Skyport – Container-Based Execution Environment Management for Multi-Cloud Scientific Workflows

Wolfgang Gerlach [*,1,2], Wei Tang [2], Kevin Keegan [2,1], Travis Harrison [1,2], Andreas Wilke [2], Jared Bischof [1,2], Mark D'Souza [1,2], Scott Devoid [2,1], Daniel Murphy-Olson [2,1], Narayan Desai [3], Folker Meyer [2,1]
[1] University of Chicago, Chicago, Illinois, USA
[2] Argonne National Laboratory, Argonne, Illinois, USA
[3] Ericsson, San Jose, California, USA

ABSTRACT

Recently, Linux container technology has been gaining attention as it promises to transform the way software is developed and deployed. The portability and ease of deployment makes Linux containers an ideal technology to be used in scientific workflow platforms. Skyport utilizes Docker Linux containers to solve software deployment problems and resource utilization inefficiencies inherent to all existing scientific workflow platforms. As an extension to AWE/Shock, our data analysis platform that provides scalable workflow execution environments for scientific data in the cloud, Skyport greatly reduces the complexity associated with providing the environment necessary to execute complex workflows.

1. INTRODUCTION

Multiple disciplines have experienced a drastic increase in the size of data that undergoes analysis. Examples in physics (CERN [27]) and biology (Human Microbiome Project [21]) have achieved widespread recognition. In biomedical research, the recent developments in DNA sequencing technology have led to an expansion in biological sequence data that dwarfs prior achievements. This trend is particularly true in the field of metagenomics [20] where single samples contain sequence data (DNA/RNA, usually in the form of millions of short sequences of a few hundred bases) from entire microbial environments. Samples hundreds of gigabytes in size are common. Storage and analysis of such data has made it necessary to exploit grid and cloud computing resources with efficient workflow management systems, making it possible to process data quickly while at the same time preserving provenance.

∗ Corresponding author: wgerlach@mcs.anl.gov

DataCloud 2014 New Orleans, LA USA

Scientific workflows are usually represented as directed acyclic graphs (DAG) in which edges represent the flow of data, and nodes represent the tasks that operate on them [4]. Workflow tasks can require computationally expensive data processing or invocation of external web services, typically developed on top of one of several existing platforms. Generally, scientific workflows are complex software systems that consist of multiple software applications assembled together to perform several tasks that perform multiple transformations and analyses on the input data. Independent researchers write applications to solve specific problems (e.g., genome assembly, gene prediction and functional annotation) that require the selection of appropriate programming models, languages and software libraries. Each problem can have specific requirements that are independent of, possibly even incompatible with, the requirements for solving other problems within the same workflow. Additionally, a strong desire exists among scientists to use the same, well documented tools to solve a problem. This is generally for two reasons: the high cost to rewrite existing applications, and because of the importance of consistently using peer-reviewed software.

Current scientific workflows often includes dozens, or more, external dependencies (e.g., requires a specific version of Python, or requires a specific version of an R package that in turn is only supported in an older version of R itself). While all of these issues are solvable, the situation is surprisingly complicated. As an example, users could install an older version of R required for a specific package. Yet, the installation of any required R package will default to the latest version of said package, which will generate version conflicts and might be incompatible with the old version of R itself. Problems of this nature have led developers to offer all-in-one solutions (i.e., snapshots), which capture the entire development environment in a single VM that acts as the target runtime environment. The well known bioinformatics suite QIIME [5] or (Cloud-)BioLinux [8, 14] are examples with external dependencies contained in a single VM.

The basis for scientific workflow execution is one of many workflow platforms. They all have in common that when the workflow tasks are mapped to the computing resources, the executable and dependent environments are required to be preinstalled on the computing resources. This requirement limits the flexibility of building different kinds of workflows

and slight variations of executable versions results in several difficulties. First, it leads to increased difficulty in reproducing results. Second, it leads to increased difficulty in reusing methods. Thus, a standardized and reproducible application environment management is vital for running scientific workflows.

Here we present the Skyport-extension to our AWE/Shock ecosystem that utilizes Linux container virtualization technology to solve the software deployment problem inherent to all existing scientific workflow platforms. Instead of relying on virtual machines that are used in cloud infrastructures, Skyport uses Linux containers to achieve software isolation.

1.1 Existing Workflow Platforms

In recent years, various systems have been developed to help scientists design and execute workflows in a variety of disciplines. These systems attempt to automate utilization of distributed computing resources in a scalable manner, making it possible for non-computational specialists to utilize the latest developments in distributed computing to enhance their ability to process and store massive quantities of data. Examples include many tools that can be applied in a general way. Kepler supports job submission to Grid resources built by the Globus Toolkit [10]. While not directly supported by Kepler, the bioKepler [3] module supports multiple distributed data-parallel execution engines including Hadoop [24, 25]. Taverna is focused on GUI-based workflow creation with strong emphasis on use of external web services. While only providing limited support for running workflows on a grid, other software suites such as MOTEUR [11] and Tavaxy [1] make it possible to execute Taverna workflows on a grid. In Pegasus [6], workflows are explicitly represented as directed acyclic graphs. Distributed computing can be realized by submission of the workflows to a HTCondor pool [19]

Other workflow systems exhibit field specialization. Galaxy is one such system built specifically for multi-omic biological data. Galaxy has a simple graphical interface that allows users without command line experience to run workflows. The CloudMan [2] software package can be used to install Galaxy on a cloud infrastructure. Globus Genomics [16] is another solution for biological workflows; it provides an integrated solution following the *software as a service* model specifically designed for next-generation sequencing data.

In 2012, we introduced the AWE/Shock data analysis platform [18]. Our main design goal for this platform was scalability to efficiently exploit cloud computing resources for our metagenomics production pipeline MG-RAST. A core element to achieve this scalability is Shock, an object-based data management system that makes it possible to overcome performance limitations of shared filesystems such as Network File System (NFS) [12]. Independence from a shared file system also provides the ability to deploy compute workers in multiple clouds. We provide a brief overview of AWE/Shock in Section 1.5.

1.2 Software installation is challenging on all platforms

Computational workflow management systems require the inclusion of software tools and/or packages installed on a grid or custom virtual machines (VM) in a cloud-based environment. On a grid, installation of new software (e.g., a new version of software required for a single step in an existing workflow) is frequently labor intensive, commonly requiring intervention by a system administrator. Custom VMs are usually created by automated installation of tools onto a base image or are initiated from images that are preconfigured with a set of tools that have already been installed with their dependencies. The latter model is commonly used for bioinformatic tools such as Galaxy server, (Cloud-)BioLinux [8, 14], and QIIME [5].

In the case of grid or cloud-based solutions, modifications to individual tools/components are difficult. This is especially true in cases where end-users do not possess the skills required to build or configure computational resources. Installation of new, or merely updates to existing, software tools of any system can be a challenge. In addition, a common requirement in scientific applications is the ability to reproduce computational provenance: to install a specific version of a tool (including all originally used versions of all dependencies) such that computationally derived results can be exactly reproduced. While this is not a (direct) problem for system administrators, it typically becomes one when end-user scientists are unable to reproduce the computational environment necessary to produce or reproduce experimental results. The installation and configuration of software applications on all platforms is complicated by several issues. Some software and configuration must be managed by a privileged user. Kernel version, driver software and core operating system configuration cannot be isolated for separate applications on a single system. Most Linux distributions include a package-management tool to install and configure software packages, but programming languages (e.g., Perl, Python, R) also have specific library management tools with their own dependency management, which creates overlapping areas of influence. Finally, different applications can require contradictory versions of the same package. Consequently, achieving sufficient isolation of each application's runtime environment can be difficult.

Because of the dependency problem and application specific system wide configurations, nearly all cloud infrastructures utilize virtual machine technology on client machines to implement software isolation and resource control. In the following, we will give a short overview of such virtualization methods, including container based virtualization.

1.3 Linux container virtualization

Hardware virtualization can be implemented using hypervisors such as the Kernel Virtual Machine (KVM) [13], Xen [28], Hyper-V [29], or VMware ESX/ESXi [23]. A hypervisor runs virtual machines as guests; this provides a high degree of isolation and resource control. Each virtual machine is run as an independent computer system, with its own resources and a complete operating system. Multiple virtual machines can run on one host, each with an independent guest OS that can be different from the host's. The

resource control (i.e., CPU and RAM limit) that is possible with VMs makes them suitable to use in an *infrastructure as a service* model used by OpenStack [30] and commercial providers such as Amazon Elastic Compute Cloud (EC2) [31].

A different level of virtualization that can be used either alternatively or in addition to hardware virtualization is **operating system-level virtualization**. In this type of virtualization, the kernel of the operating system provides mechanisms to isolate processes in system or application containers. Containers are independent from each other, but share the same underlying operating system (i.e., the kernel and device drivers). Example implementations include LXC (LinuX Containers) [32], OpenVZ [33] and Solaris Containers [15]. An early version of such virtualization was introduced 1982 with the chroot operation that restricts the filesystem view on most Unix systems. A more sophisticated implementation based on chroot was introduced with FreeBSD jails in 1998. After 2002, various features were added to the Linux kernel that provided user namespaces for global system resources such as Network (interfaces and firewalling rules), hostname, filesystem mount points, interprocess communication (IPC), and ID number spaces for users and groups to make them appear to processes within the namespace (e.g., a container) as unshared global system resources. In 2007, cgroups (control groups) was merged into the Linux kernel. Cgroups can limit, measure, prioritize, control, and isolate resource usage (e.g., CPU, memory, disk I/O) of process groups. Using these kernel features as well as chroot, namespaces, cgroups and some additional kernel features, LXC (LinuX Containers)[32], a userspace interface, was introduced in 2008. This interface makes it possible to create and manage containers via an API and command line tools.

1.4 Docker

Another management tool for Linux containers is Docker [34]. Released as an open source project in 2013, Docker has rapidly achieved widespread use. It was originally designed to access Linux kernel namespace features via LXC, but has switched to use its own "libcontainer" library that directly accesses the kernel. In addition to creating containers, Docker provides mechanisms to deploy applications into the containers with a "Dockerfile". Docker supports a variety of storage backends, most support copy-on-write semantics such as DeviceMapper (a block-level copy-on-write system) and aufs (an union filesystem) [35]. Layered filesystems make it possible for multiple containers to run on the same underlying base image. When a container is started from an image, it uses the filesystem of that image. Modifications to the filesystem by a process in the container are written to a filesystem specific to that container, which makes it possible to run multiple containers independently on top of a single image. This copy-on-write mechanism (in contrast to a complete copy of the root file system for each container) makes the deployment of containers very fast, and reduces disk and RAM occupancy. The Dockerfile deployment process makes use of this by creating a new image from an existing base image (i.e., a filesystem layer), adding additional layers that describe differences between the base image and the fully deployed application. Docker's popularity is due, at least in part, to an included system that handles revision control; this system makes it very easy to share docker images via public (such as the default Docker Hub repositories) or private repositories. Typically, public repositories are hosted for free, while the private repositories are not. Docker creates an abstraction for machine-specific settings such as networking, storage, logging, and Linux distribution. This makes Docker containers portable to any Linux-based operating system that runs Docker and is built for the same architecture (usually x86-64). Docker currently supports all major Linux distributions, making it an extremely versatile tool.

Linux containers are gaining more and more popularity because they are more lightweight than VMs. They are lightweight because container images do not require the inclusion of an entire kernel and thus are much smaller than VMs. Starting a container consists of forking a process and inserting it into a new cgroup. VMs, on the other hand, require execution of a full boot process at startup. Because containers are so lightweight compared to VMs, container virtualization can now be used for scenarios where the use of VMs would have been too expensive.

Compared to running processes directly on the host (bare metal), Docker containers (using aufs [35], a union filesystem) exhibit marginal overhead with respect to performance. In contrast, virtualization with the KVM hypervisor imposes a much higher overhead. In cases where virtual machines can be replaced with linux containers, containers can improve the overall system performance [7].

1.5 The AWE/Shock platform

The motivating use case for the development of the AWE/Shock [18] data analysis platform was MG-RAST [26, 17], a bioinformatics pipeline that processes metagenomic data produced by next-generation DNA sequencing platforms. To handle the large and ever expanding size of sequencing data sets (data sets 1GB in size are commonplace, much larger are expected in the near future), our ecosystem was designed for scalable high-performance data processing. Using this technology in production, MG-RAST has successfully processed over 23 tera base pairs from more than 2000 users in the last year. As a core technology in KBase [36] (DOE Systems Biology Knowledgebase), a multi-institutional project aimed at advancing predictive biology in microbes, microbial communities and plants, our AWE/Shock ecosystem is used in production for various bioinformatics workflows.

The AWE/Shock data analysis system is open source and has been written in Go using the REST [9] architectural style. See Figure 1 for a schematic overview of the AWE/Shock architecture. All components support Simple Auth and OAuth. We provide a brief overview of the main components of Shock, the AWE server, and AWE worker.

1.5.1 Shock

The Shock data management system is designed as an object storage system. It is conceptually similar to the cloud storage system Amazon S3 (Simple Storage Service) [37]. The use of shared file systems to distribute data for processing between machines traditionally has been a common strategy in many compute clusters but is less suited for big data applications. Shared network file systems (typically

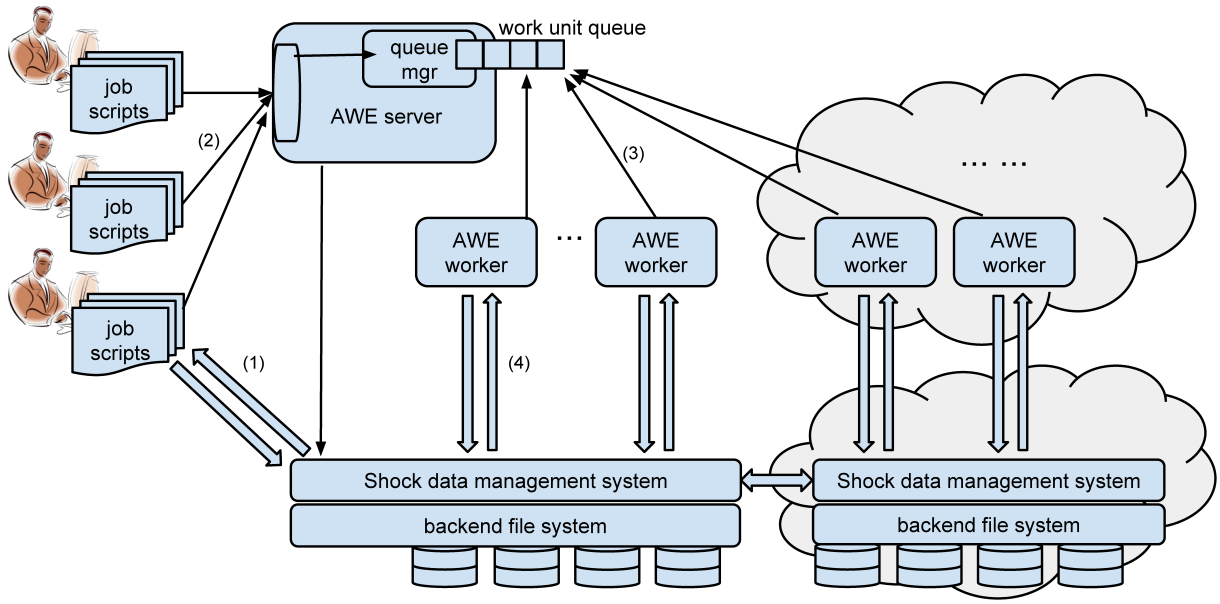

Figure 1: AWE/Shock architecture with distributed AWE workers. (1) Upload/download of user data. (2) Upload of AWE workflow document. (3) Work unit check out and check in. (4) Transfer of task input and output files between AWE worker and Shock server.

NFS) aim at providing the same POSIX semantics as a local file system, constraining the ability to harness various performance-oriented optimizations [22]. For example, the commonly used NFS is known to have limited scalability, which limits high-performance processing of mass data to clusters with a few dozen nodes [12]. While other (parallel) file systems are available (e.g., Lustre [38]) that scale well to larger numbers of clients, they are complex and difficult to maintain. However, since supporting POSIX semantics is usually not needed for web based applications and an object storage system such as Shock has the advantage of offering a direct integration of metadata storage (which includes metadata indexing for fast retrieval) as well as IO-efficient subsetting functionality, Shock better fits our architectural requirements.

Data in Shock is represented as an object with a unique identifier (the Shock node ID) and metadata describing computational and scientific provenance information. The Shock API can be used to store, query, and retrieve data and metadata. The query functionality enables the ability to search for objects with user-defined metadata entries. To efficiently support data-parallelism, data in Shock can be indexed in various ways, which makes it possible to retrieve user-defined data chunks for data processing. For example, the record-based index functionality enables Shock to support parallelism for formats relevant for biological sequence data such as FASTA, FASTQ and BAM.

Because all data connections to Shock are based on HTTP requests, connections from clients (i.e., AWE workers) to the Shock server are usually non-problematic and can be established between different administrative domains, allowing AWE to operate in multi-cloud mode (also known as "hybrid-cloud"). There are several ways remote AWE workers can be used to outsource resources to handle temporarily increased workloads, utilize more economical cloud resources, or make use of resources that fill specific hardware requirements (e.g., high RAM).

1.5.2 AWE server

The AWE server is a resource manager and job scheduler. The server takes a workflow description document as input (via Restful API) and creates smaller work units that can be checked out by AWE workers. The workflows are modeled as directed acyclic graphs to describe the data dependencies between individual tasks within the workflows. Each task represents either one work unit or, if data parallelism can be exploited, multiple work units. In the latter case, the AWE server creates work units for the task such that each work unit is assigned to one chunk of the data. Based on the workflow dependency graph, the AWE server automatically schedules individual work units that are checked out by AWE workers.

1.5.3 AWE worker

AWE workers check work units out from the server. A worker's configuration determines the work units it can check out. The configuration specifies which worker groups the AWE worker belongs to and includes the list of commands the worker can use to process data. When a work unit has been checked out, the AWE worker downloads all input files and required databases from Shock. Input files (including chunk files) are stored in a work directory on the backend/local file system on which the AWE worker is running. Databases that are commonly used are cached to avoid repetitive downloads. Once the work unit has been processed, result files are uploaded to Shock. Chunked results from data-parallel tasks are uploaded to Shock and merged back into a single data object. After all output files have been uploaded to Shock, the AWE worker deletes the work directory and tries to check out the next work unit.

2. SKYPORT

As discussed above, Linux containers provide an execution environment suitable for executing workflow tasks. By integrating Docker into our existing AWE/Shock system, our platform can automatically provision the runtime required to execute workflow steps.

The target use case for the Skyport extension to our platform is a *scientific workflow as a service* environment for the cloud that enables rapid creation of new workflow steps with a simple mechanism to reproduce the complete environment necessary to exactly reproduce the step in any other workflow. This solves a problem common to multiple scientific disciplines that rely upon computational workflows to store and process data (e.g., physics, biology/bioinformatics). Such workflows generally include multiple data processing tasks, each applying processes to one or more input files to produce one or more outputs. Our ecosystem is designed for multiple simultaneous users that execute multiple workflows consisting of reusable tasks.

Exploiting automatic deployment of Docker containers, our platform can generate a highly dynamic environment that could not easily be realized using virtual machines alone. Meta-data is generated by the system and stored together with data products in Shock. Together with Docker images, our ecosystem provides all relevant provenance information necessary to exactly reproduce workflows and their results. Our approach is a general one that can be applied to workflows used in multiple scientific disciplines.

To better exploit the advantages of Docker containers, we adapted Docker such that AWE workers are responsible for orchestrating the Docker application containers necessary to perform a workflow task on any given worker. In contrast to traditional installations that would require the AWE worker to be installed together with its software application dependencies, each application is isolated in a container that is independent with respect to the AWE worker.

An import aspect of the Docker integration in our ecosystem consists of storing the Docker images in Shock. Shock readily provides authentication and permission control (e.g., allows for public and private views) for all objects, making it possible to utilize Shock as a software repository. In addition, the lack of external dependencies (such as Docker Hub [39], or other hosted solutions) simplifies the architecture of our ecosystem, reduces potential I/O-bottlenecks, and also simplifies the deployment process because a user can use the same user credentials and upload mechanisms for Docker images and input data.

After creating Docker images (using Dockerfile or a container snapshot), images are uploaded to Shock. The Docker image identifier (`id`) and an image name (`name`) are stored together with the binary Docker images as metadata. The metadata attribute `type=dockerimage` unambiguously identifies the Shock object as a Docker image, simplifying Docker image management.

When the AWE worker checks out a new work unit, it automatically downloads input files and the Docker application image. Once the image is downloaded, the AWE worker loads the image into the Docker engine, creates and starts the container. Figure 2 depicts an activity diagram for the AWE worker with more details on individual steps.

In our model, an AWE worker runs in its own Docker container, but this is not a requirement. If the AWE worker runs in a container, the Docker UNIX socket that provides access to the Docker Remote API [40] has to be mounted to give the AWE worker the ability to manage Docker images and containers within the Docker engine. Because the work directory on the host is mounted into the work unit container, write-heavy loads will be performed directly on a volume of the host. Figure 3 shows how the AWE worker provides a software deployment mechanism using Docker.

All applications are isolated using Docker containers that the AWE worker automatically downloads and deploys. Theoretically, this means that each AWE worker can process work units from every possible task. Hardware requirements for processing individual work units (e.g., RAM requirements) can limit this ability. In our previous model without Docker, specific AWE workers ran on specialized VMs that could only execute a select set of tasks. This led to inefficient resource utilization; VMs would idle if they could not serve their workload. While theoretically it is possible to automatically provision VMs with respect to their workload demands, this solution can incur substantial overhead, and to the best of our knowledge, has not been implemented yet in the context of scientific workflows.

Deployment of Docker images to add novel workflow tasks does not require access to the cloud computing platform itself or the AWE workers, but does require upload of Docker images to Shock. Administrative control over software deployment can be achieved by Shock authentication and, optionally, by a central "app"-register that specifies the Docker images that can be checked out by a given AWE worker. The app-register can also specify the commands that are executable within each container.

With our Skyport-extension, Docker image applications, along with their supporting software and environment, can be modularly added to any workflow with great simplicity. In this way, our ecosystem replaces time consuming installation and configuration procedures with modular units. This modularity enables the creation of sophisticated workflows without specialized knowledge (e.g., of applications, their supporting software, and environments).

2.1 Test Example, MG-RAST Setup

To run the MG-RAST analysis pipeline, a worker machine must have all MG-RAST data processing scripts, custom wrapper scripts for third-party tools, and the required third-party tools installed. The AWE worker has to be deployed and configured on the worker machine so it can connect to an AWE server and check out MG-RAST tasks. The AWE worker machine can only execute MG-RAST tasks. If the MG-RAST machine is idle, its resources cannot be used to execute tasks from non MG-RAST workflows, potentially leading to an underutilization of system resources.

In order to exploit the advantages of our Skyport-enabled ecosystem, we created Docker containers for the MG-RAST

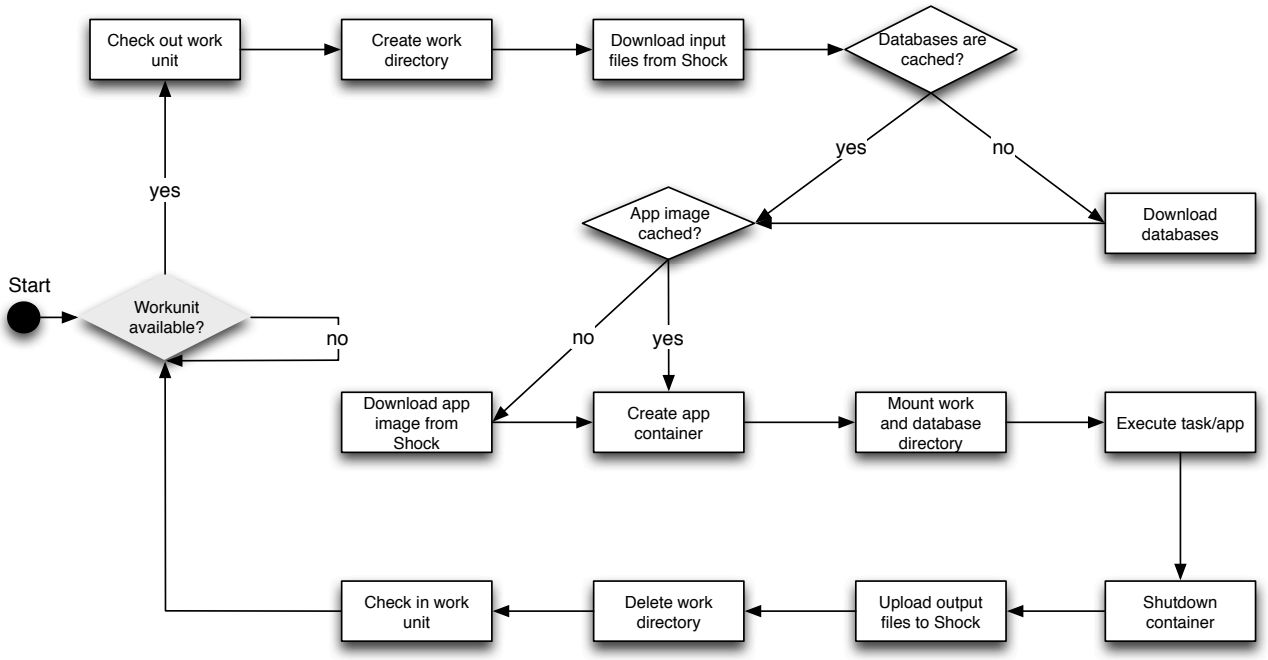

Figure 2: Activity diagram for the AWE worker

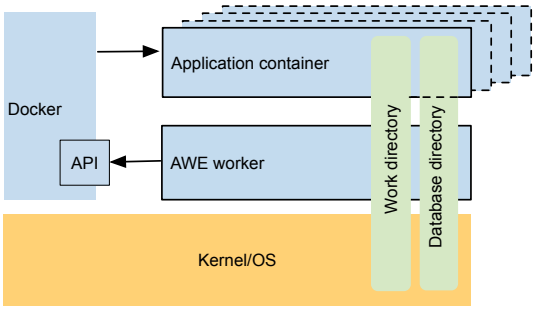

Figure 3: Architecture of the AWE worker using Docker. The AWE worker controls the application container via the Docker API. The database directory is mounted read-only in the application container.

repository:tag	base image	total	diff
ubuntu:14.04	–	213.0	213.0
mgrast/base:latest	ubuntu:14.04	586.9	373.9
mgrast/superblat:latest	mgrast/base:latest	591.2	4.3
mgrast/rna_search:latest	mgrast/base:latest	589.3	2.4
mgrast/qc:latest	mgrast/base:latest	739.6	152.7
mgrast/genecall:latest	mgrast/base:latest	643.6	56.7
mgrast/cluster:latest	mgrast/base:latest	587.8	0.9
mgrast/bowtie:latest	mgrast/base:latest	625.8	38.9
mgrast/blat:latest	mgrast/base:latest	596.7	9.8

Table 1: Size of Docker images for the MG-RAST workflow. "total" specifies the size in megabytes of each image including it's base images as reported by the Docker engine. "diff" is the images size excluding base images.

workflow. For each task that executes third-party tools (8 of 20) we created a separate Docker image, using the MG-RAST image as its base. This made it possible to isolate third-party tools from each other. In one exception(quality control, `mgrast/qc:latest`), multiple third-party tools were included in a single image, isolating them from all other third-party tools, but not from each other. Table 1 lists all images, including the `ubuntu:14.04` and `mgrast/base` base images. The table shows that those images that extend our MG-RAST base image add only a few megabytes and thus do not introduce much overhead in terms of disk usage on the AWE worker machines.

The time to deploy images by the AWE worker is mainly dominated by loading into the Docker engine. For example, loading the MG-RAST base image `mgrast/base:latest` takes about 20 seconds. This one-time cost per AWE worker is negligible compared to the combined runtime of a task on a machine using that image over days or weeks. This report is based on preliminary study. Our first tests with small datasets indicate performance similar to our production pipeline on AWE/Shock without Skyport. While these results are very promising, we are currently performing more extensive tests to evaluate the system.

In our experiment, we did not isolate third-party tools that were used within the same task as this would have required more significant changes to the MG-RAST code base. The purpose of the isolation was to show the conceptual feasibility of the Skyport approach. It also demonstrated that for existing production systems such as MG-RAST, the decision to isolate parts of the system can be based on very

pragmatic considerations (e.g. isolating third party tools by task) and does not necessarily require a complete re-design of existing software architectures.

3. DISCUSSION

Theoretically, an ecosystem such as ours could have been realized with established hardware virtualization technologies (e.g., with specialized VMs). However, the lightweight nature of Linux container virtualization compared to VMs constitutes a general paradigm shift for software development and deployment. We have shown how the Skyport-extension to our workflow platform utilizes the container technology to solve the software deployment problem inherent to all existing scientific workflow platforms. The systematic use of isolated execution environments for workflow tasks offers a convenient and simple mechanism to reproduce scientific results. The AWE workflow document, associated Docker images, and the original input files (e.g., stored in Shock) are all that is necessary to reproduce the results of any workflow.

The automated deployment of applications by the AWE worker has multiple advantages. In addition to providing an overall deployment process that is much simpler than current alternatives, the fast and dynamic deployment of containers can improve overall resource utilization. Fluctuating demand for certain types of tasks that each may require independent execution environments may lead to idle cycles. This is because a mismatch may occur between the tasks a worker is capable of performing and the tasks that are waiting in the queue.

In our current design, an AWE worker can run only one containerized process at a time. Using Linux kernel cgroups to limit memory and CPU consumption of containers would allow AWE workers to run multiple containers on the same host. This would make it possible to exploit performance gains by avoiding the use of VMs. In addition, one could run the Docker containers on top of a much more lightweight operating system such as CoreOS [41] or Boot2Docker [42]. While cgroups technology would increase flexibility regarding resource management, it would also require more resource aware scheduling of the AWE server to dynamically adapt resource limits and reduce the amount of idle resources.

With the dynamic provisioning of applications in our ecosystem using the Skyport-extension, all compute resources with AWE workers installed can execute any workflow task. Using Docker containers also provides additional flexibility to automatically scale the AWE workforce up and down using existing Docker orchestration tools such as Marathon [43], Kubernetes [44], CoreOs fleet [45] or Flynn [46].

The Skyport-extension of the AWE/Shock data analysis platform makes our ecosystem the first scientific workflow management system that uses Docker Linux containers for automated deployment of software applications together with their own isolated execution environments. Skyport's use of Linux containers greatly reduces the resources and expertise required to build scientific workflows, making it possible for non-computer science experts to integrate their own and third-party applications into new or existing workflows in the AWE/Shock data analysis ecosystem.

4. ACKNOWLEDGMENTS

This work was supported in part by the NIH award U01HG006537 "OSDF: Support infrastructure for NextGen sequence storage, analysis, and management", and U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research DE-AC02-06CH11357 as part of Resource Aware Intelligent Network Services (RAINS) and as part the Office of Science, Office of Biological and Environmental Research Systems Biology Knowledgebase (KBase).

5. REFERENCES

[1] M. Abouelhoda, S. A. Issa, and M. Ghanem. Tavaxy: integrating taverna and galaxy workflows with cloud computing support. *BMC Bioinformatics*, 13:77, 2012.

[2] E. Afgan, D. Baker, N. Coraor, B. Chapman, A. Nekrutenko, and J. Taylor. Galaxy cloudman: delivering cloud compute clusters. *BMC bioinformatics*, 11(Suppl 12):S4, 2010.

[3] I. Altintas, J. Wang, D. Crawl, and W. Li. Challenges and approaches for distributed workflow-driven analysis of large-scale biological data. In *Proceedings of the Workshop on Data analytics in the Cloud at EDBT/ICDT 2012 Conference, DanaC2012*, 2012.

[4] M. Bux and U. Leser. Parallelization in scientific workflow management systems. *arXiv preprint arXiv:1303.7195*, 2013.

[5] J. G. Caporaso, J. Kuczynski, J. Stombaugh, K. Bittinger, F. D. Bushman, E. K. Costello, N. Fierer, A. G. Pena, J. K. Goodrich, J. I. Gordon, et al. QIIME allows analysis of high-throughput community sequencing data. *Nature Methods*, 7(5):335–336, 2010.

[6] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[7] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and linux containers. *technology*, 28:32, 2014.

[8] D. Field, B. Tiwari, T. Booth, S. Houten, D. Swan, N. Bertrand, and M. Thurston. Open software for biologists: from famine to feast. *Nature biotechnology*, 24(7):801–804, 2006.

[9] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[10] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In *Network and parallel computing*, pages 2–13. Springer, 2005.

[11] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec. Flexible and efficient workflow deployment of data-intensive applications on grids with moteur. *International Journal of High Performance Computing Applications*, 22(3):347–360, 2008.

[12] T. Kim and S. H. Noh. pnfs for everyone: An empirical study of a low-cost, highly scalable networked storage. *IJCSNS*, 14(3):52, 2014.

[13] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the linux virtual machine monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, 2007.

[14] K. Krampis, T. Booth, B. Chapman, B. Tiwari, M. Bicak, D. Field, and K. E. Nelson. Cloud biolinux: pre-configured and on-demand bioinformatics computing for the genomics community. *BMC bioinformatics*, 13(1):42, 2012.

[15] M. Lageman. Solaris containers–what they are and how to use them. *Sun BluePrints OnLine*, pages 819–2679, 2005.

[16] R. K. Madduri, P. Dave, D. Sulakhe, L. Lacinski, B. Liu, and I. T. Foster. Experiences in building a next-generation sequencing analysis service using galaxy, globus online and amazon web service. In *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery*, page 34. ACM, 2013.

[17] F. Meyer, D. Paarmann, M. D'Souza, R. Olson, E. M. Glass, M. Kubal, T. Paczian, A. Rodriguez, R. Stevens, A. Wilke, J. Wilkening, and R. A. Edwards. The metagenomics RAST server - a public resource for the automatic phylogenetic and functional analysis of metagenomes. *BMC Bioinformatics*, 9:386, 2008.

[18] W. Tang, J. Wilkening, N. Desai, W. Gerlach, A. Wilke, and F. Meyer. A scalable data analysis platform for metagenomics. In *2013 IEEE International Conference on Big Data*, pages 21–26. IEEE, 2013.

[19] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The condor experience. *Concurrency and Computation: Practice and Experience*, 17(2-4):323–356, 2005.

[20] T. Thomas, J. Gilbert, and F. Meyer. Metagenomics - a guide from sampling to data analysis. *Microb Inform Exp*, 2(1):3, 2012.

[21] P. J. Turnbaugh, R. E. Ley, M. Hamady, C. M. Fraser-Liggett, R. Knight, and J. I. Gordon. The human microbiome project. *Nature*, 449(7164):804–810, Oct 2007.

[22] E. Vairavanathan, S. Al-Kiswany, L. B. Costa, Z. Zhang, D. S. Katz, M. Wilde, and M. Ripeanu. A workflow-aware storage system: An opportunity study. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 326–334. IEEE Computer Society, 2012.

[23] VMWare Staff. Virtualization Overview. *White Paper*.

[24] J. Wang, D. Crawl, and I. Altintas. Kepler+ hadoop: a general architecture facilitating data-intensive applications in scientific workflow systems. In *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, page 12. ACM, 2009.

[25] J. Wang, D. Crawl, and I. Altintas. A framework for distributed data-parallel execution in the kepler scientific workflow system. *Procedia Computer Science*, 9:1620–1629, Jan 2012.

[26] A. Wilke, E. M. Glass, D. Bartels, J. Bischof, D. Braithwaite, M. D'Souza, W. Gerlach, T. Harrison, K. Keegan, H. Matthews, et al. A metagenomics portal for a democratized sequencing world. *Methods Enzymol*, 531:487–523, 2013.

[27] `http://home.web.cern.ch/`

[28] `http://www.xenproject.org`

[29] `http://www.microsoft.com/hyper-v`

[30] `http://www.openstack.org`

[31] `http://aws.amazon.com/ec2/`

[32] `http://linuxcontainers.org`

[33] `http://openvz.org`

[34] `http://docker.com`

[35] advanced multi layered unification filesystem `http://aufs.sourceforge.net`

[36] `http://kbase.us`

[37] `http://aws.amazon.com/s3/`

[38] `http://lustre.opensfs.org/`

[39] `https://hub.docker.com/`

[40] `https://docs.docker.com/reference/api/docker_remote_api_v1.13/`

[41] `http://coreos.com`

[42] `http://boot2docker.io`

[43] `https://github.com/mesosphere/marathon`

[44] `https://github.com/GoogleCloudPlatform/kubernetes`

[45] `https://github.com/coreos/fleet`

[46] `https://flynn.io`