ISLET: An Isolated, Scalable, & Lightweight Environment for Training

Jonathan Schipp
NCSA
1205 W. Clark St.
Urbana, IL 61801
(217) 244-3583
jschipp@illinois.edu

Jeannette Dopheide
NCSA
1205 W. Clark St.
Urbana, IL 61801
(217) 244-7441
jdopheid@illinois.edu

Adam Slagell
NCSA
1205 W. Clark St.
Urbana, IL 61801
(217) 244-8965
slagell@illinois.edu

ABSTRACT

In this paper we present ISLET; the Isolated, Scalable, & Lightweight Environment for Training. ISLET overcomes many of the distribution, scaling and security challenges of providing mass training to students requiring an interactive GNU/Linux command-line environment. This Docker-based solution is evaluated and lessons learned from real world experience with ISLET are discussed.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Information Systems Education

Keywords

Security, Containers, Training, Education, Docker

1. INTRODUCTION

Virtual machines (VMs) are commonly used for software training that relies upon isolated environments, especially when command line access is required. However, the limitations of distributing VMs to every student can be significant and is exacerbated in large training settings. The images are often multiple gigabytes in size, which can be challenging to distribute before a training event begins. Furthermore, students are required to have a sufficiently powerful laptop with a VM hypervisor installed before training. This can lead to disaster if students do not have all the files downloaded and installed beforehand, crippling the host organization's wireless network infrastructure. This further implies that last minute changes to the training environment are difficult to impossible to execute, should problems be discovered before or during the training.

Such issues with VM configuration, training environments, and networking all waste time that is supposed to be dedicated to learning. The Isolated, Scalable, & Lightweight Environment for Training, or ISLET, addresses these limitations of local VM-based training by providing an alternative method to connect and interact with the learning environment.

ISLET is a framework upon which an isolated, customized GNU/Linux command line training environment is presented. Users connect to the trainer's ISLET server using a standard SSH client. ISLET does not require the trainer to distribute large files; last minute updates to the training environment can be pushed out quickly and with little interruption to the user experience; a user may become disconnected from their environment but is able to reattach for the duration of the event; account creation is simple and less burdensome on the trainer and trainee; and ISLET can be scaled to support hundreds of users with a single server, and even more by hosting the ISLET server on an expandable cloud-based platform. Finally, ISLET follows the best practices for securing its containers by disabling root access, limiting networking, and cleaning up accounts and environments after a training session has ended. All these features may be configured based on the needs of the instructor.

In section 2 we present ISLET's implementation and architecture. The performance and scalability of ISLET is evaluated in section 3 as well as some lessons learned from its real-world use. Section 4 compares ISLET to other technical training solutions. We discuss future work and conclude in sections 5 and 6, respectively.

XSEDE '15, July 26 - 30, 2015, St. Louis, MO, USA
© 2015 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-3720-5/15/07...$15.00
DOI: http://dx.doi.org/10.1145/2792745.2792762

2. IMPLEMENTATION

ISLET utilizes Linux-based containers [15] [2] to deploy command-line training environments with low overhead, isolation, and reproducibility. The building blocks of modern Linux containers have been available since Linux kernel 3.8, namely in cgroups and namespaces. Control Groups (cgroups) are a subsystem, which provide the application of resource constraints, or management, to process groups. Namespaces provide resource isolation, effectively making a system resource believe it's a part of a separate global resource through abstraction. Together, cgroups and namespaces are the foundation for Linux containers and provide what is sometimes called lightweight process virtualization, or operating system virtualization. This is in contrast to hardware virtualization used in virtual machine (VM) technology. Containers reduce overhead by sharing the same kernel, though without as strict an isolation as with virtual machines.

Container density is much greater than virtual machine density, making ISLET scalable for an ever growing number of students. It's not unheard of to be able to run several hundred containers or more simultaneously on a single host that scales vertically with hardware, which we demonstrate in Figure 6 in section 3. Furthermore, the addition of high-availability and clustering tools for containers, such as CoreOS [5] or Docker's Swarm [4], means that

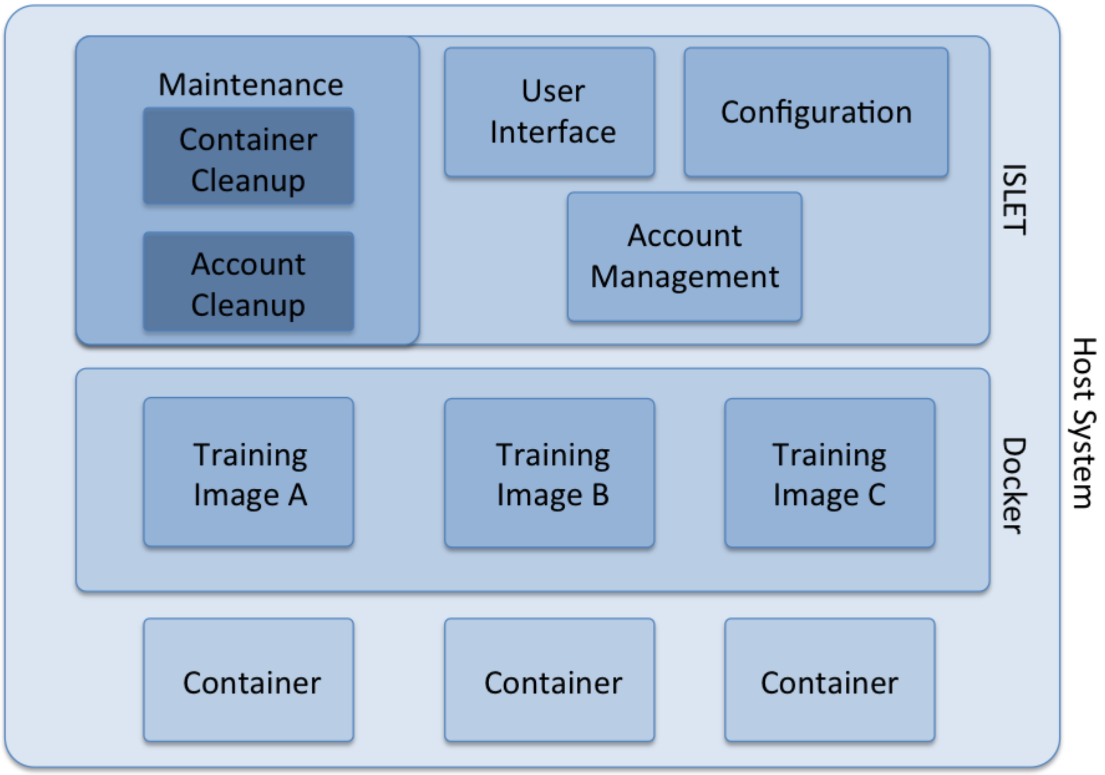


Figure 1: Diagram of ISLET's architecture

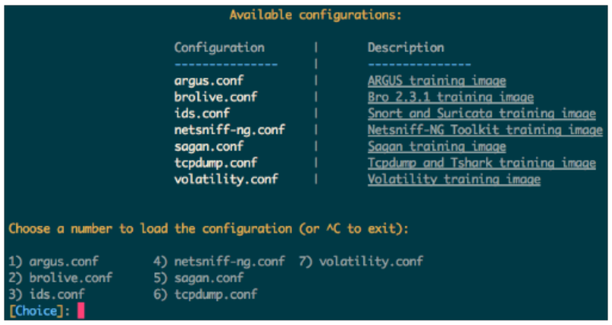
scaling horizontally is a possibility; enabling the potential to teach Massive Open Online Courses (MOOCs).

Figure 1 shows the relationship between Docker, the physical machine, and the additional components of ISLET that provide its services. ISLET uses the popular Docker implementation of a container runtime engine to provide isolation between users and rapid deployment. The Docker images are used for different customized training environments. The scripts and tools that make up ISLET provide account maintenance and provisioning, container state cleanup, user interfaces, and deployment/management tools.

The ISLET user experience gives the appearance of one's own shell on an isolated machine. By using Docker, logins are almost immediate, whereas a VM could take a few minutes to launch. As we show in Figure 7 in section 3, users typically wait less than a second for their new container to launch. This is much quicker than launching a local VM image. Also, ISLET doesn't require the VM hypervisor to be installed on their system, which can be problematic for low-powered devices and tablets.

Customizable configuration files read by ISLET allow the administrator to pass options to the container runtime tool for each training environment. Therefore, certain features, like creating multiple variations of the same image (e.g., images with different security constraints), are applied simply by using a new configuration file. Updates to the image can be made at the last minute without needing to push to all the users by taking advantage of volume mounts from the host or launching a new container. By comparison, virtual machines do not grant the same flexibility if a mistake is made, such as copying the wrong exercises onto the VM before distribution.

The ISLET user interface (Figure 2) allows users to choose an available configuration from a selection menu after they log into their ISLET account by remote connection to the ISLET system via SSH, or an alternative remote access program. Their configuration choice, typically led by an instructor, determines which training environment to run and the options it will have. All the instructor must do is (i) create a Docker image for the environment from a Dockerfile, (ii) create a new ISLET configuration file with settings for the image based on a provided template, and (iii) place the configuration file in the proper directory on an ISLET server. To begin training, the instructor distributes the server address, username, and an optional password for the server which is used to remotely connect to the system. The user follows the prompts to create their own ISLET account after logging into the system. The account is the user's responsibility and remains active for the duration of the training event. This is much easier than delivering large VM images before the event, or even worse, during the event.



```
                          Available configurations:

                 Configuration        |      Description
                 ---------------      |      ---------------
                 argus.conf           |      ARGUS training image
                 brolive.conf         |      Bro 2.3.1 training image
                 ids.conf             |      Snort and Suricata training image
                 netsniff-ng.conf     |      Netsniff-NG Toolkit training image
                 sagan.conf           |      Sagan training image
                 tcpdump.conf         |      Tcpdump and Tshark training image
                 volatility.conf      |      Volatility training image


Choose a number to load the configuration (or ^C to exit):

1) argus.conf         4) netsniff-ng.conf  7) volatility.conf
2) brolive.conf       5) sagan.conf
3) ids.conf           6) tcpdump.conf
[Choice]:
```

Figure 2: ISLET's main menu

ISLET doesn't depend on the use of full UNIX system accounts. Instead it manages them through an internal account management system. Instructors aren't burdened with setting up individual accounts ahead of time. Account information is saved if a user is disconnected from the ISLET server.

ISLET has many features to mitigate the risk of account abuse. The ISLET server can use any of the authentication mechanisms provided by OpenSSH or other compatible remote access services. For example, it can require a password or key pair to access the system before creating a new ISLET account. Account access is limited to the isolated container. And additional restrictions, like disabling access to network devices, can prevent students from using their account as a proxy for attacking other systems.

3. EVALUATION

3.1 Performance

We evaluated Docker to determine how well it scales and performs for the purpose of training. We put Docker through a series of tests relevant to its intended use by ISLET. First, it is essential to know how well ISLET can handle a given number of users for educational settings. Second, we compare the cost of ISLET as an alternative to virtual machine training hosted by a cloud provider. Third, we validate a startup time claim often seen online without substantiation. Finally, we discuss existing research on container performance. The result of these experiments support ISLET's effectiveness as a competitive GNU/Linux software training solution for events and educational institutions.

The software configuration for our tests include use of the GNU/Linux distribution Ubuntu 14.04.1 LTS (amd64), Linux kernel version 3.13.0-44, a dynamically linked version of Docker 1.5.0-dev [9], and the devicemapper storage driver. Docker's devicemapper storage driver currently has a known race condition bug [8] that is mitigated by building Docker dynamically; a required step for completing our tests. The hardware configuration consists of an Amazon EC2 virtual machine of type c4.4xlarge with the following resources: 16vCPUs, 30GiB RAM, and 30GB SSD.

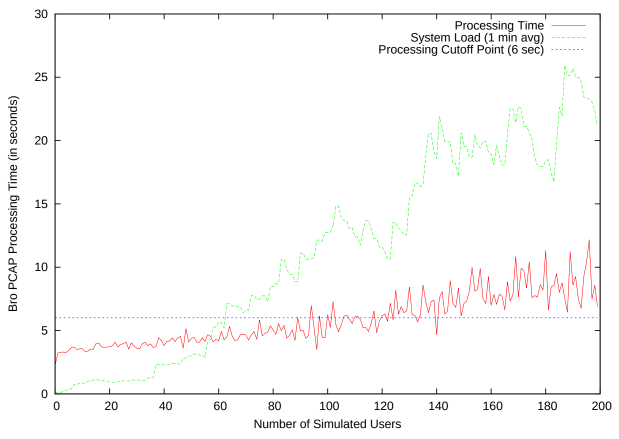


Figure 3: User Simulation
Amazon EC2 c4.4xlarge (16 vCPUs, 30GiB RAM)

Our test (See Figure 3) simulated the workload of a set of very active users learning Bro [14] (a network security monitoring tool). A small amount of randomness (less than 10 seconds) was introduced to delay the execution of commands per user. In a real training environment not every user will execute the same command at the same time. A shell script was executed in each user's container, which included common commands for maneuvering within the filesystem, executing programs, and checking system information. The script ran 20 times per container with different delays per command for each loop, which simulated more than 10 minutes' worth of user activity. Our simulation stressed a higher workload by not accounting for lecture time from the instructor where listening, rather than working, is required.

```
--------------------------------------------------------------------------------
Command:            bro -r /exercises/ssl/exercise-traffic.pcap
Massif arguments:   (none)
ms_print arguments: massif.out.248
--------------------------------------------------------------------------------


    MB
81.57^                                                                       #
     |                                                                    @@:#
     |                                                         :@::::@:::@@@:#
     |                                                     :@@@@@::::@:::@@@:#
     |                                               @@:@@:@@@@@@::::@:::@@@:#
     |                             @:::@::@::::::::::@@:@@:@@@@@@::::@:::@@@:#
     |                             @:::@: @::::::: ::@@:@@:@@@@@@::::@:::@@@:#
     |                             @:::@: @::::::: ::@@:@@:@@@@@@::::@:::@@@:#
     |                            @@:::@: @::::::: ::@@:@@:@@@@@@::::@:::@@@:#
     |                            @@:::@: @::::::: ::@@:@@:@@@@@@::::@:::@@@:#
     |                         ::@@@:::@: @::::::: ::@@:@@:@@@@@@::::@:::@@@:#
     |                        @::@@@:::@: @::::::: ::@@:@@:@@@@@@::::@:::@@@:#
     |                        @::@@@:::@: @::::::: ::@@:@@:@@@@@@::::@:::@@@:#
     |                        @::@@@:::@: @::::::: ::@@:@@:@@@@@@::::@:::@@@:#
     |                       @@::@@@:::@: @::::::: ::@@:@@:@@@@@@::::@:::@@@:#
     |                       @@::@@@:::@: @::::::: ::@@:@@:@@@@@@::::@:::@@@:#
     |            :@@:@@::::@@@::@@@:::@: @::::::: ::@@:@@:@@@@@@::::@:::@@@:#
     |        @@@::@@:@ : : @@@::@@@:::@: @::::::: ::@@:@@:@@@@@@::::@:::@@@:#
     |      @@@ @::@@:@ : : @@@::@@@:::@: @::::::: ::@@:@@:@@@@@@::::@:::@@@:#
     |    :@@@@ @::@@:@ : : @@@::@@@:::@: @::::::: ::@@:@@:@@@@@@::::@:::@@@:#
   0 +----------------------------------------------------------------------->Gi
     0                                                                   1.922
```

Figure 4: Bro Valgrind Memory Profile

Of course, system load is not the whole story, and what we ultimately care about is responsiveness for the user. Using the same simulation script above, we determined when to move from a small VM to a larger host to support more containers. We did this by simulating $n - 1$ users, and measured the impact on a particular task for the n^{th} user. In this case, we decided the processing time was too high if it took Bro three times as long to process a specific network trace, from 2 to 6 seconds. We refer to this as the *cut-off point*. The average time to run Bro through a popular exercise trace EXERCISE-TRAFFIC.PCAP in a container on our Amazon EC2 c4.4xlarge host was 2.13 seconds. This number is the average time based on 100 runs through the pcap file. Also, note that Bro as a process is not lightweight, but rather resource consuming; it peaks at about 80MiB when running through the pcap as shown in Figure 4. Using the four Amazon server configurations described in 1, we calculated the cutoff points as a function of the number of users, as seen in Figure 5.

Machine	Description	Containers	Cost (hr.)
EC2 t2.small	1vCPUs, 2GiB	11	\$0.026
EC2 t2.medium	2vCPUs, 4GiB	34	\$0.052
EC2 c4.xlarge	4 vCPUs, 7 GiB	63	\$0.232
EC2 c4.2xlarge	8 vCPUs, 15 GiB	147	\$0.464

Table 1: Cost breakdown of Amazon AWS using ISLET

Efficiency is difficult to measure, but cost is a proxy for efficiency. VMs have a cost, even if it is externalized to the students running VMs on all of their laptops and not borne by the instructor or institution directly. To measure efficiency, we compare the costs of running a single VM per student in Amazon to running a single ISLET server for all users. Table 1 shows the per hour costs of four

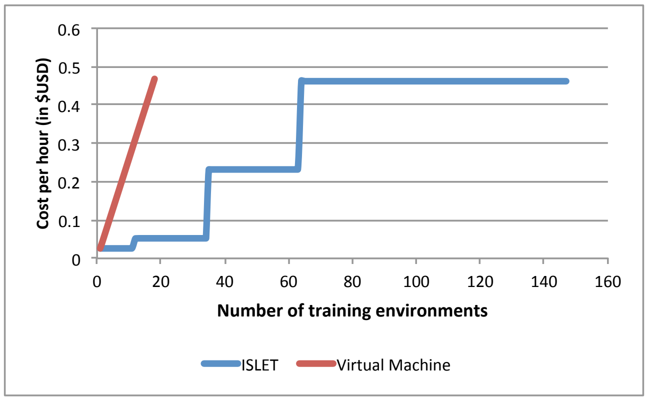


Figure 5: Cost Comparison of ISLET and Virtual Machine

different Amazon servers, with the number of containers supported determined by our Bro runtime metric. Figure 5 plots these costs as a function of the number of students, where we see how much more efficient ISLET is to giving each person their own VM.

In Figure 6 we show how many containers can run concurrently without affecting system load in a significant way. We use system load as the measurement because time spent waiting on I/O operations are factored into the load value in addition to the number of processes in the run queue. Because of the layered filesystems and copy-on-write features of Docker, there is little I/O wait among processes when creating many containers, each with its own filesystem. These features also help explain the fast startup times discussed next.

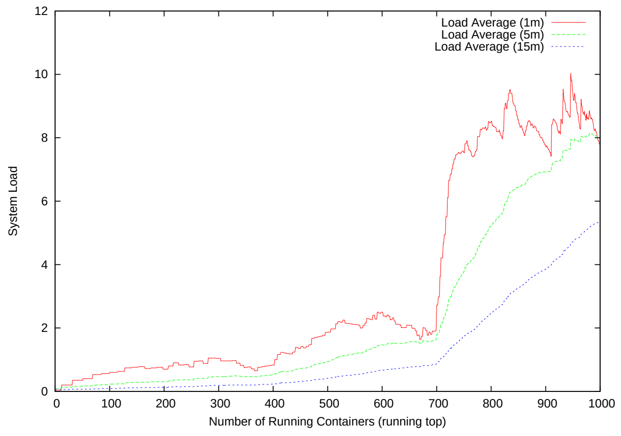


Figure 6: Container Density
Amazon EC2 c4.4xlarge (16 vCPUs, 30GiB RAM)

Fast startup times are one of the major reasons for choosing containers as the foundation for ISLET. Their nearly instantaneous startup times, especially when running small but interactive processes like shells, allow users to begin work immediately. This is more productive and ensures a smoother training experience. Online you will find it often claimed that container startup time is milliseconds, not seconds like VMs. CoreOS's website claims, "Containers can boot extremely fast (in milliseconds!)" [6]. We do not doubt this claim based on our experience with Docker, but we have not yet seen it substantiated online or in publication. We verified this claim in Figure 7, which demonstrates how quickly we are able to get users up and running with a command-line environment for training.

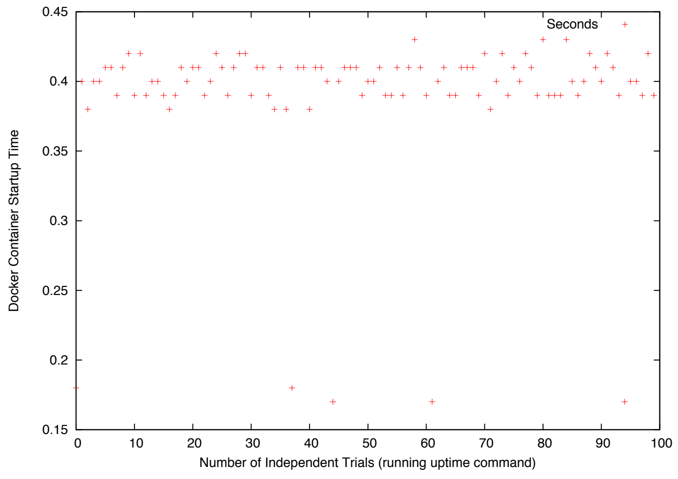


Figure 7: Docker Container Startup Time
Amazon EC2 c4.4xlarge (16 vCPUs, 30GiB RAM)

All of this aligns with general consensus in the literature that containers perform better than virtual machines in most cases. One would expect that containers can reach near bare metal performance because they share the same kernel and do not have to virtualize hardware. Research by Xavier et al. and Felter et al. demonstrates many performance advantages of containers. [19] [20]

3.2 Lessons Learned

Many lessons were learned when developing ISLET and using it at the University of Illinois and *BroCon '14* [16]. ISLET's isolated design ensures that a user cannot easily break into other containers or the host server. This is good for security but it can have a negative impact on the user experience if expected functionality is disabled or not available. Therefore, we recommend administrators familiarize themselves with the various options available for the training environments. Careful consideration should be paid to the need for root access, networking, and default package installation. ISLET has an install option that places resource constraints on Docker processes to prevent fork bombs and other Denial of Service attacks. Furthermore, administrators should consider limiting sudo to specific commands or use Linux kernel capabilities where root privileges are required.

Administrators may also want to utilize symlinks to make it easier for the user to navigate between the software program and supplementary content included in the environment. For example, it is good to have shared read-only materials, like large trace files, all in one place rather than multiple copies in the users' home directories.

Finally, if the ISLET server is hosted by a cloud service, the administrator needs to remember to disable ISLET when the training session has concluded to limit operational costs. This may seem obvious, but even if the resource is turned off it can still incur significant costs for storing all the docker images.

3.2.1 Setting Capabilities

A particular challenge arises when training software must run as root. By default, users are not given root access in ISLET containers, though this challenge can be accommodated by setting Linux kernel file capabilities [12] (e.g., cap_net_raw, cap_net_admin) on the software binary of choice. This allows users to run a program with a specific set of root privileges, rather than all of them as is the case when the user becomes root. Docker drops most capabilities by default as a security precaution, and ISLET drops even more, but they can be selectively added to a container when required by modifying ISLET's configuration files. Also, note that the AUFS Docker storage backend does not support filesystem capabilities. The default setting of ISLET is to use devicemapper as the storage backend which supports file system capabilities.

4. RELATED WORK

While there are many kinds of hands-on, web-based training environments for a variety of software tools, they don't meet the needs of courses that require a full UNIX-like command line environment [1]. Some of these share similarities to ISLET, but their differences further underscore ISLET's unique position among the training tools used in information technology. For example, Try.bro.org [17] is a web-based scripting sandbox that uses Docker to run Bro in an isolated container. Like ISLET, Try.bro does not require the user to install additional software; users access Try.bro in a web browser or their computer terminal. Unlike ISLET, Try.bro is not a free-standing installation of Bro. Its purpose is for testing Bro's scripting language, which limits its utility as a general training platform. Also, Try.bro has been developed for a single software program, whereas ISLET is more software agnostic.

Traditionally, training that requires a full command-line interface has been served either by distributing VMs to be run locally by students or giving them accounts on a shared UNIX-like system. As we note in the introduction, VMs require better hardware, need special software installed, and come with a host of distribution challenges. Shared systems require more administrative setup with real accounts that must be created, they don't provide the isolation and security properties that containers can easily supply, and often they will not scale as elastically; though platforms like Amazon's Elastic Compute Cloud (Amazon EC2) [3] do make that easier to do today.

5. FUTURE WORK

Based on feedback from live Bro training events and instructors using ISLET at the University of Illinois, we identified a roadmap for future development. Currently the trainees' work in ISLET is deleted after the event occurs. We intend to create a mechanism to export a student's work so they can access it later and reinforce what they learned. While ISLET is fairly easy for instructors to setup, it is assumed that they have administrative access on the ISLET servers. This could be made even easier by creating deployment tools to simplify configuration and setup for instructors. Imagine a script or toolset that asks an instructor a few questions and deploys an environment in their favorite cloud provider with local management tools on their desktop. While one strength of ISLET is the hands-off nature of account management, there are situations where it might be desirable to use federated identities and support something like OpenID Authentication [7]. Another area for improvement is to enable ISLET to use more operating system virtualization tools besides Docker, like LXC (Linux Containers) [11] or FreeBSD jails [18]. Finally, it would be nice to run a set of ISLET servers as a cluster that is integrated and load-balanced for large events when an external cloud provider is not preferred. Integration with OpenStack [13] is therefore one of the options we are exploring with the development of deployment tools.

6. CONCLUSION

We believe ISLET is a powerful solution for conference training events, classrooms, meetings, and other forms of organized learning where demonstrations or hands-on application is desirable or mandatory. ISLET overcomes many of the logistical and scalability challenges of providing interactive, command-line training environments. It does this while putting minimal technical requirements on student systems and providing secure, isolated workspaces. It is simple to deploy and setup for instructors, saving them time and effort as well.

ISLET was originally created to solve challenges of providing training at the annual BroCon event, but since has been generalized and used for training in courses at the University of Illinois, FlowCon, the Open Network Security Monitoring Group (OpenNSM), UIUC GNU/Linux User Group, and is now being investigated by other organizations. Download it and try it yourself at our GitHub repository [10].

7. ACKNOWLEDGMENTS

We would like to thank colleagues and friends who have used, promoted, and given feedback on this project; these include but are not limited to Wayland Morgan, Warren Raquel, Justin Azoff, Liam Randall, Imani Palmer, and Tyler Manson.

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No.ACI-1348077. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] Comparable training environment solutions include: A Tour Go https://tour.golang.org/welcome/1, O'Reilly School of Technology's Learning Sandbox http://www.oreillyschool.com/whyost/learn-by-making/, Docker's Web Tour https://www.docker.com/tryit/.

[2] Linux* Containers Streamline Virtualization and Complement Hypervisor-Based Virtual Machines. http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/linux-containers-hypervisor-based-vms-paper.pdf, 2014. [Online; accessed: 11-February-2015].

[3] Amazon Web Services, Inc. Amazon EC2. http://aws.amazon.com/ec2/, 2015. [Online; accessed: 24-February-2015].

[4] Andrea Luzzardi and Victor Vieux. Swarm: a Docker-native clustering system. https://github.com/docker/swarm, 2015. [Online; accessed: 11-February-2015].

[5] CoreOS, Inc. https://coreos.com/, 2015. [Online; accessed: 11-February-2015].

[6] CoreOS, Inc. Container Overview. https://coreos.com/using-coreos/containers, 2015. [Online; accessed: 05-March-2015].

[7] David Recordon and Brad Fitzpatrick. OpenID Authentication 1.1. http://openid.net/specs/openid-authentication-1_1.html, 2015. [Online; accessed: 16-February-2015].

[8] Github, Inc. Docker fails to mount the block device for the container on devicemapper. https://github.com/docker/docker/issues/4036, 2015. [Online; accessed: 18-March-2015].

[9] Jon Schipp. Build a Dynamically Linked Docker. http://sickbits.net/build-a-dynamically-linked-docker/, 2015. [Online; accessed: 18-March-2015].

[10] Jon Schipp. Isolated, Scalable, & Lightweight Environment for Training. https://github.com/jonschipp/ISLET, 2015. [Online; accessed: 24-February-2015].

[11] LinuxContainers.org. Home. https://linuxcontainers.org/, 2015. [Online; accessed: 05-March-2015].

[12] Michael Kerrisk. Linux Programmer's Manual. http://man7.org/linux/man-pages/man7/capabilities.7.html, 2015. [Online; accessed: 25-February-2015].

[13] Open Stack Foundation. OpenStack: The Open Source Cloud Operating System. http://www.openstack.org/software/, 2015. [Online; accessed: 24-February-2015].

[14] V. Paxson. Bro: a System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[15] Rami Rosen. Linux Containers and the Future Cloud. http://media.wix.com/ugd/295986_d5059f95a78e451db5de3d54f711e45d.pdf. [Online; accessed: 11-February-2015].

[16] The Bro Project. BroCon '14. https://www.bro.org/community/brocon2014.html, 2014. [Online; accessed: 17-March-2015].

[17] The Bro Project. Try Bro. http://try.bro.org/, 2014. [Online; accessed: 17-March-2015].

[18] The FreeBSD Documentation Project. FreeBSD Handbook. https://www.freebsd.org/doc/handbook/jails.html, 2015. [Online; accessed: 05-March-2015].

[19] Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio. An Updated Performance Comparison of Virtual Machines and Linux Containers. http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf, 2014. [Online; accessed: 06-March-2015].

[20] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. De Rose. Performance evaluation of container-based virtualization for high performance computing environments. In *Proceedings of the 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, PDP '13, pages 233–240, Washington, DC, USA, 2013. IEEE Computer Society.