheduler such as TORQUE or PBS, though a shared file system between the Galaxy server instance and the cluster is required. While the Galaxy Pulsar runner can overcome the shared file system requirement, valid tools may not work properly if, for example, they reference hard-coded file paths for inputs, and set-up and installation are non-trivial.

By leveraging Docker images and Agave applications, endofday minimizes the setup and installation needed to execute workflows. In the case of Docker, any image available from the public registry [18] can be used: if the image is already installed on the execution system, endofday will use that, otherwise, it will be automatically downloaded before executing the application. These executions can take place on any host that has the Docker daemon installed on it, so issues of porting and scaling become trivial. In the case of an Agave application, the deployment and execution hosts have already been registered with Agave, so no additional software installation or configuration is needed.

III. ENDOFDAY: A WORKFLOW ENGINE FOR DOCKER CONTAINERS

At its core, endofday is an open-source, command-line Python application build on top of the pyyaml [19] and pydoit [20] libraries to execute workflows of Docker containers from a simple YAML definition file describing a DAG of tasks. It first performs dependency analysis to build the DAG and then schedules the execution of the containers while managing the flow of data throughout. endofday makes use of Python's multiprocessing library to launch independent steps in parallel on the host in which it is running. Additionally, it can leverage the Agave Platform's Docker compute cluster to launch either individual containers or entire workflows in the cloud. This feature enables seamless transition from executing an entire workflow on one's personal machine, to executing part or all of it in a remote cloud, freeing up local resources for other tasks.

A. Local Execution

The endofday application itself ships as a Docker container for ease of installation: simply pull the official image from the public hub and begin using it. Alternatively, the Docker image can be built from source using a Dockerfile provided in the public repository hosted on Github (https://github.com/joestubbs/endofday/). Once the Docker image for endofday is obtained, a single setup command is run to install a small bash script, endofday.sh, in the current working directory. No configuration is needed. Workflows can then be executed with the command

```
endofday.sh /path/to/workflow.yml
```

```
inputs:
  - fasta_input_1 <- /home/jstubbs/data/g001.fasta
  - fasta_input_2 <- /home/jstubbs/data/g002.fasta
```

Fig. 1. An Example inputs section defining two global inputs

and relative paths in the workflow definition will be resolved against the current working directory.

Endofday workflows are defined in YAML using a syntax that will be familiar to users of other Docker tools such as Docker Compose [21]. A workflow definition file is made of a `name` field together with three sections: `inputs`, `outputs` and `processes`. The inputs section defines a list of global inputs in a `label -> source` format. The label can be any unique string and is used to refer to the input in other parts of the definition file, while the source is a path to a file or folder on the host. When configured with an Agave account, the input source can be any Agave URL—essentially any file or folder on any system registered with Agave that the user has access to.

The files and directories defined in the input section are namespaced within the `inputs` keyword and can be referenced in subsequent sections of the workflow using the assigned label. For example, the first input defined in the inputs section in Figure 1 would be referenced using `inputs.fasta_input_1` while the second would be referenced with `inputs.fasta_input_2`.

Similarly, a section of global outputs for the workflow can be defined. The global outputs definition is optional and primarily serves as documentation of the workflow, but in a future release, endofday will be able to compose multiple workflows, and it will be possible to reference global outputs of one workflow as inputs to another workflow. We plan to make an `include` directive available which would allow a workflow to reference inputs, outputs and processes from another workflow definition in the same working directory or specified by an absolute path. In such a situation, the objects from a given workflow can be referenced using the worflow's name; for example, `myworkflow.inputs.fasta_input_1` might refer to a global input named `fasta_input_1` within a workflow named `myworkflow`.

The main section of an endofday workflow definition file is the processes section in which the actual steps of the workflow are defined. Each process is given by a YAML mapping under a user-defined id for the process. The id can be any valid YAML mapping key as long as it is unique within the processes section. The recognized keys within a given process include `image`, `description`, `inputs`, `outputs`, and `command`. The value for the `image` key is simply the Docker image that should be used for the process and similarly, `command` is the command string passed to the Docker daemon when executing the container. The `inputs` sections is a list of strings of the form `source -> dest`. Here, `source` is a reference to either a global input or an output of another task while `dest` is a path in the container. Similarly, the outputs

```
processes:
  fast_lmm:
    image: validate/fast-lmm
    description: GWAS analysis for large datasets.
    inputs:
      - inputs.fam_input -> /data/inputs/fam_input
      - inputs.ped_input -> /data/inputs/ped_input
    outputs:
      - /data/outputs/lmm.csv -> output

  winnow:
    image: validate/winnow
    description: Known-truth testing analysis
    inputs:
      - fast_lmm.output -> /winnow/inputs/lmm.csv
      - inputs.known_truth -> /winnow/know_truth
    outputs:
      - /winnow/results.txt -> output
```

Fig. 2. An Example of the processes section

section is a list of strings of the the form `path -> label` where `path` is a file path to an output in the container and `label` is a unique string used to reference the output in other sections of the workflow.

Figure 2 provides an example processes section in which two processes are defined. The first process runs the FaST-LMM program for genome wide association studies, explored further in the subsequent section. It references two inputs from the global inputs section (not depicted) and mounts them into the container's `/data/inputs` directory. A single output is defined for the container path `/data/outputs/lmm.csv` and given the label `output`. It can be be referenced in other processes by prefacing it with the process namespace, i.e., `fastlmm.output`, just as it is in the definition of the winnow process that follows.

B. Support for Agave Apps and Hybrid Execution

Just as endofday can execute a workflow of Docker containers, it also supports executing applications in the Agave application catalog with a similar syntax. Agave applications are launched by submitting requests to the Agave jobs service. The applications run on the remote execution systems defined for each application which means that endofday can be used to launch workflows comprised of virtually any application on any server in the world. Complete documentation of the workflow syntax and command line arguments is available from the endofday Read The Docs page [22]. It is possible to mix Agave applications and Docker container applications in the same workflow: the containers will run locally while the Agave apps will run on their defined execution hosts. If outputs from Agave apps are needed as inputs to tasks running locally, endofday will download the necessary outputs to the local file system.

When executing Agave applications, endofday uses Agave's OAuth2 implementation to authenticate with the jobs service. Note that Agave's jobs service uses a separate set of credentials

when authenticating with the actual storage or execution system, typically SSH keys or grid (X509) credentials, though many different kinds of authentication are supported. These credentials are provided to Agave when the system is registered. It is not uncommon for portal administrators to register these systems and share them with their users, meaning that end users may not even be aware of the system credentials being used.

User's supply their OAuth credentials through endofday's configuration file, a small text file in INI format. Users can specify additional configurations in this file such as the Agave storage system to use when archiving output files and an email address for notifying users when jobs finish. We note that this configuration file is only needed when working with the Agave platform.

C. Remote Execution

There is also a hosted endofday service as part of the Agave platform that can be used to execute entire workflows in the Agave cloud. This can be accomplished by simply passing the `--agave` flag when executing the workflow. When the `--agave` flag is passed, the endofday engine uploads the workflow definition file and any global input files residing on the local file system to an Agave storage system (either the user's default storage system, or one configured in endofday) and submits a single job to Agave to asynchronously execute a version of endofday registered as an Agave application. Once the job has been submitted, the local endofday engine exits, freeing the local system resources. By adding an email address to the endofday configuration file, Agave will notify the user when the workflow execution completes via email.

IV. Case Study - Validate: A Workflow for Genome Wide Association Studies

A. Overview

The Validate Workflow is a series of programs designed to test the accuracy and precision of analysis tools for either genome wide association studies (GWAS) or quantitative trait loci (QTL) analysis. Validate provides information on both effect sizes for single nucleotide polymorphisms (SNPs) and the statistical significance of certain SNPs in various models, when given known effect-size and SNP inputs from simulations. The Validate Workflow provides a means not only to judge the appropriateness of an analysis tool for a given data set, but also to integrate existing tools into a pipeline or workflow for testing. The Validate Workflow consists of four steps: Simulate or another suitable simulation app, an analysis tool (for the sake of this demonstration, we have used the tool FaST-LMM), Winnow, and Demonstrate. The Validate Workflow is currently on its fourth iteration, version 0.9, and the source code for the workflow may be found on Github [3]. Future versions of the software will include prediction methods for missing phenotype data, ensemble analysis for reduced error in SNP classification, and extended file format types such as hdf5 for large-scale data.

B. Components

Simulate is a forward-in-time genetic individual-based simulation program written in Python. For a given population or sub-populations, Simulate creates a quantitative trait for that population based on an additive model [23]. Once Simulate has established the final population state, it generates the following outputs: a genomic information file in CSV format, a phenotype file with the final quantitative trait value for each individual, and a "known-truth" file detailing the original effects and contributing SNPs or individuals for later use in the workflow.

FaST-LMM (Factored Spectrally Transformed Linear Mixed Models) is a genome wide association studies program from Microsoft Research designed to handle extremely large data sets, and provides a good example of a tool that could be analyzed with Validate. Further information on FaST-LMM may be found at the Microsoft Research Github page [24].

Winnow is a Python program with functions for known-truth testing of analysis tools [25]. Given the known truth of a data sets significant SNPs and effect sizes under GWAS analysis, Winnow evaluates the scores from the output of a GWAS analysis tool in comparison to that known-truth. Then, it generates a series of fit statistic values. These fit statistics are validations from binary classifier algorithms, and therefore include many statistics one might find in or derive from a confusion matrix (e.g. true/false positives, sensitivity, precision). In addition to the results, Winnow also creates a parameter file which details certain aspects of a run such as the output file name, the threshold for statistical significance, and the p-value adjustment method used, if any. Finally, Demonstrate is an R program for producing human-readable summary graphics from Winnow results.

```
docker run \
  -v fastlmm_out:/tmp/GWAS_out \
  -v known_truth.ote:/inputs/known_truth \
  taccsciapps/winnow \
  --verbose \
  --Folder tmp \
  --Class known_truth \
  --SNP SNP \
  --Score Pvalue \
  --beta SNPWeight \
  --kttype OTE \
  --seper comma \
  --kttypeseper whitespace \
  --filename YAML_Winnow_Results
```

Fig. 3. An example of invoking docker

C. Validate - Source Code and Docker Images

The components of the Validate system are all open source and available from the project's Github repository. As such, users who are interested in using Validate are free to clone the repository and install the applications on any system. However, installing from source is involved and somewhat

```yaml
name: Validate_wf_docker

inputs:
  - ped_input <- data/toydata.ped
  - map_input <- data/toydata.map
  - bed_input <- data/toydata.bed
  - bim_input <- data/toydata.bim
  - fam_input <- data/toydata.fam
  - pheno_input <- data/toydata.phe
  - known_truth <- data/fakekt.ote

outputs:
  - demonstrate.comptable
  - demonstrate.TPhist
  - demonstrate.FPhist
  - demonstrate.TPvsFP
  - demonstrate.AUCvsMAE

processes:
  fastlmm:
    image: taccsciapps/fastlmm
    description: Analyzes the data to produce GWAS output
    inputs:
      - inputs.ped_input -> /tmp/test.ped
      - inputs.map_input -> /tmp/test.map
      - inputs.bed_input -> /tmp/test.bed
      - inputs.bim_input -> /tmp/test.bim
      - inputs.fam_input -> /tmp/test.fam
      - inputs.pheno_input -> /tmp/pheno.txt
    outputs:
      - /fastlmm/LMM_Docker_Results.csv -> GWAS_out
    command:
      fastlmmc -verboseOutput -bfile /tmp/test
      -fileSim /tmp/test -pheno /tmp/pheno.txt
      -out LMM_Docker_Results.csv



  winnow:
    image: taccsciapps/winnow
    description:
      Produces fit statistics for determining
      appropriateness of GWAS analysis tool
    inputs:
      - fastlmm.GWAS_out -> /samples/GWAS_out.csv
      - inputs.known_truth -> /kt.ote
    outputs:
      - /outputs/YAML_Winnow_Results.txt -> Winnow_out
    command:
      --verbose --Folder /samples --Class /kt.ote
      --Snp SNP --Score Pvalue --beta SNPWeight
      --kttype OTE --seper comma --kttypeseper whitespace
      --filename  /outputs/YAML_Winnow_Results


  demonstrate:
    image: taccsciapps/demonstrate
    description:
      Produce human-readable graphics from the Winnow
      output of the previous step
    inputs:
      - winnow.Winnow_out -> /tmp/results.txt
    outputs:
      - /tmp/ComparisonTable.csv -> comptable
      - /tmp/'TP Histograms.pdf' -> TPhist
      - /tmp/'FP Histograms.pdf' -> FPhist
      - /tmp/Test_Run_Pos_Plot.pdf -> TPvsFP
      - /tmp/Test_Run_Error_Plot.pdf -> AUCvsMAE
    command:
      Rscript /usr/bin/DemonstrateRun.R /tmp TRUE
      Test_Run_Pos_Plot.pdf TRUE Test_Run_Error_Plot.pdf TRUE
```

```yaml
name: Validate_wf_Stampede

inputs:
  - ped_input <- agave://val.storage//data/toydata.ped
  - map_input <- agave://val.storage//data/toydata.map
  - bed_input <- agave://val.storage//data/toydata.bed
  - bim_input <- agave://val.storage//data/toydata.bim
  - fam_input <- agave://val.storage//data/toydata.fam
  - pheno_input <- agave://val.storage//data/toydata.phe
  - known_truth <- agave://val.storage//data/fakekt.ote

outputs:
  - ComparisonTable.csv
  - TP Histograms.pdf
  - FP Histograms.pdf
  - True Positives vs. False Positives.pdf
  - Plot of AUC by MAE.pdf

processes:
  fastlmm:
    app_id: FaST-LMM-2.07
    execution: agave_app
    description: Step 1
    inputs:
      inputFAM: [“inputs.fam_input”]
      inputPED: [“inputs.ped_input”]
      inputBED: [“inputs.bed_input”]
      inputBIM: [“inputs.bim_input”]
      inputMAP: [“inputs.map_input”]
      inputPHENO: [“inputs.pheno_input”]
    parameters:
      MainFileset: "P"
      SimFileset: "BEDBIMFAM"
      output: "YAMLTest_LMM.csv"
      verboseOutput: 0
    outputs:
      - YAMLTest_LMM.csv -> output

  winnow:
    app_id: Winnow-0.9
    execution: agave_app
    inputs:
      Folder: [“fastlmm.output”]
      Class: [“inputs.known_truth”]
    parameters:
      SNP: "SNP"
      Filename: "YAML_Winnow_Results"
      Score: "Pvalue"
      beta: "SNPWeight"
      kttype: "OTE"
      seper: "comma"
      kttypeseper: "whitespace"
    outputs:
      - YAML_Winnow_Results.txt -> output

  demonstrate:
    app_id: Demonstrate-0.9
    execution: agave_app
    inputs:
      dir: [“winnow.output”]
    parameters:
      make_pos_plot: 1
      pos_plot_title: "'Test Run - Pos Plot'"
      make_error_plot: 1
      error_plot_title: "'Test Run - Error Plot'"
      extra_plots: 1
    outputs:
      - ComparisonTable.csv -> comparison_table
      - TP Histograms.pdf -> tp_histograms
      - FP Histograms.pdf -> fp_histograms
      - True Positives vs. False Positives -> tp_vs_fp
      - Plot of AUC by MAE.pdf -> auc_by_mae
```

Fig. 4. Full examples of workflow definitions in YAML

error-prone since the various components have a multitude of dependencies including R, Python, and several scientific libraries. Some of the libraries such as numpy are notoriously challenging to install since they depend on C extensions and Fortran libraries as well as linear algebra packages. Care must be taken to ensure that the right versions of each dependency are installed, and this entire process must be repeated on each new system and for each separate version of Validate the user wishes to run.

Fortunately, all the Validate applications have been packaged into Docker images available on the public Docker hub. Users can therefore run any application in the Validate suite using their local machine or any host with the Docker daemon installed. For example, the command in Figure 3 will run Validate's Winnow application against a (previously generated) FaST-LMM output and a known-truth file.

Using endofday, users can run entire workflows of Validate applications by defining the workflow in a YAML file. Two examples of Validate workflow description files are included (see 4).

Moreover, Validate Docker images are tagged with the version of the software they represent. If no tag is supplied, as in the example above, the latest image is pulled, but by supplying different tags, users can seamlessly switch between versions of the software without worrying about the potential for conflicting dependencies. For instance, the image `taccsciapps/winnow:0.9` will use version 0.9 of the Winnow software.

D. Using Validate Through Agave

The software components of Validate have been installed on the Stampede supercomputer at the Texas Advanced Computing Center and have been registered as applications within Agave's iPlant tenant. Any user with an iPlant account can run these applications on Stampede using a shared community account through a variety of methods including iPlant's discovery environment which provides a graphical web interface [7].

Therefore, users can leverage the support for Agave apps in endofday to launch entire Validate workflows on Stampede. An example workflow definition utilizing the Stampede versions analogous to the previous Docker example is included in included (see 4). The syntax for such a YAML file is remarkably similar to that of one leveraging Validate Docker containers making it easy to convert from one to the other.

Note that it is not nearly as easy to move the non-Docker versions of validate from Stampede to another execution host. In brief, the applications must be re-registered with Agave for the new host, and this will only work once all the dependencies have been installed on the new system.

It should be noted that developing the Docker images for the Validate applications was in fact much easier than installing the applications even once on Stampede, primarily because generic images containing dependencies already existed in most cases. For example, the Dockerfile for the Winnow image is a trivial three lines as it descends from a generic scientific Python image that includes numpy, scipy, matplotlib, etc.

V. Conclusion and Future Work

In this paper we showed how the endofday application leverages Docker's powerful container abstraction to execute and reproduce scientific workflow computations with exceptional ease. Additionally, by leveraging applications in the Agave catalog, we established that endofday can execute workflows across heterogeneous computing resources comprised of virtually any machine connected to the internet. We also introduced the Validate software system for testing the accuracy and precision of Genome Wide Association methods and demonstrated how the endofday engine can be used to execute Validate workflows with zero installation on any machine running Docker as well as on the Stampede supercomputer.

The endofday project is still very young and actively being developed, and several areas of future work are planned. In general, endofday will add improved provenance for all workflow executions, leveraging the provenance support already in Agave for those applications. Better validation will be added to improve user experience when initially developing workflow definitions. The hosted offering in the Agave platform will be expanded to include a powerful API for introspecting details about a workflow execution.

For Docker applications, endofday plans to expand support for executing on remote clusters so that users have a seamless way not only of transitioning to the Agave cloud but any cluster of Docker hosts. Initially, we plan to support clusters running Docker Swarm [26] and Apache Mesos [27] with support for additional cluster schedulers coming later.

For Agave applications, endofday will soon support more flexibility in archiving intermediate results so that users can decide which data sets should be archived.

Acknowledgment

This material is based upon work supported by the National Science Foundation Plant Cyberinfrastructure Program (DBI-0735191), the National Science Foundation Plant Genome Research Program (IOS-1237931 and IOS-1237931), the National Science Foundation Division of Biological Infrastructure (DBI-1262414), the National Science Foundation Division of Advanced CyberInfrastructure (1127210), and the National Institute of Allergy and Infectious Diseases (1R01A1097403).

References

[1] (2014) Build, ship and run any app, anywhere. Last access: 2015-03-13. [Online]. Available: http://docker.com

[2] (2014) endofday. Last access: 2015-11-11. [Online]. Available: https://github.com/jstubbs/endofday

[3] (2015) Uncw-iplant/validate-master. Last access: 2015-11-30. [Online]. Available: https://github.com/UNCW-iPlant/Validate-Master

[4] R. Dooley *et al.*, "Software-as-a-service: The iplant foundation api," IEEE. 5th IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS), 2012.

[5] (2015) Update: Raspberry pi dockercon challenge. Last access: 2015-03-13. [Online]. Available: https://blog.docker.com/2015/09/update-raspberry-pi-dockercon-challenge/

[6] (2014) Yaml: a human friendly data serialization standard for all programming languages. Last access: 2015-03-13. [Online]. Available: http://yaml.org/

[7] S. A. Goff *et al.*, "The iplant collaborative: Cyberinfrastructure for plant biology," *Frontiers in Plant Science 2*, 2011.

[8] (2014) Openvz virtuozzo containers wiki. Last access: 2015-03-13. [Online]. Available: https://openvz.org/Main_Page

[9] (2014) Linuxcontainers: Infrastucture for container projects. Last access: 2015-03-13. [Online]. Available: https://linuxcontainers.org/

[10] (2014) Jails: System administration. Last access: 2015-03-13. [Online]. Available: https://www.freebsd.org/doc/handbook/jails.html

[11] J. Yu and R. Buyya, "A taxonomy of workflow management systems for grid computing," *Journal of Grid Computing 3*, vol. (3-4), no. 10, 2005.

[12] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *Future Generation Computer Systems*, vol. 5, no. 25, p. 528540, 2009.

[13] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. da Silva, M. Livny, and K. Wenger, "Pegasus: a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[14] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. N. de la Hidalga, M. P. B. Vargas, S. Sufi, and C. Goble, "The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud," *Nucleic Acids Research*, vol. 41, 2013.

[15] J. Goecks, A. Nekrutenko, J. Taylor, and T. G. Team, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome Biol*, vol. 11, no. 8, p. R86, 2010.

[16] D. Blankenberg, G. V. Kuster, N. Coraor, G. Ananda, R. Lazarus, M. Mangan, A. Nekrutenko, and J. Taylor, "Galaxy: A web-based genome analysis tool for experimentalists," *Current protocols in molecular biology*, pp. 19–10, 2010.

[17] B. Giardine, C. Riemer, R. C. Hardison, R. Burhans, L. Elnitski, P. Shah, Y. Zhang, D. Blankenberg, I. Albert, J. Taylor, W. C. Miller, W. J. Kent, and A. Nekrutenko, "Galaxy: a platform for interactive large-scale genome analysis," *Genome research*, vol. 15, no. 10, pp. 1451–1455, 2005.

[18] (2014) Docker hub. Last access: 2015-03-13. [Online]. Available: https://www.docker.com/docker-hub

[19] (2014) Pyyaml - a yaml parser and emitter for python. Last access: 2015-11-11. [Online]. Available: http://pyyaml.org/

[20] (2014) Doit automation tool. Last access: 2015-11-11. [Online]. Available: http://pydoit.org/

[21] (2014) Docker compose. Last access: 2015-03-13. [Online]. Available: https://www.docker.com/docker-compose

[22] (2014) endofday documentation. Last access: 2015-11-11. [Online]. Available: http://endofday.readthedocs.org/

[23] (2015) Simulate - atmosphere manual - iplant collaborative wiki. Last access: 2015-11-13. [Online]. Available: https://pods.iplantcollaborative.org/wiki/pages/viewpage.action?pageId=21139107

[24] (2014) Microsoft genomics. Last access: 2015-11-11. [Online]. Available: https://github.com/MicrosoftGenomics

[25] (2015) Winnow - atmosphere manual - iplant collaborative wiki. Last access: 2015-11-10. [Online]. Available: https://pods.iplantcollaborative.org/wiki/display/atmman/Winnow

[26] (2014) Docker swarm. Last access: 2015-03-13. [Online]. Available: https://www.docker.com/docker-swarm

[27] (2014) Apache mesos. Last access: 2015-03-13. [Online]. Available: http://mesos.apache.org/