[INFO-F-309]
Administration de systèmes
Project: part 3

LUCI Horatiu
MAHMOUDI Hamza
SWIRYDOWICZ Szymon
ZHOU Anthony

December 2020

Contents

Glossary

Apache An open-source web server software that powers around 35% of websites in 2020.

Container An entire environment that packages up code, configuration files, libraries and dependencies required for the app to run quickly across environments.

Docker A software platform for building applications based on containers.

Git A version control system (in command line) which allows to track changes made to files.

GitHub A code hosting platform for collaboration and version control that uses Git.

HTML stands for Hypertext Markup Language. It allows the user to create and structure web pages.

Hypervisor A software that creates and runs virtual machines.

SSH A network protocol that gives a secure way to access a computer over an unsecured network.

Virtual Machine An emulation of a computer system.

VirtualBox An open-source hypervisor developed by Oracle.

VMware A virtualization and cloud computing software provider.

Wordpress An open-source content management system that allows to create your own website or blog easily.

1 Part 1: Scenario (October)

An IT company with about 40 employees is encountering problems linked to the maintenance of their server applications running on deprecated software, and of their cloud-based applications operating on constantly updating software.

Each employee is using his preferred operating system and is working in a small team of around 4 people. Each team is using their preferred Software Version Control (SVC) system (mainly GitHub). As each team is tasked with 1 or 2 projects, it has been observed that as projects grow, they become challenging to maintain and the following issues usually arise:

- Code written by one developer often does not work on another's machine and might even not work as expected once pushed onto the server;
- Adding support for new platforms usually requires setting up everything from scratch;
- Virtual Machine's snapshot backups used for version control use significant disk space, while mainly being needed only for little checks across different versions;
- Bringing a new developer to the team, requires serious projects onboarding that is often unfeasible due to time constrains.

Their CEO requires finding a new and cheaper solution for the project maintenance.

2 Part 2 : Analysis Report (November)

2.1 Issue 1

Code written by one developer often does not work on another's machine and might even not work as expected once pushed onto the server.

2.1.1 Further Explanations

Each employee is using their preferred operating system (Windows, MacOS, Linux) and thus development environments vary wildly from employees to servers. Some employees update their software and others do not, while some production servers on which the applications run, are updated and some are not. This remark applies to both the operating system and to the various other applications and dependencies.

2.1.2 Problems

Sometimes a small piece of code is tested with success on a laptop. After being pushed to the software version control system and deployed in testing, it does not work as expected on servers due to different operating systems, dependency versions, architectures, updates etc. There is a lot of work required to find these bugs as they are not apparent in the code itself, and usually a System Administrator is called in resulting in serious delay to the work.

2.1.3 Proposed Solutions

- Require all employees to use the same operating system: unfeasible, because some employees are more comfortable with some operating systems and having them change will incur productivity loss as well as extra costs for Windows licenses or MacOS computers, should those be the choice.
- Always update dependencies and operating systems on all servers and laptops: unfeasible since some application might require precisely some version of some dependency therefore making the application not work if updated.
- Require developers to run their code only in a virtual machine that is configured exactly the same as the production virtual machine. Can be feasible if the configurations of all production virtual machines are documented.

2.2 Issue 2

Adding support for new platforms usually requires setting up everything from scratch.

2.2.1 Further Explanations

When an application needs to be ported or expanded to another operating system, the system administrators manually create the development, testing and production environments, install the necessary software, and give access to developers. The developers are sharing the development and testing environments but this has proven to be inefficient, due to some developers strongly preferring to work on their laptop.

2.2.2 Problems

When a new platform is initially set up, the developers will request that some reconfigurations are made to the new platform such as RAM increase, storage increase, etc. This requires the system admin and the developers to work together until they find the proper configuration, slowing down the development process considerably.

2.2.3 Proposed Solutions

- Give broader control to the developers to configure the servers themselves. This is feasible as long as developers are reasonable with the configurations as to not use significantly more than they need.
- Hire more system admins: this is unfeasible due to the cost increase.
- Have the development and testing environments on each developer's laptop letting them change the virtual machine configuration as they see fit. This can be feasible if the configurations of all production virtual machines are documented and are identical to the development and testing environments.

2.3 Issue 3

Virtual Machine's snapshot backups used for version control use significant disk space, while mainly being needed only for little checks across different versions.

2.3.1 Further Explanations

When a new version of an application is deployed, a snapshot of the virtual machine(s) that run is created manually and stored with an online storage service with a cost of $50 per 1TB of data per month. These snapshots are used to check for bugs and changes in system configurations.

2.3.2 Problems

With each application release, the storage cost increases because all the previous snapshots are stored. Furthermore, it is time consuming to check two configurations side by side and spot the differences.

2.3.3 Proposed Solutions

- Store only the last few versions of the snapshots as to keep a constant online storage cost. Unfeasible, since some clients require that their application or server configuration updates be fully traceable.
- Create scripts to check for configuration changes from one version to another. Also unfeasible, because it is time consuming.
- Have all the server configurations written in a file. Provision the server using that file and when a change is needed, make edits to the file.

2.4 Issue 4

Bringing a new developer to the team, requires serious projects on-boarding that is often unfeasible due to time constrains.

2.4.1 Further Explanations

When maintaining the code or adding new features, some developers download the latest snapshot of the virtual machine that the code runs on, modify the code directly on the virtual machine or delete the code altogether before cloning a git repository with the latest version. Other developers (usually those who work with cloud applications) rebuild everything on each release, most with automated provisioning scripts (Docker, Ansible, Chef, Puppet, Bash) and some do it manually. After the changes are successfully tested, the virtual machines are moved into production.

2.4.2 Problems

Since there are so many ways to work on an application, developers find switching projects (or even just dropping by to help) a rather difficult task. There are already so many different paradigms and ways of work across teams that it is necessary to have the development and testing environments similarly built across teams.

2.4.3 Proposed Solutions

- Implement processes that are to be followed when updating or maintaining legacy or new applications: unfeasible because it requires rigorous documentation and in worst case scenario it means that each team will start documenting their team's practices thus.
- Keep teams separated - unfeasible because the expertise of some very good developers needs to be shared across teams.
- Document the application infrastructure: operating system on which it runs, dependencies, daemons and services for each virtual machine in production as well as the code that runs. This information will be used to create one file per application, that we'll name from now on Vagrantfile and that we'll be further explained below. This is the most feasible option as it's a simple one-day task that only requires each team to document their project from a technical standpoint if this documentation does not already exist.

2.5 Problem sum-up

The problem is that there is nothing in common in the development processes of different teams. Also, current trace-ability and provision processes are not working as expected. In order to solve all problems explained above, Vagrant will be used. Vagrant is an open-source technology used to build and maintain virtual software development and testing environments in a portable matter. It will simplify the software configuration management (controlling changes in the software) of the virtualisations in order to increase development productivity and decrease cost.

2.5.1 Description

Vagrant uses "Provisioners" as building blocks to manage the development environments. Provisioners are tools that allow users to customise the configuration of virtual environments. Puppet, Ansible, Chef and even Bash scripting can be used as provisioners in the Vagrant ecosystem. Providers are the services that Vagrant uses to set up and create virtual environments. Support for VirtualBox, Hyper-V, and Docker virtualisation ships with Vagrant, while VMware and AWS are supported via plugins. This will help to adapt all out use cases, from legacy applications running on deprecated software, to state-of-the-art server-less computing.

Vagrant sits on top of virtualisation software as a wrapper and helps the developer interact easily with the providers. It automates the configuration of virtual environments using Chef, Ansible, Puppet and Bash, and the user does not have to directly use any other virtualisation software. Machine and software requirements are simply written in a file called "Vagrantfile" to execute necessary steps in order to create a development-ready virtual machine. This Vagrantfile is essentially a file written with Ruby syntax, that has all the system commands given as declarations.

2.5.2 Steps to implement

- Every developer will be given all the necessary information to install Vagrant on their work machines.
- Every application that the company develops will be wrapped into a Vagrantfile, along with the virtual machine configuration that it needs to run on.
- Base version of all the operating systems used in production will be stored in another server if necessary. The Vagrantfile can be instructed to download the operating system image directly from the provider or from a custom location.
- When developing and testing an application, Vagrantfile will contain the environment configuration and the Git repository that contains the application code. The developer will then connect to the virtual machine via SSH and develop or test the application in an environment that is identical to the production one.

2.5.3 Choice of OS

Vagrant works on all common operating systems.

2.5.4 Servers Required

One server to be provisioned to store the operating systems what will be used as an initial image when configuring an environment with Vagrant.

3 Part 3 : Implementation Report (December)

3.1 Choices made

Vagrant is a common tool for virtual machines management combined with a simple command line interface. The tool is expected to make the entire process of provisioning a virtual machine for development and testing purposes easy without polluting the main installation with tons of configuration files. In the back-end, there is one Virtual Box instance utilised by the Vagrant tool to run virtual machines. **A Vagrant box is just another term for virtual machines handled specifically by this utility**. Vagrant supports Hyper-V, Docker, VirtualBox, and this tool can manage other machines like Amazon EC2, VMWare, etc., as well.

In our environment, Vagrant will be used to provision virtual machines for development and testing our applications and websites. An application or website deployed on an internet server is called a project. As projects vary wildly, from simple HTML websites running on updated software, to complex Java apps running on legacy software, to cloud-based web-apps, Vagrant has been found as a common denominator to manage their development and testing environments. It is important to have a centralised management tool for development and testing as projects age and developers switch between them. Vagrant keeps an expected configuration for all the projects, giving the developers more time to work in code and worry less about compatibility issues.

Vagrant has a clear and beginner-friendly documentation available on the main website: `https://www.vagrantup.com/docs`. This documentation enables developers to create the new Vagrant-based environments themselves in very little time. If necessary, Vagrant offers a getting started page allowing you to understand all Vagrant's basic concepts in around 20 minutes, on : `https://learn.hashicorp.com/collections/vagrant/getting-started`. However, more complex projects will require help from the system administration team so that the transition runs smoothly.

3.2 Technical description

The main configuration file is called *Vagrantfile*. Its primary function is to describe the type of machine required for a project, and how to configure and provision these machines.
Vagrantfile is automatically created by starting a new project in a directory by using the command `vagrant init`, which can take a box name and URL as arguments.

3.2.1 Useful commands:

- `vagrant init`: Used in the desired directory, creates a configuration file for the VM used in the project.
- `vagrant box`: Allows to manage the boxes on the local machine. This includes adding, removing, listing and updating.
- `vagrant up`: Initialises the VM specified in the Vagrantfile.
- `vagrant ssh`: Allows to login in the VM with the default credentials via secure shell (ssh).
- `vagrant reload`: Reloads the configuration and runs the box.
- `vagrant snapshot`: Allows to create a backup and go back to the previous state. This can be done using `vagrant snapshot push`, to create a snapshot, and `vagrant snapshot restore snapshot-id`, to restore the snapshot. The snapshot-id can be found from the snapshots list by using `vagrant snapshot list`.
- `vagrant halt`: Stops the VM, start again with `vagrant up`.
- `vagrant provision`: Automates the process of installing and configuring software so that it is repeatable and requires no human interaction. The provisioning scripts can be written in Bash, Ansible, Puppet, etc.
- `vagrant destroy`: Deletes the VM.
- `vagrant --help`: Prints most common commands and is a great place to start if you require further research.

3.2.2 Vagrantfile configuration

Vagrantfile offers many options to configure Vagrant and easily launch your project. The generated file comes as a well commented template allowing you to instantly set up everything without needing further research, whether you want to quickly launch a VM environment or set up a running server. Below is a list of the most useful options you might want to know about :

- `config.vm.box`
  The name of the Vagrant box that will be used to build the Vagrant virtual machine. Example: `config.vm.box = "project1"` will name the Vagrant box as `project1`

- `config.vm.box_version`
  The version of the Vagrant box to be used (it will default to latest version if this parameter is not used). Example: `config.vm.box_version= "20180529.0.0"` will specify version `20180529.0.0`

- `config.vm.box_url`
  URL referencing where the Vagrant box should be downloaded from if it doesn't exist on your system. Example downloading a Ubuntu Vagrant box directly from `ubuntu.com`: `config.vm.box_url = http://cloud-images.ubuntu.com/xenial/current/xenial-server-cloudimg-amd64-vagrant.box`

- `config.vm.synced_folder`
  A synced folder enables you to share files between your host and guest machine. Great for developing on your preferred OS and then simply transferring everything for testing on the guest OS.
  This option takes two mandatory arguments, the shared folder's path on the host and the mount path on the guest. It is common practice to sync the folder containing the Vagrantfile with `/vagrant` on the Guest machine, as follows: `config.vm.synced_folder ".", "/vagrant", owner: "vagrant", group: "vagrant", mount_options: ["uid=1234", "gid=1234"]` (this also sets the owner, group and mount options but these parameters are not compulsory).

- `config.vm.provision`
  Configures the provisioner. Generally provisioned with shell scripts, allows either to specify a path to the shell file with `path:"script_name.sh"` or directly in Vagrantfile with `inline:` followed by code and ending with `SHELL`. However, any provisioning software such as Ansible or Puppet can be used.

- `config.vm.network`
  Configures networks on the Virtual Machine and allows expanding Vagrant's environment access.

3.2.3 Vagrant network configuration

Vagrant offers three network options:

- Port forwarding
- Private network (host-only network)
- Public network (bridged network)

Port forwarding

By default, we can access Vagrant VMs via SSH using `vagrant ssh command`. When we access a VM via SSH, Vagrant forwards port 22 from the guest machine to an open port in the host machine. This is called port forwarding. Vagrant automatically handles this port forwarding process without any user intervention. You can also forward a specific port of your choice.
For example, if you forwarded port 80 in the guest machine to port 8080 on the host machine, you can then access the web server by navigating to http://localhost:8080 on your host machine.

The forwarding port can be configured in the Vagrantfile at the following line:

```
config.vm.network "forwarded_port", guest:  80, host:  8080
```

In case the defined port is already in use, Vagrant can automatically find an unused port by enabling the auto-correction option in the port forwarding definition in your Vagrantfile.

```
config.vm.network "forwarded_port", guest:  80, host:  8080,
auto_correct:  true
```

By default, Vagrant uses TCP protocol for port forwarding. However, this can be changed to use UDP protocol if you want to forward UDP packets.

```
config.vm.network "forwarded_port", guest:  80, host:  8080,
protocol:  "udp"
```

Private network

In private or host-only networking, a network connection is created between the host system and the VMs on the host system. The private network is created using a virtual network adapter that is visible to the host operating system. The other systems on the network can't communicate with the VMs. All network operations are allowed within the host system.

The private network can also function as DHCP server and has its own subnet. Each VM in the private network will get a IP address from this IP space. So, we can access the VMs directly using the IP from the host system.

To configure private or host-only networking in Vagrant with static IP address, adapt the following line in the Vagranfile:
`config.vm.network "private_network", ip: "192.168.121.60"`

Here, *192.168.121.60* is an arbitrary IP address given to the VM.

If you want to set IP address automatically from DHCP, modify the private network definition like below:
`config.vm.network "private_network", type: "dhcp"`

Public network

In public or bridged networking, all the VMs will be in same network as your host machine. Each VM will receive its own IP address from a DHCP server (if available in the local network). So all VMs will act like just another physical system on the network and they can communicate with any system in the network.

To configure public or bridged networking, find the following line in the Vagrantfile and modify the network definition like below:
`config.vm.network "public_network"`

The VM will automatically get an IP address.
However, if you want to set a static IP, just modify the network definition:
`config.vm.network "public_network", ip: "192.168.121.61"`

Enabling multiple network options

Each network option has its own upside and downside. For some reason, you might want to configure all network options to a single VM. If so, Vagrant has the ability to enable multiple network options. All you have to do is just define the network options one by one in the Vagrantfile.

```
config.vm.network "forwarded_port", guest: 80, host: 8080,
auto_correct: true
config.vm.network "private_network", ip: "192.168.121.60"
```

When you create a new VM with this Vagrantfile, vagrant will create a VM with following network details:

- configure port forwarding
- configure a private network with static IP 192.168.121.60

In addition to configuring multiple types of networks, we can also define multiple networks as well. For example, you can define multiple host-only network with different IP address.

```
config.vm.network "private_network", ip:  "192.168.121.60"
config.vm.network "private_network", ip:  "192.168.121.61"
```

3.3 Implementation steps - Vagrant Basics

Below is a condensate of the most important information you'll need. For more specific information consult Vagrant's official documentation on `https://www.vagrantup.com/docs`.

3.3.1 Vagrant installation

Download the latest installer of Vagrant on `https://www.vagrantup.com/downloads` by choosing the right version depending on your operating system and simply following the installation steps.

To verify if the installation was accomplished correctly, open a new command prompt or console and enter the following command: `vagrant -v`

3.3.2 Choosing Providers

Vagrant relies on software called hypervisors to create the virtual environments. You'll be required to install a provider depending on the type of box you'll want to use, and in some cases of your host operating system. Each has its pros and its cons, below are some of them you might consider:

- `VirtualBox`
  VirtualBox is the most accessible option since it is free and cross platform.

  Download VirtualBox's installer or sources on `https://www.virtualbox.org/wiki/Downloads`.

  - macOS
    VirtualBox might be downloaded using Homebrew.
    ```
    brew install virtualbox
    brew install virtualbox-extension-pack
    ```

  - Linux
    VirtualBox might be downloaded using apt.
    ```
    sudo apt-get update
    sudo apt install virtualbox virtualbox-ext-pack
    ```

- `VMware`
  VMware is not as accessible as Virtualbox, nor free, but offers a overall better performance and stability thanks to, inter alia, Vagrant's developer, HashiCorp, which develops an official VMware provider. You can learn more about pricing and advantages on `https://www.vagrantup.com/vmware`.

VMware requires additional installation steps that you might want to check on `https://www.vagrantup.com/docs/providers/vmware/installation`.

- `Hyper-V`
  Hyper-V works for Windows 8.1 and later versions of Windows only. If you meet those specifications, all you have to do is to enable it on PowerShell with the following command:
  `Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All`

  Hyper-V can be disabled at all moments with :
  `Disable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V-All`

  Note that Hyper-V must be disabled in order for other hypervisors as VirtualBox or WMware to work.

  For further information on Hyper-V refer to Microsoft's documentation:
  `https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/quick-start/enable-hyper-v`

- `Docker`
  Despite not being a virtual machine hypervisor, Docker is supported with Vagrant to back your environment with docker containers:
  `https://www.docker.com/`

Note for Windows users.
Whenever you have enabled Hyper-V on you system or it was enabled by default, make sure it is disabled before using any other virtualisation provider. Refer to 3.3.2-Hyper-V.

3.3.3 Your first Vagrant Environment

Choose a box
"Boxes" in Vagrant are the base images used by Vagrant to quickly clone a virtual machine, thus note that they are provider-specific. Vagrant proposes a public cloud storage where you can choose a box from `https://app.vagrantup.com/boxes/search`, but you are free to create and use your owns. Refer to section 3.4.1 for further explanation.

Initialize the directory
Once the box has been chosen, enter this command:
`vagrant init <url or name of the box>`
This command will create a Vagrantfile for you. The Vagrantfile allows you

to:

- Mark the root directory of your project. Many of the configuration options in Vagrant are relative to this root directory.
- Describe the kind of machine and resources you need to run your project, as well as what software to install and how you want to access it.

Bring up a virtual machine

Enter this command: `vagrant up`
Once this command will finish, you will have a virtual machine running the chosen box.

3.4 Implementation steps - Projects

3.4.1 Storing the Virtual Machine Images

An important prerequisite before migrating to a Vagrant-based development and testing environment is having all the needed virtual machine image files available. The latest versions of known Linux distributions that work with Vagrant can be found on the vendor website. However, there are some virtual machine images we use that are modified for security reasons. These images will need to be self-hosed by our company on another server.
In order to create a base box for Vagrant to use and host it we need to do the following:

- Create or select a virtual machine with the desired virtualisation provider (VirtualBox, VMWare, etc.)
- Log into the newly created guest machine to change the settings and install desired dependencies for intended use (e.g. for VirtualBox provider, VirtualBox-Guest-Additions package is needed)
- Log out and on the host machine run `vagrant package -base my-virtual-machine`, where `my-virtual-machine` is replaced by the name of the virtual machine in VirtualBox to package as a base box.

After the Vagrant box file is generated with a .box extension, it can be hosted on our company internal server or on each developer machine, should they refuse to have it hosted. From there, the Vagrant box can be accessed as follows:

- If the Vagrant box is hosted on the company internal server in `/var/www/html/vagrant_boxes` it will be accessible at `http://company_host.com/vagrant_boxes/box_name.box`
- If the Vagrant box is hosted locally on the computer, the box will need to be added to Vagrant `vagrant box add -name my-box /path/to/box_name.box` and specified in the `config.vm.box` field inside the Vagrantfile before running `vagrant up`.

3.4.2 Moving old projects to Vagrant

In our environment, each project consists of one or more applications running on one or more servers that communicate between them. Each project will need to be documented from a technical standpoint (server(s) hostname, hardware specifications, network environment software/services/daemons running on the server and obviously, the code that the company built).

The following step is to compile each project documentation into a Vagrantfile and store the it into the software version control alongside the code that it architects the virtual machine for. Therefore, every time the code is downloaded, the Vagrantfile will be downloaded as well. If the Vagrantfile is configured to sync the folder that it's in to `\vagrant` on the guest machine, the project can be downloaded and run in 3 simple steps steps: `git clone <repo>; cd <repo>; vagrant up`

3.4.3 Sharing access

Sharing and distributing the box is as simple as giving somebody access to a software version control system (such as GitHub) repository. The Vagrantfile specifies where to get the box from, therefore everyone can access all the boxes, considering they have internet access and are permitted to connect to the company intranet server.
Access to a running Vagrant box for debugging or testing can be made via IP, hostname or with the `vagrant ssh` command. Check the network section (3.2.3) to understand more about how virtual machine networks can be setup.

3.4.4 Start working with an existing project

The issue of working in another project to give some help is now solved. If a developer is given access to another project's SVC repository, they can see the code as well as the Vagrantfile which contains the server specifications. This holds true as well for system administrators brought into a project for infrastructure problems.
Teams are now able to share configurations for the servers as well as code and spot exactly where a bug or another issue might be. This will considerably reduce the time of somebody more experienced helping with an issue.

3.5 Example project

Some template projects have been made by the system administration team. These projects are made in order to show how easily an Apache-hosted website can be deployed, as well as the depth to which Wordpress and databases can be configured without any sort of user input. They are attached with this file but can also be found at: `https://github.com/horatiuluci/sysadmin-project`

3.6 Maintenance

3.6.1 Removing Vagrant locally

- Linux :
  ```
  rm -rf /opt/vagrant
  rm -f /usr/bin/vagrant
  ```
- macOS :
  ```
  rm -rf /opt/vagrant
  rm -f /usr/local/bin/vagrant
  sudo pkgutil -forget com.vagrant.vagrant
  ```
- Windows :
  Simply delete using Windows' programs section of the control panel.

You can remove the user data by deleting the `~/.vagrant.d` directory if you're on Linux or macOS, and `C:\Users\Username\.vagrant.d` if you are using Windows.

3.6.2 Moving out of Vagrant's services

Moving out of Vagrant is extremely easy as well, considering the Vagrantfiles are written in Ruby. They can be re-purposed and re-used, should we decide to move out of these services.

4 Conclusions

All the issues listed in the analysis report have been solved using Vagrant.

- **Every line of code developed by our company can now be executed by all of our employees regardless of their operating system.** The Vagrantfile will take care of provisioning a virtual machine with correct specifications for the code it's supposed to run. This will increase productivity as well as the quality of the development and testing processes.
- **Adding support for new operating systems requires no work from the system administration team anymore.** Developers are more independent because creating development and testing environments for new operating systems is as easy as changing a line in the Vagrantfile, and in the process, speeding up the development and testing phases.
- **Disk space costs are now significantly reduced.** Due to our company opting to track individual changes in a Vagrantfile (which is less than 1MB in size) rather than the changes in the whole virtual machine image (several GB in size) our company is saving significant costs on the hosting system. Furthermore, these costs will remain constant as time passes, thus stopping the trend of increasing that was true until now and improving trace-ability.
- **Teams can now work better together.** It is now easier to grasp the whole picture of a project (virtual machine specifications, dependencies, network set-up) only by looking at its corresponding Vagrantfile. This will dramatically improve the valuable time it takes an experienced developer to assist in other teams' issues, thus saving company money.