Virtual Machines via Vagrant and VirtualBox

Evan Russenberger-Rosica

What is a Virtual Machine?

The fundamental idea behind a **virtual machine** (VM) is to "abstract the hardware of a single computer (the CPU, memory, disk drives, network interface cards, and so forth) into several different execution environments, thereby creating the illusion that each separate environment is running on its own private computer" [Operating Systems Concepts, 711]. In other words, VMs allow (or give the appearance of allowing) several different operating systems to run on the same hardware, at the same time! The underlying hardware system that runs the virtual machines is known as the **host system**, and the virtual machines are known as **guest systems**. VMs are created and run by a program known as a **virtual machine manager** (VMM) (or **hypervisor**) [712].

Advantages of Virtual Machines

- Parity:
    - Everyone develops on exactly the same system, regardless of the host OS. Thus "works on my machine" bugs are eliminated.
- Security:
    - The VM can be damaged or even destroyed without any consequences to the host OS.
    - In the event of a mistake, the VM can be quickly rebuilt from the box.
- Flexibility:
    - VM's enable the user run software that would not otherwise run on the host. E.g. Linux programs a Windows host.
- Portability:
    - In minutes, the entire virtual machine can be saved, duplicated, and shared through the cloud!

Configuring a Virtual Machine

Several choices of hypervisors are available on the internet. We will use **VirtualBox** which is free and reliable. We will also use **Vagrant**, a configuration layer for VirtualBox which is also free and reliable. Follow the following instructions to prepare your virtual machine.

Step 1: Download and Install Vagrant

Download the appropriate version of Vagrant for your host OS here. You can verify that the installation was successful by typing vagrant on your system command line as shown below.

Step 2: Download and Install VirtualBox

VirtualBox can be downloaded here. We will use it to download a "base image" or "**box**" from which virtual machines are built.

Step 3: Create a Directory to Store Your Virtual Machines

You need somewhere to store the virtual machine which will be created from the box. For our purposes, we'll create a directory named DL-VM (short for "deep learning virtual machine") as a subdirectory of your home directory (denoted by ~). We then change into this directory. This can be done at the system command line (for both powershell and bash, **DO NOT USE** the old windows command prompt known as cmd.exe) as follows.

Style Note: We use `>` to denote the host prompt, and `$` to denote the guest prompt.

```
> # The `>` shows we are on the host's command line.
> mkdir ~/DL-VM                          # make directory ~/DL-VM
> cd ~/DL-VM                             # change to directory ~/DL-VM
```

Step 4: Download the DeepLearning Box

In the directory `~/DL-VM`, run

```
> vagrant init errosica/DeepLearning   # download the DeepLearning box
```

This will download the DeepLearning box from the Vagrant cloud. I have specially designed this box to contain software used in deep learning by default, so that the user does not have to deal with installing various libraries and solving conflicts between them. The file is quite large (~4GB), so this may take a while.

Step 5: Starting the Virtual Machine

From the directory ~/DL-VM,run:

```
> vagrant up                             # start the virtual machine
```

Step 6: Connect to the Virtual Machine

In the directory `~/DL-VM`, run:

```
> vagrant ssh                          # connect to the VM
```

After this step, you will be connected to the virtual machine, and your prompt will change to reflect this:

```
(base) vagrant@ubuntu-bionic:~$
```

Step 7: Activate the Anaconda Environment

The deep learning tools are installed in an Anaconda Environment called `py36`. Enter this environment with the following command:

```
(base) vagrant@ubuntu-bionic:~$ conda activate py36
```

Step 8: Start the Python Interpreter

After activating the `py36` environment, you'll see `(py36)` prepended to the path, letting you know that is the current environment. Start the python3 interpreter as below:

```
(py36) vagrant@ubuntu-bionic:~$ python
```

You'll see the python interpreter start and display the python prompt `>>>`:

```
Python 3.6.9 |Anaconda, Inc.| (default, Jul 30 2019, 19:07:31)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Observe that deep learning tools are already installed:

```
>>> import tensorflow as tf, keras, cv2, matplotlib as plt, numpy as np
```

Step 9: Shutting Down

```
>>> quit()                                  # exit python interpreter
(base) vagrant@ubuntu-bionic:~$ exit        # logout from the VM
logout
Connection to 127.0.0.1 closed.
PS C:\Users\Evan\DL-VM> vagrant halt        # shutdown the VM
```

In the future, you can restart the VM by running `vagrant up` in the `~/DL-VM` directory and connecting via `vagrant ssh`.

Intelligent Virtual Machine Use

We now discuss how to intelligently use the VM by comparing two two workflows. In the first workflow [which we will not use here, unless you already know this approach and are comfortable using it], illustrated in figure 0.0.1, everything is done on the VM. Files are stored, edited, compiled and run - all on the VM. This has several downsides

- Since VMs are run "headless" i.e. with no GUI, we can't use modern text editors like V.S. Code or Sublime Text this way. We must use text-based editors over the terminal like vi and Emacs, both of which date to the mid 1970s, and have large learning curves.
- Since the files are stored on the VM, if the VM is deleted (or say accidentally destroyed), so are your files

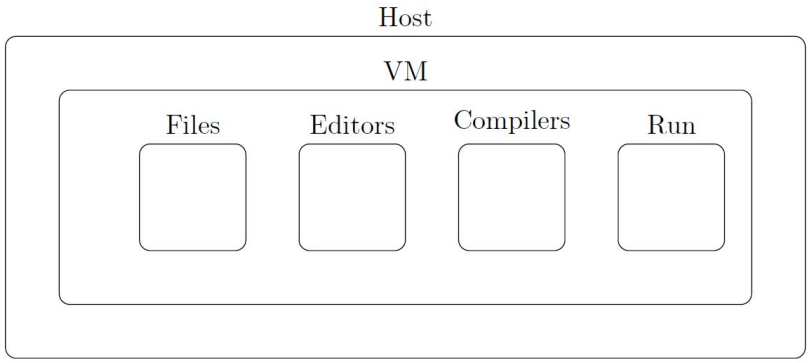

FIGURE 0.0.1. Naive VM Use - Everything Done on VM

The smarter way of using a VM [which we will use] is shown in figure 0.0.2.

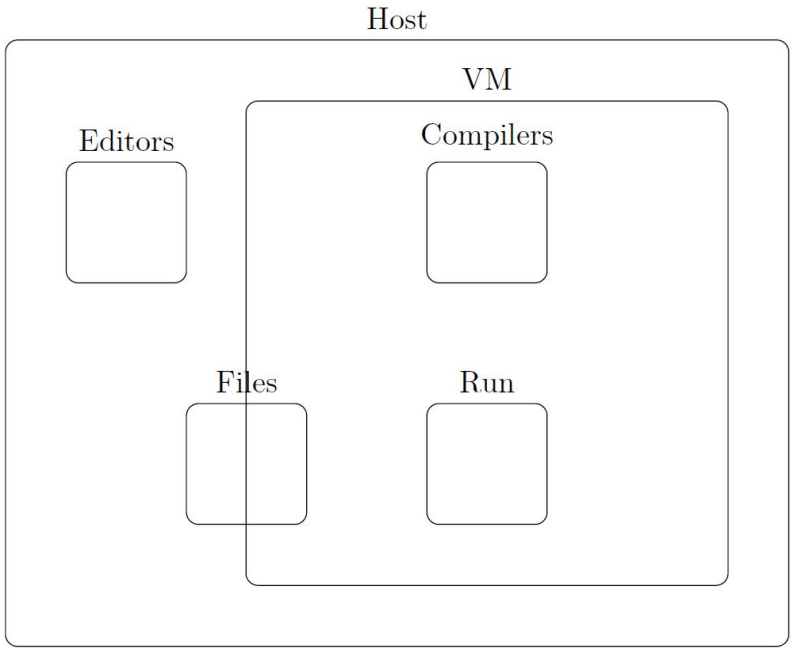

FIGURE 0.0.2. Intelligent VM Use

In this workflow, files are saved in a shared folder, edited using modern GUI editors on the host OS, and compiled and run on the guest OS (i.e. the VM). The key difference is the use of shared folders. Shared folders are useful since files in them can be:

- accessed from both the host OS (e.g., Windows) and the guest OS (in our case, Linux)
- preserved even if the VM is destroyed, keeping your work safe.
- edited using a modern editor of choice on the host machine (e.g., Windows) such as Word, though, for programming, Evan recommends V.S. Code (See below for an introduction to V.S. Code)

Once you've finished writing your code in your editor of choice and saved your program in a shared folder, you can then simply SSH into the VM, change into the shared directory, and use excellent linux compilation tools such as gcc to compile your programs, which can then be executed on the VM. In the next section, we discuss how to use shared folders in practice and first time setup.

Using Shared Files

By default, Vagrant sets the shared directory on the host to be the location of the Vagrantfile. In our case, this is `~/DL-VM`. This is mapped to `/vagrant` on the VM. Thus any files you put into `~/DL-VM` on the host will instantly be shared to `/vagrant` on the guest and vice versa.

We illustrate this below on a Windows host (top) with a linux VM (bottom). Note that the "Vagrantfile" document is shared between host and guest machines. Vagrant takes care of the creation of these basic shared folders for us (although we can add more shared folders by manipulating the vagrantfile (line 44)).

Windows Host

```
> cd ~/DL-VM
> ls

    Directory: C:\Users\Evan\DL-VM

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        1/21/2019   3:40 PM           3135 Vagrantfile
```

Linux Guest

```
(base) vagrant@ubuntu-bionic:~$ cd /vagrant
(base) vagrant@ubuntu-bionic:/vagrant$ ls
Vagrantfile
```

The Visual Studio Code Editor

As previously discussed, an intelligent VM workflow takes advantage of the host GUI. A GUI is particularly useful when editing code, as this enables the use of a mouse, icons, menus, etc. Modern text editors provide some or all of the following features:

- Error checking
- Code navigation
- Code completion
- Code coloring/syntax highlighting
- Version control integration
- Integrated debugging tools
- Extensions/Add-ons/Plugins

For purposes of this class, students may use a text editor of their choice. For students who do not already have a strong preference, we recommend **Visual Studio Code** aka **V.S. Code**. In choosing to recommend a GUI text editor, we follow the wisdom of the crowd. According to the Stack Overflow Developer Survey (2018), **Visual Studio Code** is currently the most popular text editor in the world. V.S. Code is supported by Microsoft, but is free, open source, cross-platform, relatively fast, and extensible. Users can download the correct version for their OS here. Because of its popularity, there are a massive number of self help resources and tutorials. This YouTube channel has good series of tutorial videos, which might be worth a watch.

Troubleshooting

If you have any problems setting up, the following websites contain useful information:

http://www.howtogeek.com/196060/beginner-geek-how-to-create-and-use-virtual-machines/

http://www.instructables.com/id/How-to-install-Linux-on-your-Windows/ (press on the steps one by one

For more on Vagrant, see:

Hashimoto, Mitchell. *Vagrant: Up and Running: Create and Manage Virtualized Development Environments*. " O'Reilly Media, Inc.", 2013.