Vagrant essentials

Build portable environments

Multi-machine deployment

Build multi-tenant environments

The need for multi-machines

- In the third section of this course we deployed a complete web application development environment that uses Laravel 5 as a PHP framework.
- You noticed that the environment was made up of a web server and a database server, both deployed on the same server.
- This might be ok for a small project but in larger, more complex projects more components are involved. You might want to use a reverse proxy server (like Nginx) in front of the web server to enhance performance. You might also have a Node.js server for availing some API services and so on.
- In a production environment, those services should not be on the same machine; they'd rathe be scattered on different nodes for more scalability and enhanced performance.
- Since development environments should mimic production environments as much as possible, there should be a way by which you can disperse your development environment on several VMs using Vagrant. Fortunately, there is.

LAB: divide the LAMP environment

- In this lab we are going to break up our LAMP development environment to be:
  - Web server, PHP (with prerequisites), and Laravel 5 on one node
  - MySQL database on the second node.
- We are going to do this using one Vagrantfile. After completing this lab, a single vagrant up command should bring two machines up: the application server and the database server.
- Let’s have a look at how the Vagrantfile should look like.

- **To multiple machines to the Varantfile, simply add a stanza as the following before the last line (before end):**

```
 config.vm.define "app" do |app|
  app.vm.provision "shell", inline: "echo I am the app server"
end
config.vm.define "db" do |db|
  db.vm.provision "shell", inline: "echo I am the database server"
end
```

- Notice that you can add whatever configuration options after a `config.vm.define` stanza. However, any configuration lines on the outside will be executed first before moving to the ones defined inside the stanza (the outside-in way). Note also how we replaced “config” with the new object app or db depending on the machine. This restricts any configuration to the scope of this machine only.

- Accordingly, we should be placing a separate, different config.vm.provision shell script (or a Chef recipe or whatever configuration management system you favor) into each machine to deploy the necessary components.

- Changes must be made to the provisioning script of both machines to configure Laravel to communicate with the database on a different host (instead of the default localhost), and also on the database machine to enable MySQL to receive communication from outside 127.0.0.1. The necessary scripts can be found in the project files of this section.

- It’s also a good idea to set each machine to be using a private or public network mode with a pre-assigned IP address; so that you can configure the DB settings in the app server accordingly.

Working with your machines

- You now have two machines in the directory: app and db. So how can you manage them?
- All the commands that you learned so far for controlling the machine state are still valid but with one minor difference: you would add the machine name after the command. For example, to halt the app machine you'd run the following command: `vagrant halt db`
- Forgetting to add the machine name after the command will make this command apply on all the machines. For example, `vagrant halt` alone will halt both the app and the db machines.
- You can even limit the command execution to only a subset of the machines (if you have more than two nodes). For example, vagrant suspend machine1 machine3 machine5 will ignore machines 2 and 6 (if they exist).
- If you want to have a general idea about the state of the machines, just issue `vagrant status` with no machine name. Adding the machine name will give the status of that specific machine.