THE EXPERT'S VOICE® IN OPEN SOURCE

# Pro Vagrant

*MASTER THE CREATION AND CONFIGURATION OF VIRTUAL DEVELOPMENT ENVIRONMENTS*

Wlodzimierz Gajda

Apress®

*For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.*


Apress®

# Contents at a Glance

CHAPTER 1



# Getting Started with Vagrant

This chapter introduces Vagrant, the tool you want to master. We start with a short description of Vagrant: its role and why you should learn it in the first place. To fully appreciate its benefits, we will analyze the current approaches to set up a development environment for a web application. After summarizing the problems with traditional methods, we will discuss how virtualization can help resolve various issues. With a clear image of the virtual approach, we will demonstrate the simplicity offered by Vagrant. Then we will proceed with the installation of the software; finally, you will take a look at the built-in manual and the Vagrant documentation. The pros of using Vagrant that you will learn here should convince you to give Vagrant a try.

## What Is Vagrant?

Vagrant is a tool that simplifies the workflow and reduces the workload necessary to run and operate virtual machines (VMs) on your computer. It also does the following:

- Offers a very simple command-line interface to manage VMs
- Supports all major virtual solutions: VirtualBox, VMWare, and Hyper-V
- Supports most popular software configuration tools, including Ansible, Chef, Puppet, and Salt
- Facilitates procedures to distribute and share virtual environments

Vagrant shines when it is used for web applications but is not restricted to this particular task. It should be considered a general-purpose tool to work with VMs. If you have any previous experience with virtualization, you will be amazed by the simplified workflow offered by Vagrant. If virtual solutions are something that you have not tried yet, you will be surprised by the opportunities Vagrant offers.

Vagrant provides a simple and uniform command-line interface. You can think of it as the way to standardize the workflow when using VMs. No matter which virtualization solution you use or how intricate your virtual environment might be, Vagrant will boot your VM (or VMs) with just one command:

```
$ vagrant up
```

This single command is the most fundamental and important feature of Vagrant and it is all your team members need to know to proceed with their work. You might be working with VirtualBox, VMWare, Hyper-V, or any other virtual platform. Your application can be written in arbitrary languages and frameworks, such as Ruby/Ruby on Rails, Python/Django or PHP/Symfony. You might use arbitrary operating systems (OSs) to deploy your application: Linux, FreeBSD, or Windows. All these details are of no

importance because (thanks to Vagrant) the complete virtual environment can be brought to life with just one command:

```
$ vagrant up
```

If you were to run the virtual development environment manually — without Vagrant's help, that is — you would have to follow these steps:

1. Download the VM image.
2. Start the VM.
3. Configure the VM's shared directories and network interfaces.
4. Maybe install some software within the VM.

With Vagrant, all these tasks (and many more) are automated. The command `$ vagrant up` can do the following (depending on the configuration file):

- Automatically download and install a VM, if necessary
- Boot the VM
- Configure various resources: RAM, CPUs, network connections, and shared folders
- Install additional software within the VM by using tools such as Puppet, Chef, Ansible, and Salt

---

■ **Note** Vagrant is a tool that provides a simple and unified command-line interface to work with VMs managed by well-known virtualization solutions such as VirtualBox, VMWare, and Hyper-V.

---

Although Vagrant is a general-purpose tool and can be used in many different ways, the easiest way to learn about its features is to adopt it for web development (because of the nature of web applications, which have front ends and back ends).

The next three sections discuss the background knowledge you need about the following:

- The nature of web applications and the problems that accompany setting up a development environment for a web application
- Traditional methods for setting up development environments for web applications
- Virtualization

# Client/Server Paradigm and its Aftermath

The dual nature of web applications imposes difficulties that can turn the task of setting up a development environment into a chore. Web applications work on the basis of the client/server paradigm, using the HTTP protocol for communication purposes. When you run the application, it consists of two communicating parties: the client, which is the process that sends requests; and the server, which is the process responsible for responding.

The very nature of the application is to enable the client and server to be run on different machines, usually equipped with different hardware and OSs. The client is the process that is being executed on your laptop, and the server is the process that runs on a remote machine accessible through the network. This setting is depicted in Figure 1-1.

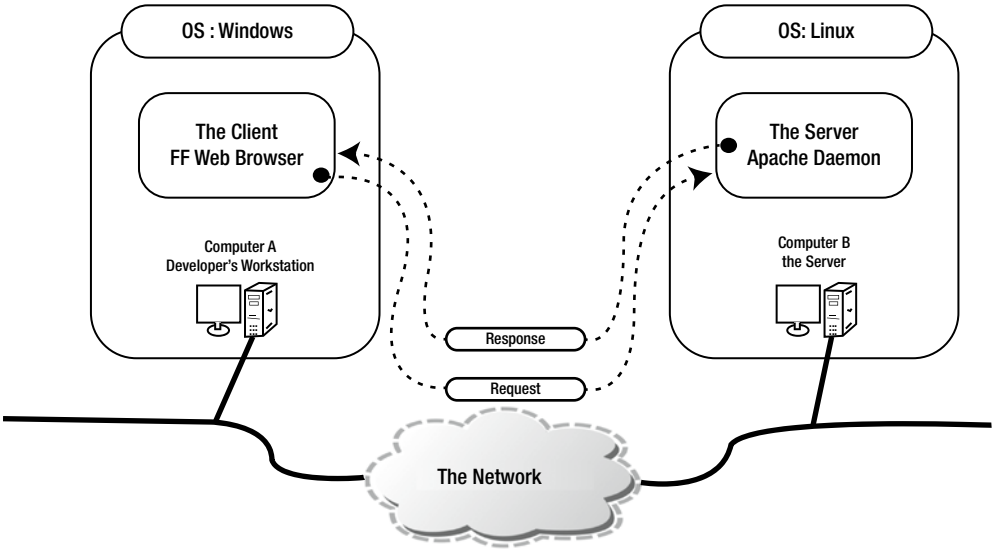


***Figure 1-1.*** *A web application in action*

Figure 1-1 shows two different computers using two different OSs: Windows and Linux. The client is the Firefox web browser running on Windows; the server is the Apache daemon executed on the Linux machine. The consequence of the client/server paradigm is that the application consists of two different pieces of code: the code that runs on the client and the code that runs on the server. A developer who works on the application needs (at least sometimes) to run and access the client-side code as well as the server-side code.

---

■ **Note** The two ingredients of a web application are commonly referred to as the *front end* and *back end*. The front end is executed within a web browser on the client side; the back end runs on the server side.

---

# Traditional Approach to Setting up a Developer Environment

There are two ways to set up a development environment. In the first, both the client and server processes are run on the same machine; in the second, the client and server run on different computers.

The first solution is shown in Figure 1-2. The developer installs all the software on the workstation, and the front and back ends run locally. The source code of the application is stored on the developer machine's hard drive.

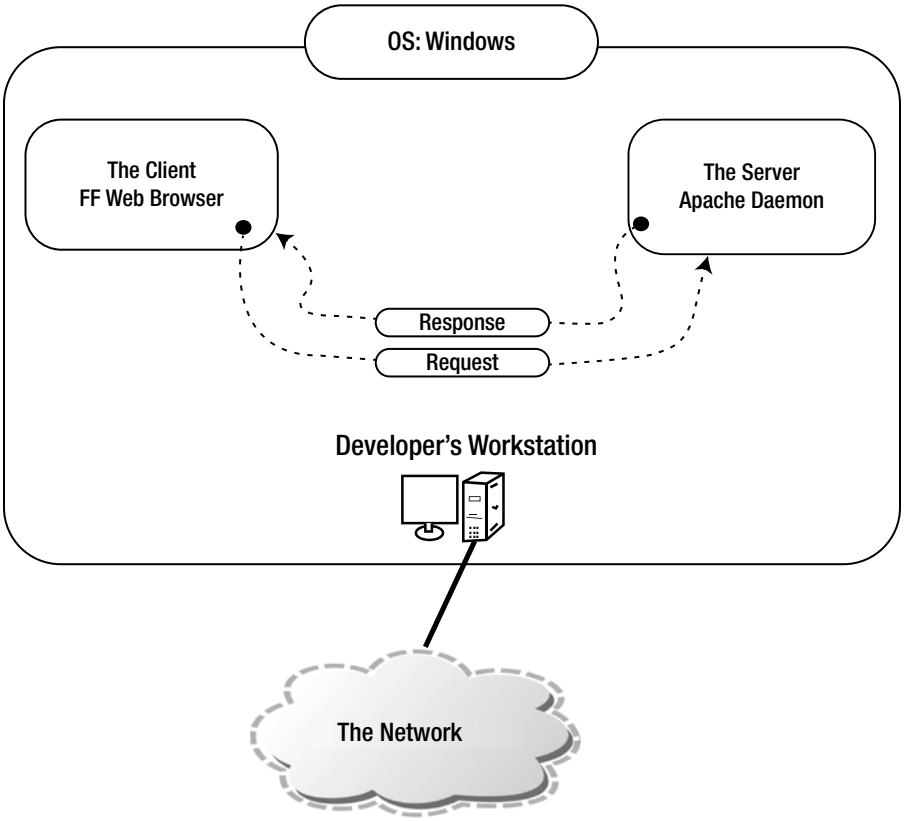


***Figure 1-2.*** *Development environment: the client and the server run on the same machine as the processes of the same OSs*

The second approach, the one in which the client and the server run on different machines, imitates the way the deployed application is executed by end users. The client-side software runs on the developer's workstation, and the server-side software runs on a remote machine. The source code of the application is stored on remote machine and, depending on preferences, the developer can do the following:

- Synchronize the files from within the integrated development environment (IDE), as shown in Figure 1-3.

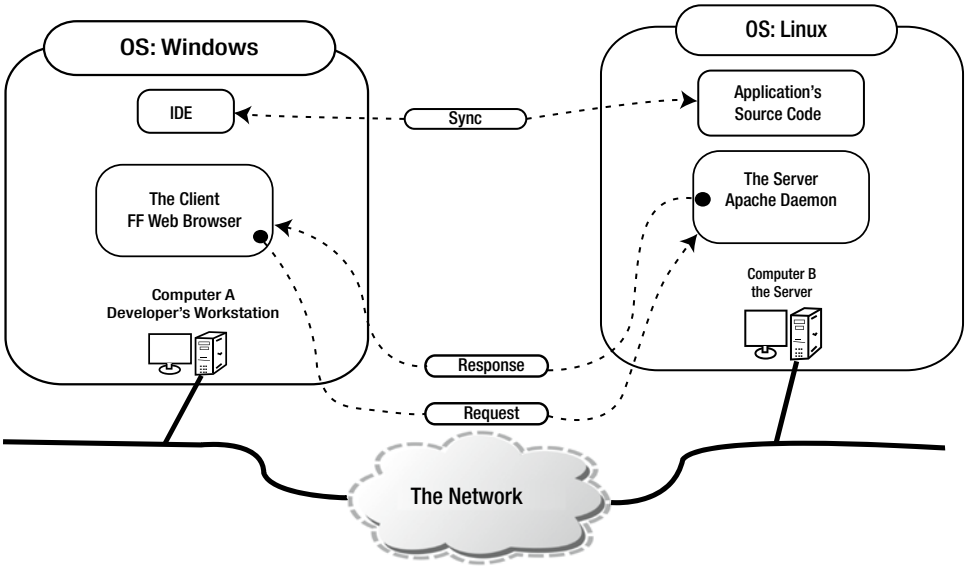


***Figure 1-3.*** *Development environment: the client and the server run on different machines; the application's source code is synchronized within the IDE*

- Access the source with the terminal (e.g., SSH/vi), as shown in Figure 1-4.

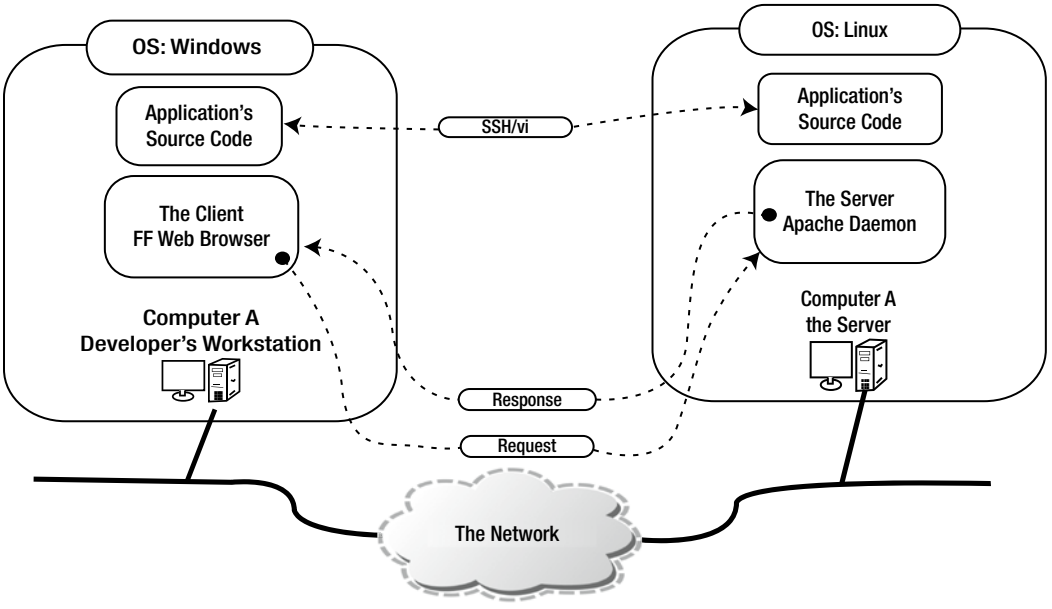


***Figure 1-4.*** *Development environment: the client and the server run on different machines; the source code is accessed via a terminal session SSH/vi*

- Emulate local access with drive mapping (see Figure 1-5)

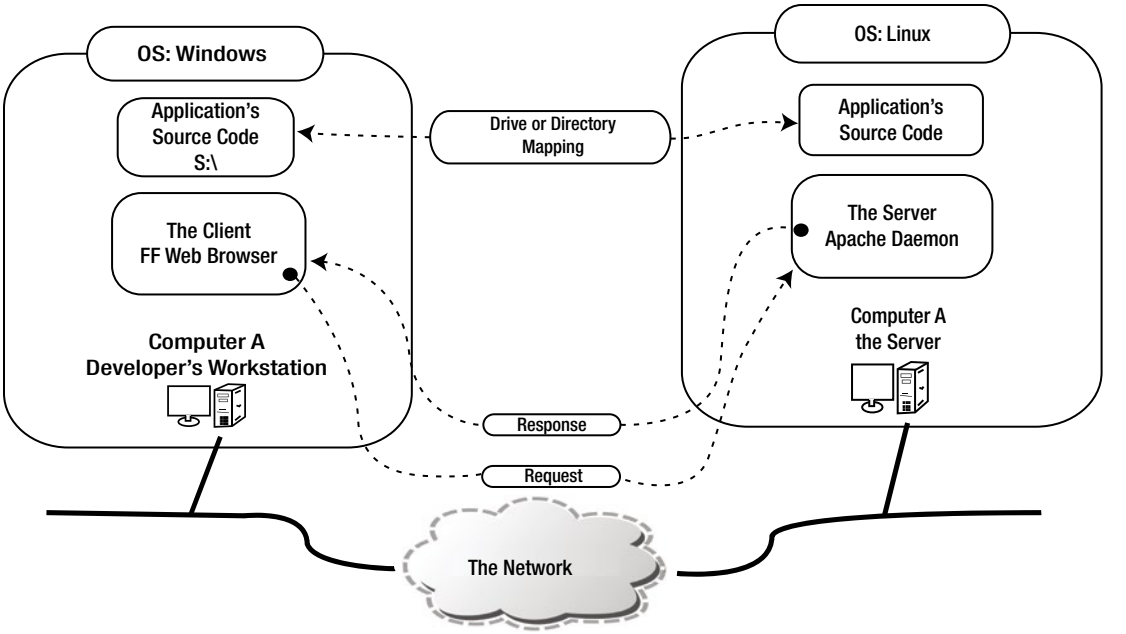


***Figure 1-5.*** *Development environment: the client and the server run on different machines; the source code is accessed through a mapped drive or directory*

The most evident shortcomings of the first solution shown in Figure 1-2 are the following:

- The installation and updates of the server-side software (for example, an HTTP server or SQL Server) have to be done manually by every developer. From the company's point of view, this can be an enormous waste of developers' time.
- When all developers use their favorite OS, it can be difficult to set up the identical environment on all platforms. Some developers might be unable to reproduce the complete environment on their machines.
- Because the installation is done by hand, environments used by developers might differ, which can lead to "it works on my machine" problems.
- New members of the team have to spend time setting up their environment. On a global scale, this onboarding is an unnecessary waste of time and resources.
- If the OS of the server is different from the one installed on the developer's machine (which happens very often), the development environment always differs from production because it runs software compiled for the different platform.
- All members of the team need specialized knowledge about the software necessary for the application to run. Even if they are responsible only for the design of the application and don't take part in development, they still have to know how to install and configure everything. So it can be difficult for designers to install all packages required by the back end.

- Developers who work on multiple projects at the same time can have problems because one development environment is shared by different projects. For example, one project might be designed to work under settings (e.g., PHP 5.3) different from other project (e.g., PHP 5.5). The projects are not separated from each other: they share compilers, interpreters, and tools installed on the host system.
- With multiple projects, the procedures to start or shut down necessary daemons and services are usually project-dependent. For every project, you might need to run different commands or start different services to run the application.

The scenarios shown in Figures 1-3, 1-4, and 1-5 have the following disadvantages:

- They all require a network connection to work on an application; the performance of the server-side software influences the work of every developer. When the network is down, all the developers are idle.
- Some companies can apply restrictive security policies and forbid access to the servers from the outside world. Thus, development environments can't be accessed by collaborators working remotely;

Now that you understand the possible drawbacks that accompany the traditional approach, let's look at some virtual solutions.

# Virtualization to the Rescue

Virtualization helps to exploit the best features of both previous solutions while eliminating drawbacks at the same time. When working with VMs, we install both front end and back end components of the application on the same machine: the developer's computer. The client's processes are managed exactly as before: as regular processes in the native OS of the developer's workstation. The server-side software, however, runs within a VM. Both the front and back ends run at the same machine, but the solution closely imitates the production settings, thanks to virtualization (see Figure 1-6).

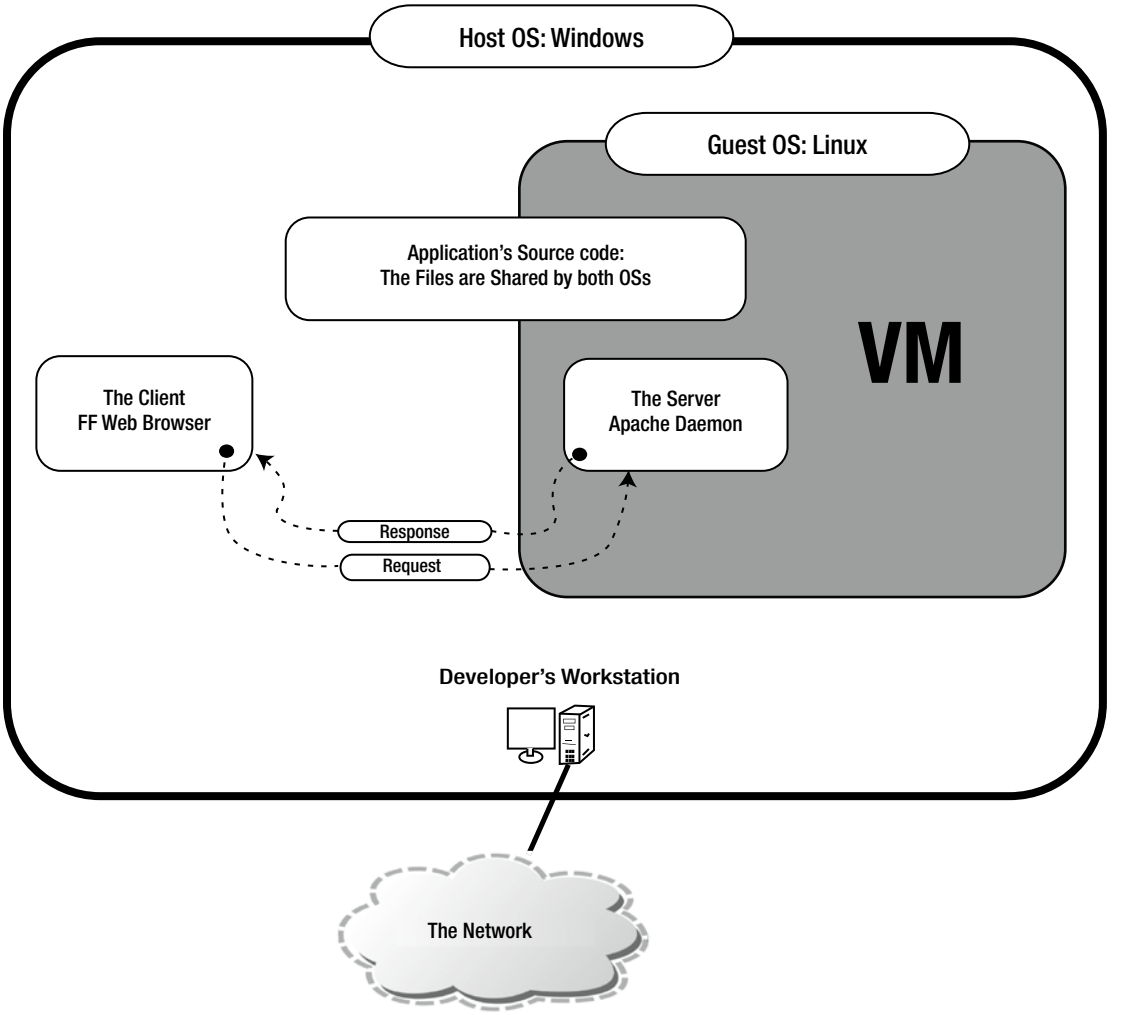


***Figure 1-6.*** *Development environment using a VM*

When you work in the system shown in Figure 1-6, you are dealing with two (usually different) OSs labelled as *host* and *guest*. The original OS of the developer's machine will hereafter be referred to as *host OS* and the VM's OS will be called the *guest OS*. In Figure 1-6, the host OS is Windows and the guest OS is Linux.

---

■ **Note** The host OS is the system that boots when you power on your computer. If you work on Mac OS X, your host OS is OS X. When you work on a laptop running Windows 7, your host OS is Windows 7. It should also be obvious that there is only one host OS.

The guest OS is the system used by the VM. Because you can boot many VMs at the same time, you can have more than one guest at the same time.

---

To be sure, the settings shown in Figure 1-6 can be achieved with various virtualization platforms: VirtualBox, VMWare, and Hyper-V, to name the most popular. But the procedure to install, distribute, start, stop, and configure VMs is manual, so it differs depending on the solution you use. This is where Vagrant can help; it provides a very simple and uniform interface to manage VMs, regardless of the virtualization solution you use.

# Enter the Vagrant

Vagrant is open source software distributed under an MIT license. It was originally authored by Mitchell Hashimoto, and as of February 2015, almost 500 programmers have contributed to it. The source code of Vagrant is available at `https://github.com/mitchellh/vagrant`.

Vagrant sits on top of existing and well-known virtualization solutions such as VirtualBox, VMWare Workstation, VMWare Fusion, and Hyper-V; and provides a unified and simple command-line interface to manage VMs. This architecture is shown in Figure 1-7.

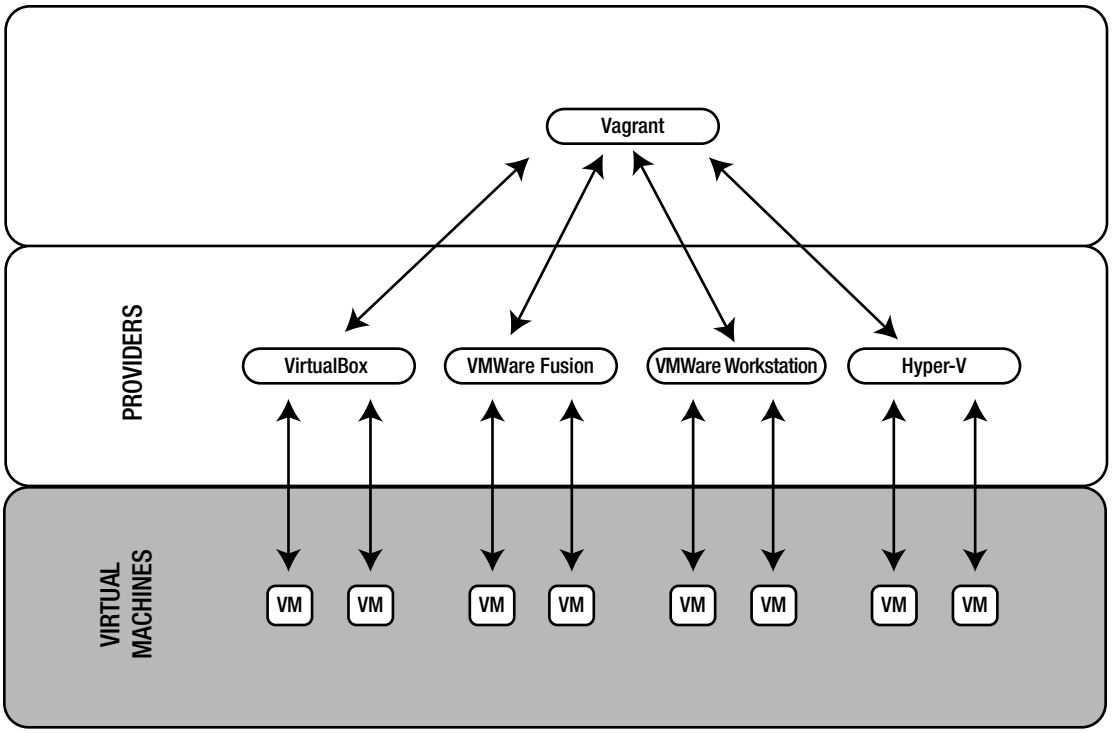


***Figure 1-7.*** *Layered architecture of Vagrant using providers*

To quote its home page, Vagrant is a tool that helps to "create and configure lightweight, reproducible, and portable development environments."[1]

[1]The quote comes from the Vagrant home page: `http://vagrantup.com`.

In Vagrant's terminology, the underlying virtualization solutions are called *providers*. To work with Vagrant, you have to install at least one provider. The official list of providers includes the following:

- VirtualBox
- VMWare (Fusion and Workstation)
- Docker
- Hyper-V

---

■ **Note** In Vagrant terminology, a *provider* is a software virtualization solution such as VirtualBox, VMWare Fusion, or VMWare Workstation.

---

Although it is technically possible to work with virtual development environments without using Vagrant, doing so requires a fair share of manual labor. Vagrant automates lots of things and exposes the magic `$ vagrant up` command to developers. In many cases, this single command is all developers need to boot the application, no matter how complicated the server-side settings are.

---

■ **Note** Vagrant sits on top of providers and exposes a very simple, consistent, and powerful command-line interface to all of them. This interface consists of commands such as `$ vagrant up`, `$ vagrant halt`, and `$ vagrant destroy`.

---

# Vagrant Rulez!

How does Vagrant outclass traditional approaches? The first and most eye-catching feature of Vagrant is that the complete development environment is created with a single command: `$ vagrant up`. You don't need to know the internals of the server-side ingredients. The command boots the VM or VMs, and sets all the properties: RAM size, shared directories, networking, and so on.

The developer is no longer responsible for the installation of the server-side software; although it is executed on the developer's machine, there is no struggle with the installation. The burden of preparing and configuring server-side software is moved from developers to system engineers. The developers use a prefabricated VM they may treat as a black box that can be turned on and off with Vagrant commands. In general, it takes a minute or two to get the development environment up and ready for development, although that can depend on many factors.

As you look deeper into the internals of Vagrant operation, note that every project runs within its own VM. Thus, all the projects are separated, and each can use whatever software stack is appropriate.

One project can use a Linux/Apache/MySQL/PHP (LAMP) stack; the other can use Ruby on Rails with Nginx and PostgreSQL (Postgres), all running on FreeBSD. At the same time, even though VMs are truly isolated, you still use only one and the same interface to control (to boot or shut down) the environment. This process simplifies the workflow enormously, and the workflow is no longer project-dependent.

Project isolation can be seen as breaking the ties between your workstation and the project. When working with Vagrant, you can replace your computer or its OS without worrying about the software necessary to run the project. Once you install the provider and Vagrant, the project should run exactly as it did before the changes took place. You don't have to search for and install all the libraries used in any of the projects on which you are collaborating.

Yet another aspect of using virtualization is that developers do not have to install software that expose resources, such as network ports, to the outside world. With Vagrant, you are using a sandboxed solution that by default is not accessible by the outside world. You can ease those restrictions, of course, but it is done explicitly and in a consistent way for all the resources. And, of course, the configuration of exposed resources is done by your sysadmin — the person who can be relied on when it comes to security. Thus, the overall security of your workstation is improved.

From a company perspective, problems with different versions of server-side software used by different developers will vanish. Everyone in the organization uses exactly the same virtual environment prepared by a system engineer. This not only leaves no place for "works on my machine" issues but also facilitates the creation of development environments that are identical to production settings — the system engineer can use the same recipes to prepare the development and production configuration. Moreover, the procedure of updating the server-side software on all the machines used by developers is also painless — all that has to be done is to distribute a new boxed VM. (In fact, it is the main reason I loved Vagrant right from the very moment I learned how to use it.) More on this in the following "Vagrant for Trainers, Instructors, and Teachers" section.

When you combine all the features discussed above with the fact that Vagrant is an open source, free software that works seamlessly across many platforms (including Windows, Linux, and OS X), you will see that the advantages of Vagrant over the traditional approach are overwhelming.

To summarize, let's enumerate Vagrant's most important features:

- Vagrant has an easy workflow: one command (`$ vagrant up`) gets the development environment ready.
- The time needed to bring the development environment to life, even though it depends on many factors, is only a minute or two.
- The developer can change the workstation or replace the OS without having to install software. There are only two packages to be reinstalled: a provider (e.g., VirtualBox) and Vagrant.
- The developer is no longer responsible for installing server-side software.
- The VMs used by all developers are prepared by system engineers.
- Everyone uses exactly the same server-side software.
- Every developer can create, destroy, and re-create virtual environments within a couple of minutes.
- Vagrant facilitates easy updates of server-side software on developers' machines.
- Access to all the resources, such as network ports, is restricted by default.
- The explicit rules that expose a workstation's resources are defined by a sysadmin who is responsible for preparing the sandboxed VM. Thus, the security of all the machines used by developers can be managed globally in a consistent way.
- You can quite easily get 1:1 (or nearly 1:1) mapping across development, testing, and production environments.
- Virtual development environments are created on a per-project basis (all the projects run in isolated cocoons).
- Every project can use arbitrary packages (e.g., one project can use Linux/PHP 5.3; another can use CentOS/Rails).
- One workflow is used for all projects, no matter what virtualization platform, guest OS, language, framework, or server-side software is used.

- Developers don't need a network connection to proceed with their work.
- Vagrant works on all major platforms, including Windows, Linux, and Mac OS X.
- Vagrant is open source and free.
- Vagrant has detailed, well-organized, and clear documentation.
- If you find it necessary, you can extend Vagrant's features with plug-ins.
- The popularity of Vagrant is growing exponentially; it is quickly becoming the tool of choice in many companies.

# Disadvantages of Vagrant

The most important disadvantage of using Vagrant is efficiency; the workstations used by developers have to be powerful enough to work with the chosen provider. Moreover, the workstation used by the sysadmin to prepare boxed VMs needs to be even more superior; otherwise, the process of preparing and testing the boxed VMs will be nerve-wracking. When I started preparing boxed VMs, I had to upgrade my laptop to Mac Book Pro.

These constraints don't constitute the complete list. Some combinations of host OSs and providers are known to yield ineffective outcomes. For example, the access time to the shared directory of VMs created with the VirtualBox provider is quite slow. When your host OS is Linux or Mac OS X, you can bypass this issue using a network file system (NFS), but there is no universal solution if you use Windows. Although a workaround is to avoid using shared folders and to use deployment procedures to access the storage in VM instead, it requires additional effort from developers.

# Vagrant for Trainers, Instructors, and Teachers

I have loved Vagrant from the very first day I learned how to use it during my classes. It has really changed the way I work. I am the only one who bears the responsibility for creating development environments, so I am the only one who has to deal with installing software, updating system libraries, and so on. I don't have to explain how to install the software any more. Last year, when I introduced Vagrant on a mass scale (for all my classes), I finally stopped spending hours dealing with the installation of all the server-side stuff.

Right now, no matter what platform is used in the computer laboratory or what OS is installed on students' laptops, all I have to do is distribute boxed VMs and tell the students how to use them, which can take as little as five minutes. And, of course, the procedure doesn't change when we change language or framework; once learned, it can't be forgotten.

Exactly the same arguments apply for the training I provide for commercial companies. To proceed with training without any hassle, I need a boxed solution available on the desktop computer of every participant. When that is done and verified, everything goes smoothly. Before Vagrant, there were always problems that could be classified as "doesn't work on my machine" issues.

I also find Vagrant indispensable when I want to take a look at applications or solutions. Gerrit, Gitlab, or Jenkins — you don't have to mess with your desktop any more just to try them. (This is exactly what I suggest to my readers in my book *Git Recipes*.)[2] To recompile git or experiment with gitolite, you can use VM managed by Vagrant/VirtualBox. Working this way, I can provide a single tutorial to be followed by all readers, no matter what platform they work on. Were the readers to install the software on their computers, the instructions would be OS-specific, and it would be much more time-consuming to uninstall everything because the process would be manual.

[2]Gajda, Włodzimierz. *Git Recipes: A Problem-Solution Approach*. Apress, 2013.

# Installing the Software

During this course you will need three packages (all are free and open source):

- Git
- VirtualBox
- Vagrant

## Git

Git is a distributed version control system that has gained enormous popularity since its birth ten years ago and is the *de facto* standard for open source projects. You have probably already had some experience with git; if not, don't worry — I will provide you with all git-related information you may need.

To install git, visit `http://git-scm.com` and download the release for your platform. During the installation, leave all the available options with their default values. Then run the command line and type this command, which should output the version of git:

```
$ git --version
```

---

■ **Note** If you work on Windows, don't use the standard Windows command line shipped with the system. Run the `git bash` shell instead. Working this way, you can use the commands exactly as they appear in the book without any modifications.

---

## VirtualBox

The home page of VirtualBox is `https://www.virtualbox.org`. Go to the Downloads section and fetch the latest version available for your platform. (During the writing of this book, I used VirtualBox 4.3.22, but that is not mandatory.) Proceed with the typical procedure you use to install software on your platform. The main screen of the installation program on OS X is shown in Figure 1-8.

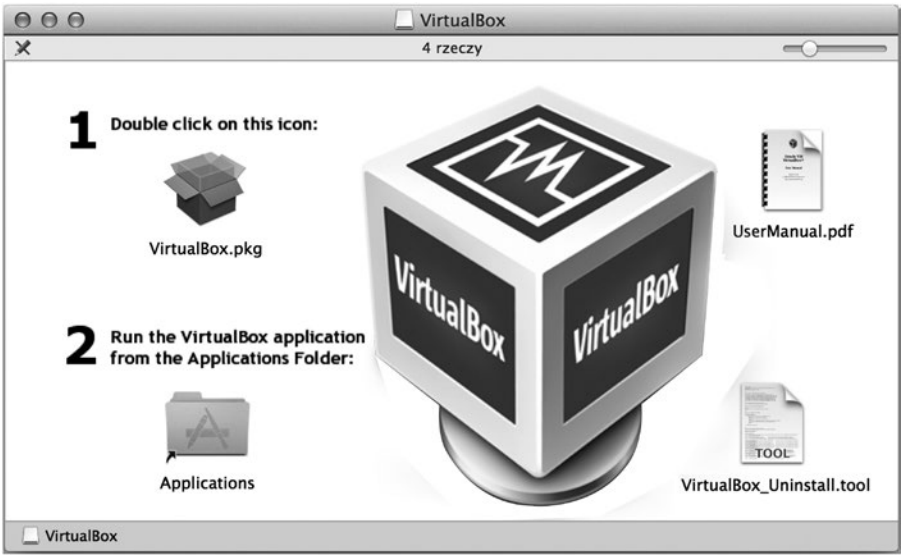


***Figure 1-8.*** *Main screen of VirtualBox installator on OS X*

Recent VirtualBox distributions are known to be quite easy to install, but in case of any problems, the VirtualBox manual can come in handy: `www.virtualbox.org/manual/ch02.html`.

If you have an older version of VirtualBox on your workstation, remove it prior to installing a newer version. And because VirtualBox installs low-level drivers, you might have to restart the system after the installation or removal of VirtualBox (at least on Windows).

## Vagrant

To install Vagrant go to `www.vagrantup.com/downloads.html` and download the distribution for your system. Then proceed with the installation of the downloaded package. The main window of the Vagrant installator is shown in Figure 1-9.


***Figure 1-9.*** *Main screen of the Vagrant installator*

## Check the Installation

How do you verify that Vagrant is ready to work? Just open your command line[3] and execute one of the commands shown here:

```
$ vagrant --version
$ vagrant -v
```

The command should produce output similar to the following:

```
Vagrant 1.7.2
```

I used Vagrant 1.7.2 during the work on this book. I usually prefer to work with the latest software version and suggest that you to do the same (although it isn't obligatory).

---

■ **Caution** If you work on Windows and your account contains non-Latin letters with diacritical marks, such as óąśż, Vagrant doesn't work. You have to create and use a new account that consists only of Latin characters.

---

[3]If you work on Windows. I strongly advise you to use the `git bash` prompt that is shipped with git.

# Basic Vagrant Configuration

By default, Vagrant stores boxed VMs in a current user's home directory in the `~/.vagrant.d/` folder. Because boxed VMs are images of the complete guest OS, they are quite large. It is not uncommon to have a box consuming 1 – 2 GB. And because the boxes are used only as templates, they can be shared by all users who work on the given computer without any negative consequences. By sharing installed Vagrant boxes, users save hard disk drive (HDD) space and gain immediate access to all boxes without having to download them again. This can be important when workstations are used by many people, as in a computer laboratory, for example.

The location of the Vagrant home directory can be redefined with the `VAGRANT_HOME` environment variable. You can change its value by executing the following command in bash:

```
$ export VAGRANT_HOME=/some/shared/directory
```

When working on Windows, you can set a new environment variable using System/Properties/Advanced System Settings/Environment Variables dialog box.

---

■ **Note** The list of other environment variables used by Vagrant can be found at `https://docs.vagrantup.com/v2/other/environmental-variables.html`.

---

# Documentation

Sooner or later, you'll need other sources of information. You can use the Vagrant built-in manual, which comes very handy, indeed. Access it with the following commands:

```
$ vagrant
$ vagrant -h
$ vagrant --help
```

When you run one of them, the output is similar to the one shown in Figure 1-10.

```
Terminal — bash — 99×38
~$ vagrant
Usage: vagrant [options] <command> [<args>]

    -v, --version                    Print the version and exit.
    -h, --help                       Print this help.

Common commands:
     box             manages boxes: installation, removal, etc.
     connect         connect to a remotely shared Vagrant environment
     destroy         stops and deletes all traces of the vagrant machine
     global-status   outputs status Vagrant environments for this user
     halt            stops the vagrant machine
     help            shows the help for a subcommand
     init            initializes a new Vagrant environment by creating a Vagrantfile
     list
     login           log in to HashiCorp's Atlas
     package         packages a running vagrant environment into a box
     plugin          manages plugins: install, uninstall, update, etc.
     provision       provisions the vagrant machine
     push            deploys code in this environment to a configured destination
     rdp             connects to machine via RDP
     rebuild
     reload          restarts vagrant machine, loads new Vagrantfile configuration
     resume          resume a suspended vagrant machine
     share           share your Vagrant environment with anyone in the world
     ssh             connects to machine via SSH
     ssh-config      outputs OpenSSH valid configuration to connect to the machine
     status          outputs status of the vagrant machine
     suspend         suspends the machine
     up              starts and provisions the vagrant environment
     version         prints current and latest Vagrant version

For help on any individual command run `vagrant COMMAND -h`

Additional subcommands are available, but are either more advanced
or not commonly used. To see all subcommands, run the command
`vagrant list-commands`.
```

***Figure 1-10.*** *The output of the* `$ vagrant` *command describes basic usage and lists all the subcommands*

The Vagrant interface consists of commands such as these:

```
$ vagrant box
$ vagrant package
$ vagrant up
```

During the course, we will discuss all these commands. Some of them have subcommands such as the following:

```
$ vagrant box add
$ vagrant box list
$ vagrant box remove
```

These subcommands will also be explained in the following chapters. Right now, remember that to print the syntax of any of the commands and subcommands, you have to use the `--help` or `-h` switch, as shown in these examples:

```
$ vagrant box -h
$ vagrant box --help
$ vagrant box add -h
$ vagrant box add --help
```

When the manual is not enough, you can access Vagrant documentation online: https://docs.vagrantup.com/v2/getting-started/. The documentation is very clear and includes in-depth descriptions of all Vagrant features.

# Summary

In this chapter, I wanted to give you a clear picture of the advantages that Vagrant offers. When working with Vagrant, you move the responsibility for setting development environment from developers to one central point — a sysadmin, for example. Developers run both client-side and server-side software of the application on their workstations. Server-side services such as a database server or HTTP server are executed within a virtual system that can mimic the system used on a production server. The VM can be seen by developers as a black box. All developers need to know about server-side software is reduced to a couple of commands that turn the environment on and off, such as the following:

```
$ vagrant up
$ vagrant halt
```

Moreover, all projects that developers might be working on are separated, and each can use arbitrary server-side solutions. The projects do not overlap nor do they collide with each other.

The Vagrant's workflow not only facilitates easier onboarding of new developers but also simplifies the task of updating the environment used by all developers. This is especially attractive for training and teaching purposes.

Because Vagrant really changed the way I work, I sincerely recommend it and promote it to everyone.

# In the Next Chapter, You'll Learn . . .

There's nothing better than hands-on experience. All the discussions from this chapter will be more understandable after you run the examples awaiting you in Chapter 2. In just a few minutes, you will run four web applications, each written in different language:

- JavaScript/AngularJS
- Python/Django
- Ruby/Ruby on Rails
- PHP/Symfony

Even without any knowledge of these languages and frameworks, with Vagrant you can still run them all on your computer.

# Reading List

If you're interested in reading more about Vagrant, the first source of information should be its documentation: https://docs.vagrantup.com/v2/getting-started/.

For a more detailed, extensive, and thorough introduction, I strongly recommend *Vagrant: Up and Running*, by Mitchell Hashimoto.[4] Hashimoto is Vagrant's author and project leader, and his book is an excellent source of information for novices and advanced Vagrant users alike.

[4]Hashimoto, Mitchell. *Vagrant: Up and Running*. O'Reilly, 2013.

You might also need more information about your provider. For VirtualBox, see www.virtualbox.org/manual/.

For Vagrant's untypical behavior, the list of current issues can be helpful: https://github.com/mitchellh/vagrant/issues.

For basic introduction to web applications and the client/server model, see the following Wikipedia entries:

- http://en.wikipedia.org/wiki/Web_application
- http://en.wikipedia.org/wiki/Client-server_model

# Test Yourself

1. What is Vagrant? Define it in one sentence.
2. What is the most important Vagrant command?
3. What is the URL of the Vagrant home page?
4. What difficulties are caused by the client/server paradigm for developers?
5. What traditional approaches to setting development environments do you know?
6. Explain the Vagrant approach to setting up the development environment.
7. Define the terms *host* and *guest*.
8. Define the term *provider*.
9. Name the providers supported by Vagrant.
10. What is the command to print the Vagrant version?
11. What is the command to print the Vagrant built-in manual?
12. How do you list Vagrant commands?
13. How do you print the manual for one of the subcommands?
14. Where does Vagrant store boxes? Why should you want to change this location? How can you do it?

CHAPTER 2



# Four Web Frameworks in Four Minutes

Chapter 1 discussed some of the amazing advantages of Vagrant: how it can change the workflow of a company and how every member of the team can benefit from using it. I specifically underlined the advantages of running a web application in a virtualized environment with just one command: `$ vagrant up`.

Because the best way to understand Vagrant is to see it in action, this chapter will act as a guided tour to running four simple web applications written in four popular web frameworks:

- AngularJS (JavaScript)
- Django (Python)
- Rails (Ruby)
- Symfony (PHP)

Of course, this discussion can't be a complete tour of AngularJS, Django, Ruby on Rails, or Symfony. I will not dive into details concerning any of the frameworks because that is not the purpose of this book. I aim to prove to you that even if you are new to these frameworks and languages (and even if you don't have Python, Ruby, or PHP on your laptop), you can still run the examples with just one command. And the procedure to run each of them is almost identical.

Each of the four projects presents exactly the same web pages; each is set up and brought to life in a couple of minutes (maybe not exactly at the speed of an example per minute, as promised in the title, but much faster than can be done manually). And most importantly, the booting of the applications is completely automated. It leaves no place for mistyped commands, misconfigured services, incorrect configuration settings, and similar typos and other human errors.

Just one more notice before we proceed. Every example uses one TCP port on your host computer:

- *AngularJS project*: port 8800
- *Django project*: port 8000
- *Ruby on Rails project*: port 3000
- *Symfony project*: port 8880

If any of these ports is not available on your computer, the project will not run. (This problem will be addressed and solved in the next chapter.) Right now, let's assume that at least some of the listed ports are available. And if security is a concern, you might want to postpone the practical exercises described in this chapter until you have fully understood the implications of running the `$ vagrant up` command. This topic is described in great detail in Chapter 4.

■ **Tip** The source code for all the projects discussed in this chapter and in the entire book is stored on GitHub at http://github.com/pro-vagrant.

# Project 1: "Songs for kids" Written in AngularJS

The first project is the application titled "Songs for kids," which is written in AngularJS framework. It consists of three web pages, each of which displays the text of one song for kids. To run the project, execute the commands shown in Listing 2-1.

***Listing 2-1.*** Commands to Run "Songs for kids" in AngularJS

```
$ cd folder/with/examples
$ git clone https://github.com/pro-vagrant/songs-app-angularjs.git
$ cd songs-app-angularjs
$ vagrant up
# Run webbrowser and visit http://localhost:8800/
```

■ **Note** AngularJS is a web framework written in JavaScript. Its home page is https://angularjs.org/.

Depending on your connection, the complete procedure to get this first example running can take up to several minutes. My timing was about 2 minutes and 10 seconds.

Here is the explanation for the commands in Listing 2-1. Start by entering a directory in which you want to keep the examples:

```
$ cd folder/with/examples
```

You might have to create this folder, of course. It can be located anywhere on your hard drive (it really doesn't matter where).

The source code of the AngularJS "Songs for kids" project is available at https://github.com/pro-vagrant/songs-app-angularjs.git. To copy the sources from the GitHub server to the hard drive, use the `$ git clone` command:

```
$ git clone https://github.com/pro-vagrant/songs-app-angularjs.git
```

You now have the complete source code of the application inside the `songs-app-angularjs/` directory. Enter the project's folder:

```
$ cd songs-app-angularjs
```

You can boot the VM required to run the application with this command:

```
$ vagrant up
```

Although this command prints a lot of output, at this point I want to skip it. I prefer to postpone the in-depth discussion concerning the messages printed by Vagrant during booting until Chapter 3. Right now, I intend to convince you how simple Vagrant really is. With that goal in mind, wait until the command has finished running and proceed with the next step of this example.

---

■ **Note** An in-depth analysis of the internals behind vagrant commands such as `$ vagrant up`, `$ vagrant ssh`, and `$ vagrant destroy` is in Chapter 3. Chapter 2 is meant to be the bait that gets you hooked; no matter what the server-side solution is, you can get the project running in a couple of minutes without any specific knowledge about the back end.

---

Start your web browser and visit `http://localhost:8800/`. The web page that displays is shown in Figure 2-1.

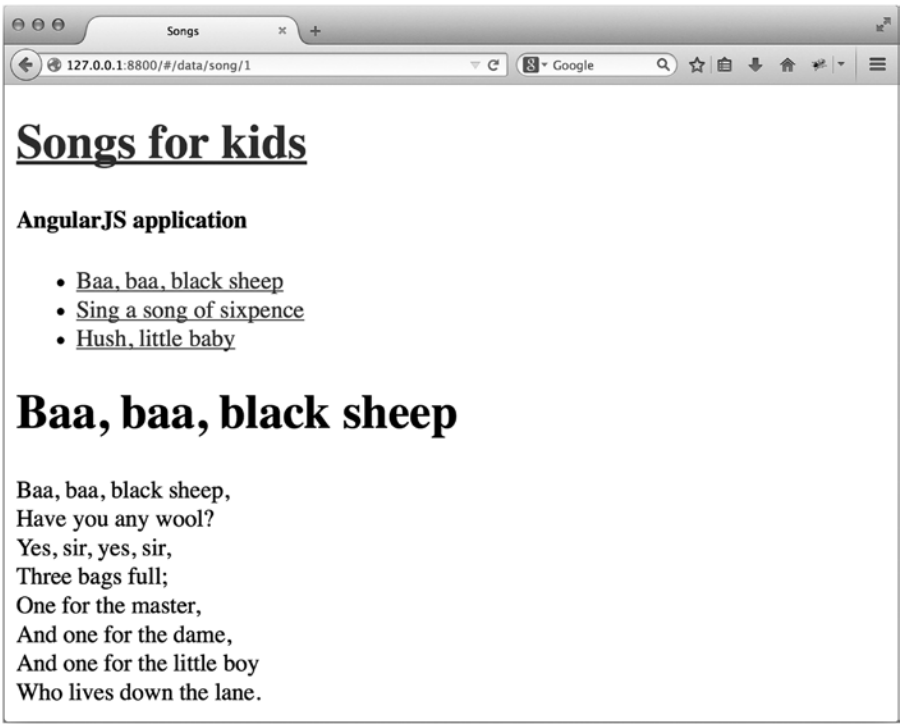


***Figure 2-1.*** *"Songs for kids" application written in Angular JS*

At this point your computer runs two OSs:

- Host: primary OS
- Guest: virtual OS started by `$ vagrant up`