

published as: Thomas Staubitz, Maximilian Brehm , Johannes Jasper, Thomas
Werkmeister, Ralf Teusner, Christian Willems, Jan Renz, and Christoph Meinel:
Vagrant Virtual Machines for Hands-On Exercises in Massive Open Online Courses,
In Proceedings of Smart Education and e-Learning 2016 (KES-SEEL2016), 15-17
June 2016, Puerto de la Cruz

Vagrant Virtual Machines for Hands-On Exercises in Massive Open Online Courses

Thomas Staubitz¹, Maximilian Brehm², Johannes Jasper², Thomas Werkmeister², Ralf Teusner¹, Christian Willems¹, Jan Renz¹, and Christoph Meinel¹

Hasso Plattner Institute, Potsdam, Germany

¹firstname.lastname@hpi.de

²firstname.lastname@student.hpi.de

Abstract. In many MOOCs hands-on exercises are a key component. Their format must be deliberately planned to satisfy the needs of a more and more heterogeneous student body. At the same time, costs have to be kept low for maintenance and support on the course provider's side. The paper at hand reports about our experiments with a tool called Vagrant in this context. It has been successfully employed for use cases similar to ours and thus promises to be an option for achieving our goals.

Keywords: MOOC, Online Learning, Virtual Machines, Virtualization

1 Introduction

MOOCs are often criticized for being mere distributors of instructional videos and theoretical quizzes. Real learning, however, requires more than mere instruction. According to constructivist theory, learning is not providing "true" representations of an objective environment", but enabling the construction of a "relative fit with the world as it is experienced by the learner"—a process called adaptation [2]. Practical exercises and assignments are an essential element in this process, providing the possibility for trial and error and thus to construct knowledge in an active way. More and more courses, particularly in the area of computer science, are aiming to provide hands-on experience and practice online. However, distributing and maintaining practical exercises in a scalable environment poses technical challenges. In this paper we show how existing approaches satisfy the needs of users and providers, and evaluate a novel technique based on virtual machines for providing hands-on experience in IT focused online courses.

1.1 Metrics for Hands-On Exercises

To carry out hands-on exercises in MOOCs, the teaching team needs to provide an exercise environment to the students. As presented in Section 1.2, such environments can be provided in a variety of ways. To evaluate the different approaches we defined the following metrics.

Setup costs The amount of effort and time the user has to invest in order to get started with the exercise.

Support costs The amount of effort and time the course provider has to invest to deliver the exercise or to help users with the setup.

Hosting costs The amount of money and resources the course provider has to spend on providing the required infrastructure.

Control The degree to which the course provider can control the behavior of provided tools or to which it has the ability to help users.

Real world application The degree to which the exercise prepares students for the use of tools as used by professionals and confronts them with real world workflows.

Responsiveness The time needed by a tool to react to user input.

Exploration The degree to which the environment allows students to learn and explore on their own initiative.

Integration How well can the working environment be integrated with the course platform.

Target audience The user group for which an environment is suited.

1.2 Current Solutions for Hands-On Exercises

Existing approaches usually differ in where applications are installed and how students interact with them. We evaluated several of the most common scenarios using the metrics named above. Table 1 gives an overview of this evaluation by estimating values on a scale from 0 to 10 where 10 indicates the best solution for the challenge.

Table 1. Comparison of different ways to bring hands-on exercises to the students

	Local setup	Browser based	VM at user	VM at prov.
Setup	1	10	7	10
Support Costs	1	9	7	10
Hosting Costs	10	3	9	1
Control	1	10	6	8
Real World App.	9	2	9	8
Responsiveness	10	2	9	9
Exploration	10	2	10	9
Integration	1	10	5	8
Target Audience	adv.	beg.	beg./adv.	adv.

Local setup Setting up applications such as programming environments, IDEs or other required tools on the students' machines seems to be the most naive approach as it is closest to the setup of professional users. It teaches the use of real tools and guarantees best possible responsiveness. However, the installation

and configuration of such environments goes beyond the capabilities of novice users and results in high demand of support. Furthermore, the course provider has no control over the tools and thus cannot handle unexpected behavior.

Browser based environments cover the other extreme of the spectrum. The user is provided with browser-based tools running the students' work and test it on correctness. The provider can tailor the tools to the needs and abilities of the course participants and remain in control over their behavior. The user has no setup costs at all and can switch seamlessly between the MOOC environment and practical exercises. Users do not, however, train with real world tools and never undergo the associated learning process. For a detailed overview of existing solutions see [4].

VMs on user machines Virtual machines provide the opportunity to combine benefits of the approaches named above. Course providers can prepare the machine and install software tailored to their needs, thus preventing setup costs and other issues with the students. At the same time computation is performed on the participants' machines, therefore, lowering costs for the provider. Students use a system as it would be used by a professional, thus, training them for real world applications. The issue with this approach is that such VM images can easily take several Gigabytes of memory. Distributing such VMs—in a way that is both cost-efficient for the provider and comfortable for the user—is the primary focus of this work.

VMs hosted by the course provider can provide fully functional systems whose setup would go beyond the scope of a course. An example for courses on network security was introduced in [7]. These server-side VMs are beyond the scope of this paper.

2 Related Work

The rising demand for practical assignments in MOOCs is well documented. Willems et al. [6] describe the introduction of hands-on assignments to openHPI¹. Here, the focus lies on applications installed on the students machine. The inherent heterogeneity was used to diversify the assignment and underline the complexity of distributed applications. The paper does, however, report massive support efforts and problems and suggests moving to more homogeneous environments. Staubitz et al. [5] report about a different approach on the same platform, which provided the students with hands-on coding assignments. Here the focus lies on cost efficient and easy to set up practice environments by integrating browser-based third party coding tools into existing MOOC infrastructures including automated assessment.

Staubitz et al. [4] provide an overview on the wide range of possible approaches to implement hands-on exercises similar to Section 1.2. While they focus on coding assignments and the use of web-based editors, they do provide deep insights into various testing and assessment methods.

¹ <https://open.hpi.de>

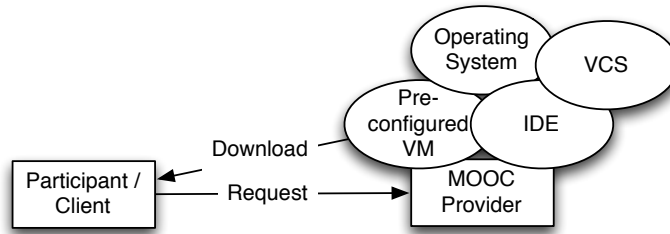


Fig. 1. Downloading a preconfigured VM—including operating system, integrated development environment, and version control system—from the MOOC provider’s server.

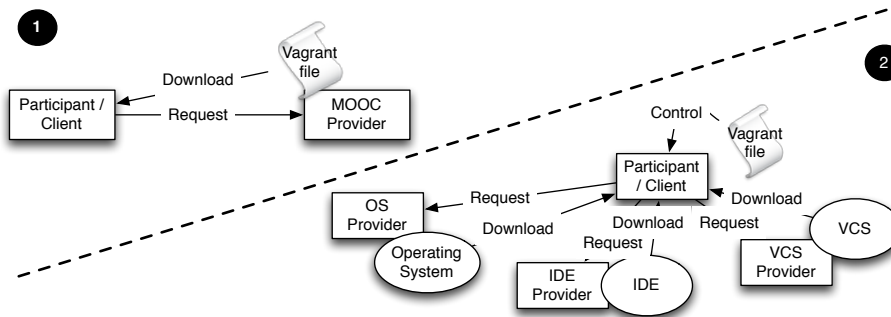


Fig. 2. (1) Downloading the Vagrantfile from the MOOC provider’s server. (2) Downloading the required tools from their original location.

There have been few courses and research projects exploring the use of virtualization software on the user side. E.g. a course on software defined networking was conducted by Princeton University on Coursera², which was based on the Mininet VM [3]. In another course on Coursera, *Audio Signal Processing for Music Applications*³ a VM was provided with the required open source software pre-installed. Berger et al. [1] conducted a course on database systems using VMs as an optional means for the assignments. They suggest that VMs could not only be helpful for reducing setup and support costs for hands-on exercises, but also for collecting data on the learning process.

3 Virtualization Software and Vagrant

As suggested in Section 2, virtual machines can, and have been, successfully employed to form the basis for realistic hands-on exercises in MOOCs. The question

² <https://www.coursera.org/course/sdn1>

³ <https://class.coursera.org/audio-002>

that remains is how to deploy such VMs to the participants. Traditionally, VMs are distributed through proprietary file formats that contain a description of the virtual machine and its state including its hard drive.

While this meant that every machine is virtually identical, such files rapidly grow to multiple Gigabytes, which makes them hard (respectively expensive) to distribute (see Figure 1). With Vagrant⁴ a relatively new technique was established to deploy virtual machines. In contrast to traditional VMs—where the VM is created, preconfigured, stored, and then distributed—a Vagrant box is defined only by a text file called *Vagrantfile* (see Figure 2). Using this file, Vagrant creates a virtual machine on the host system using existing VM providers such as VirtualBox.

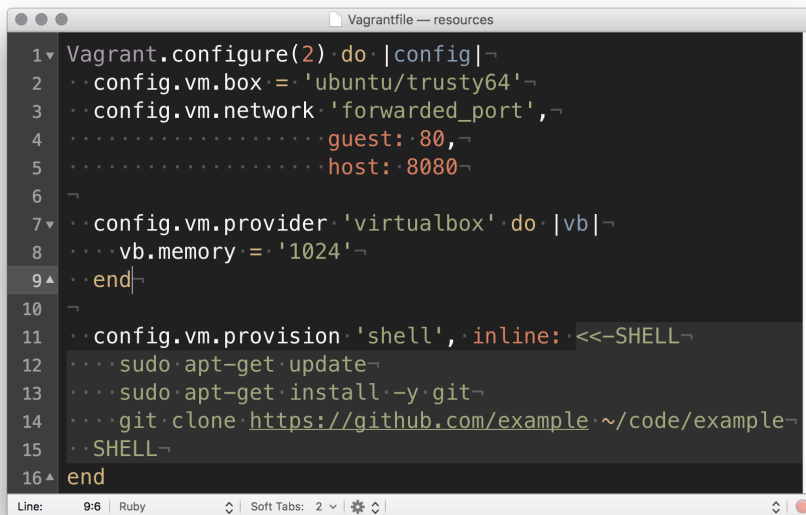
3.1 Vagrantfile

An exemplary Vagrantfile is displayed in Figure 3. The VM on the users machine is created and configured according to the description given in this file (see also Figure 2 (2)). The first commands define which base box should be used and some network settings. The commands in the block `config.vm.provider` include settings specifically directed at VirtualBox. With `config.vm.provision` the provisioning steps are defined. These steps are executed after the setup of the machine and are used to install software or configure the system. It is this provisioning step where the VM is tailored to the needs of the MOOC it is used in. To deploy a basic setup only a small text file has to be provided on the MOOC platform. All other resources are pulled from their original locations on the internet. If additional custom content has to be deployed, Vagrant can pull this from any location on the internet, e.g. the MOOC platform, or cloud storage. Thus hosting costs, for storage and traffic, on the MOOC provider’s side are kept at a minimum.

3.2 Customization of Vagrantfiles

As Vagrant builds on nothing but the Vagrantfile, customizing a machine requires a mere substitution. Willems et al. [6] suggest using heterogeneity to diversify the learning process or tailor it to specific users. Vagrant promises to enable teachers to achieve the goal of per user customization more easily. Rather than providing a single Vagrantfile, individual Vagrantfiles could be generated on demand from a template by substituting single system variables. Customization on a per user level also has the distinct advantage that user specific information such as access tokens can be embedded in the VM. This could be put to use for the hand-in procedure as discussed in Section 4. As a proof of concept we implemented a tool that writes arbitrary content into a Vagrantfile template depending on the user it is supposed to be delivered to.

⁴ <https://www.vagrantup.com/>

A screenshot of a code editor window titled "Vagrantfile -- resources". The editor shows a Ruby Vagrantfile configuration with line numbers 1 through 16. The code defines a VM configuration for Ubuntu Trusty 64, sets network forwarding for port 80 on host 8080, uses the VirtualBox provider with 1024MB memory, and provisions the VM with shell commands to update apt, install git, and clone a repository from GitHub. The status bar at the bottom indicates "Line: 9:6 | Ruby" and "Soft Tabs: 2".

```
1 Vagrant.configure(2) do |config|
2   config.vm.box = 'ubuntu/trusty64'
3   config.vm.network 'forwarded_port',
4     guest: 80,
5     host: 8080
6
7   config.vm.provider 'virtualbox' do |vb|
8     vb.memory = '1024'
9   end
10
11  config.vm.provision 'shell', inline: <<-SHELL
12    sudo apt-get update
13    sudo apt-get install -y git
14    git clone https://github.com/example/~code/example
15    SHELL
16 end
```

Fig. 3. Basic Vagrantfile

3.3 Using Graphical Applications

Vagrant virtual machines are usually accessed via command line *ssh* sessions, more specifically with the Vagrant specific command *vagrant ssh*. Standard Vagrant boxes in their default configuration do not come with a graphical user interface (GUI). Access via *ssh* would pose a great challenge to the novices, who are our main target group. As one of our intended use cases involves graphical applications such as Integrated Development Environments (IDEs) we examined several methods for GUI access to a Vagrant box. The traditional approach for coding within Vagrant is to use a locally installed editor while the VM serves only as the execution environment. The simplest approach would be to use a shared folder that is accessible both by the host and the guest system. However, since the idea of our approach is to deliver a fully preconfigured environment, this option misses the point.

Alternatively, the GUI of an application running in the VM can be forwarded to the host system using X11. After connecting to the Vagrant box with *vagrant ssh* one may simply launch GUI applications and display them at the local X Server. However, as Windows does not support X Server natively, additional setup is required. This way we only replace one complexity with another.

Finally, Virtual Box can be configured to deactivate the headless mode and launch a window showing the whole desktop. This method works on all operating systems with low configuration overhead. We therefore favor this option when targeting a wide audience. To improve the performance of the VM, we selected

the LUbuntu operating system as it comes with the lightweight LXDE desktop instead of a more standard Unity, Gnome, or KDE desktop.

4 Implementation

In this paper we consider two different scenarios: A Java programming course at an intermediate level and a security lab in an Internet security course. Those two courses are very different in nature and each has its own requirements and challenges. Both courses have already been taught in a similar form on openHPI. The problems that have been reported by some participants with setting up the required environments on their own have been a main inspiration for the paper at hand. Next to the bare setup, we will consider the tasks of downloading new exercises, updating faulty exercises and submitting solutions.

4.1 Java Programming Course

The envisioned Java course is a follow-up to an introductory hands-on Java programming course. After basics of the language have been taught using a browser-based learning environment, this course teaches the use of real world tools, particularly the Eclipse IDE, by having the participants apply their knowledge in a Java programming project. It requires a JDK, the Eclipse IDE, and git to be installed and configured correctly. For novices this can be a serious challenge. Using Vagrant we can reduce the frustrating setup procedures to installing VirtualBox and Vagrant and deliver a perfectly configured course environment to the student. Throughout the course students download weekly assignments onto their virtual machines, solve them and submit the results back to the course provider. All of the following automated solutions might benefit from templated Vagrant files as described in Section 3.2. Those templated Vagrant files could introduce user IDs and authentication tokens as environment variables into the VM.

Task Retrieval & Update Assignments should be delivered to the users' machines on a regular basis. Also, a method for updating buggy or incomplete assignments needs to be in place as mistakes tend to occur even with an optimal preparation. Often, students download assignment files into a folder of their choice. As the manual transport of files into the VM can be tedious, we recommend having a clickable script that downloads the newest versions inside the VM. These scripts might be customized with the user's ID or a token to authenticate with the course provider. In the simplest form, the scripts download the files as a zip and unpack them into the user's Eclipse workspace. Hash sums can guarantee that the scripts download new or updated files only. Somewhat more complex, the scripts can retrieve the assignment files using VCS software, such as git. These tools might save resources using incremental updates over the zipped solution if the assignment files are very large.

Solution Upload After working on an assignment, students need to upload solutions to the course provider. It is important that the upload happens in an automated fashion, because file transfer between the host and the VM can be error prone. Again, the simplest solution might be zipping the assignment solution and sending it to the course provider for grading via a simple HTTP upload. If course providers and students are interested in keeping a history of the uploaded solutions it makes sense to use a VCS tool for uploading. It comes at cost of hosting a repository for every student, however.

4.2 Security Labs

The requirements of a security lab are very different from those of a programming course. Usually, a security lab involves multiple machines with a specific network configuration and configured services. Oftentimes, instead of producing a piece of software the assignment is to obtain information by intercepting communication or breaking into systems (penetration test). Security labs can involve software that is difficult to obtain on users machines and that can cause harm if used improperly. Usually, these two challenges have been tackled by giving the user access to remotely hosted virtual machines. Using virtual machines at the users side reduces hosting cost for the organizers, but introduces the concern of users obtaining the information or cheating by introspection of the local machines. In the following, if there are multiple machines involved, the term attacker machine refers to the VM that is equipped with the required penetration testing tools for the student. Accordingly, the term defender machine refers to a VM that exposes exploitable vulnerabilities.

Multiple machines can be configured in a single Vagrant file. Afterwards, network configuration is a matter of assigning IP addresses and naming the virtual networks. Multi-host environments include scenarios, such as remote exploitation with one attacker and one defender machine or man in the middle attacks with one attacker and two defenders.

Separation from the Outside World To provide a safe learning environment for the student we want to prevent accidental attacks of uninvolved targets. Those uninvolved targets can be machines on the learners local network or hosts on the the internet. For a proof of concept, we use Linux' standard firewall *iptables* and the extension *Conntrack*. First, we disallow any traffic leaving the machine to the outer network by using *iptables* to prevent any new outgoing connections on Vagrant's default NAT adapter. *Conntrack* now helps to allow an exception: Only outgoing traffic of connections that were initiated from the outer world to our machine is allowed to leave. Hence, users can establish an *ssh* connection to the machine and have a two way communication, but they cannot open a new connection from inside of the machine to the outside. The shown provisioning steps have to be executed for every VM of the scenario.

Task Retrieval & Update Assignments might require exploitation of various services and operating systems. Hence, security labs need to be carried out using assignment specific VMs. This reduces the reusability of the provisioned VMs.

For each assignment that requires a new set of VMs a new Vagrant file must be downloaded and in turn the VMs must be provisioned. After that, users can attempt to solve the assignment.

Task Submission Often, users have to obtain information or so called *flags* in security labs. Those flags might be a password, a user name or a random string. They can be obtained from various activities, such as password cracking, reverse engineering or exploiting services. Given that they are only strings, submitting flags can be as simple as pasting them into a browser form. The feasibility of this approach has been demonstrated by Staubitz, et al. [5] for a different use case but would work here exactly the same way.

Cheating Many concerns about cheating arise if the defending machines are run by the students themselves. There are several ways to obtain the information in unrighteous ways. Students might try to get access to the machines by guessing login credentials, using hypervisor functionality or mounting the virtual machines hard drives to another machine. There are technical remedies for some of these back doors, but it is ultimately very difficult to protect an application from the user running it. Also, these remedies introduce technical complexity. While course providers can prevent students from mounting the hard drives by encrypting them and hiding the key on the boot partition, this introduces a new source of errors and lowers reusability. A more feasible way of preventing cheating might be embedding the submission of flags into a quiz of the respective learning section. Students can prove that they rightfully obtained the flag by demonstrating knowledge of the involved techniques, tools, and steps.

5 Evaluation

Assuming that VMs can reduce local setup issues for practical tasks in a MOOC environment, we prototypically have set up two basic scenarios: a penetration testing setting in an Internet Security course and a programming environment in a Java course. With this experiment we have shown that the heterogeneity of different environments, maintenance, and support efforts for the teaching teams can be reduced by employing VMs. We have also shown that the benefits of employing VMs are differing from scenario to scenario. For the Java programming environment described in Section 4.1 we recorded a number of metrics⁵. The Ubuntu base image accounts for 378 MB which have to be downloaded and stored locally. The required packages add another 498 MB to download. The fully built VirtualBox VM has a size of 3598 MB. Provisioning the box, not counting the time required for downloading the base box, took 11:37 minutes. In comparison, an Ubuntu box without a GUI and having some lightweight tools only takes up roughly 1500 MB. This shows that providing Vagrantfiles instead of fully pre-configured VMs can reduce the costs of a MOOC provider significantly.

⁵ The machine that has been employed was a standard PC with 8 GB RAM, i5 CPU 2.67 GHz, SSD running Windows 7

6 Future Work

We intend to employ this approach in one of our upcoming courses and evaluate its acceptance among the participants and its perceived usefulness in terms of the defined learning outcomes.

7 Conclusion

In this paper, we examined the use of Vagrant as a tool to deploy preconfigured, hands-on learning environments to our online students. Vagrant provides the opportunity to deliver customizable and cheap to host VMs that create realistic and responsive hands-on environments. Furthermore, we introduced eight relevant metrics to evaluate the different solutions for hands-on tasks in MOOCs and assess their benefits and shortcomings. We have shown that Vagrant reduces the friction for creation, distribution, setup and update of Virtual Machines. We demonstrated the use of Vagrant in the context of different scenarios and discovered and discussed several challenges in carrying out hands on assignments using virtual machines. In summary, we can say that the use of Vagrant, in complex server scenarios, can reduce hosting costs, improve the user experience for learners compared to traditional virtualization software and thus reduce support efforts on the course provider's side.

References

1. Berger, O., Gibson, J.P., Lecocq, C., Bac, C.: Designing a virtual laboratory for a relational database mooc. In: Proceedings of the 7th International Conference on Computer Supported Education. pp. 260–268 (2015)
2. von Glasersfeld, E.: Learning and adaptation in the theory of constructivism. *Communication and Cognition* 26(3/4), 393–402 (1993)
3. Lantz, B., Heller, B., McKeown, N.: A network in a laptop: rapid prototyping for software-defined networks. In: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. p. 19. ACM (2010)
4. Staubitz, T., Klement, H., Renz, J., Teusner, R., Meinel, C.: Towards practical programming exercises and automated assessments in massive open online courses. In: International Conference on Teaching, Assessment, and Learning for Engineering (TALE). pp. 23–30. IEEE (2015)
5. Staubitz, T., Renz, J., Willems, C., Jasper, J., Meinel, C.: Lightweight ad hoc assessment of practical programming skills at scale. In: In Proceedings of IEEE Global Engineering Education Conference (EDUCON). pp. 475–483. IEEE (2014)
6. Willems, C., Jasper, J., Meinel, C.: Introducing hands-on experience to a massive open online course on openhpi. In: International Conference on Teaching, Assessment and Learning for Engineering (TALE). pp. 307–313. IEEE (2013)
7. Willems, C., Meinel, C.: Tele-lab it security: an architecture for an online virtual it security lab. *International Journal on Online Engineering (iJOE)* 4, 31–37 (2008)