17

DEPLOYMENT OF A DJANGO APPLICATION (PART 1 – SERVER SETUP)

OVERVIEW

This chapter details the initial setup of a virtual server to host a Django application. We will start by learning about the architecture of deployment and how the frontend and the application servers work together. Then, through a detailed step-by-step walk-through, we will set up an Ubuntu Linux virtual machine that can run on either a PC or macOS. Next, we will study **SSH** and its basics and we'll use the `ssh` command to connect to our newly set up server. Then, using the `sudo` command over an SSH connection, we will learn how to administer a server remotely. Toward the end of the chapter, we'll shore up our server administration skills by learning how to update and install packages with `apt`. We'll then use `apt` to install **NGINX**, **PostgreSQL**, and **Gunicorn** on our virtual server.

SERVER ARCHITECTURE

So far, we have been using the Django dev server to both execute the Python code for Bookr and serve the static and media files. We have mentioned throughout the book that the Django dev server is not suitable for use in production. It is not designed to run multiple processes or handle many users, but more importantly, it doesn't run when the **`DEBUG`** setting is set to **`True`**.

We will start this chapter by looking at the architecture used in a production web server, that is, where the functions of the Django dev server are split and handled by two different applications. There is a *frontend web server* (for example, NGINX, **`lighthttpd`**, or Apache), which receives the request from the browser, and the *application server* (for example, Gunicorn or uWSGI), which executes the Python code. The frontend web server decides how the request should be handled. If it is a request for a static or media file, then the frontend web server can handle the request itself; it can just read and send the file. If the request is for Python-specific code, then it will be forwarded to the application server to handle. The Django application then parses the URL and other HTTP data, generates an **`HttpRequest`** object using a view, and sends the response back to the frontend web server. The frontend web server then passes the response back to the browser.

The following figure elaborates on this process:

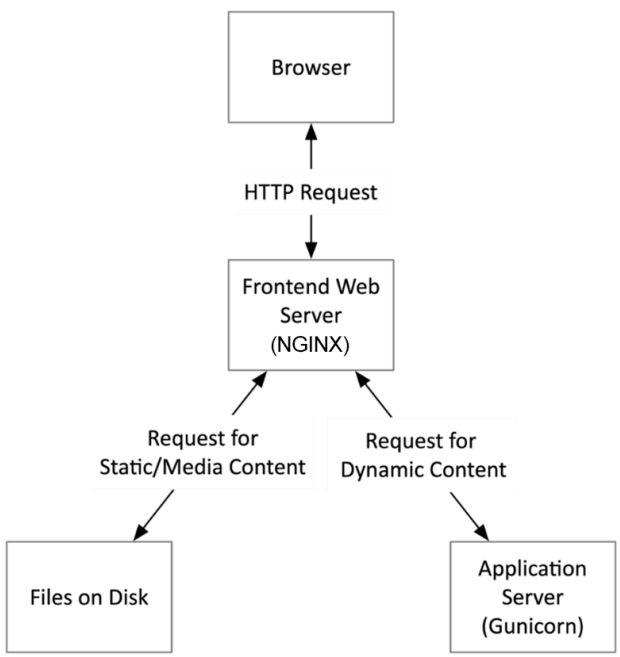

Figure 17.1: Request validation

We will discuss **Gunicorn** and **NGINX** (and explain what they are) in the upcoming sections, titled *NGINX* and *Gunicorn*, respectively. While there are plenty of options out there for servers, we're using NGINX and Gunicorn because they're fast, lightweight, and easy to configure. Usually, generating a web page's content using Python (or another programming language) takes much longer than it does to just serve a static or media file off a disk. Therefore, the performance of a website can be increased by adding more backend application servers behind a single frontend server. You could run multiple instances of the application server on the same machine as the frontend server. Due to modern hardware having multiple CPUs, this can provide speed benefits. For even better performance, you can move the application server onto its own separate server or even run multiple application server instances across different servers. The frontend server can delegate requests to backend servers sequentially or to the one that is currently least busy. The architecture of this would look like the following:

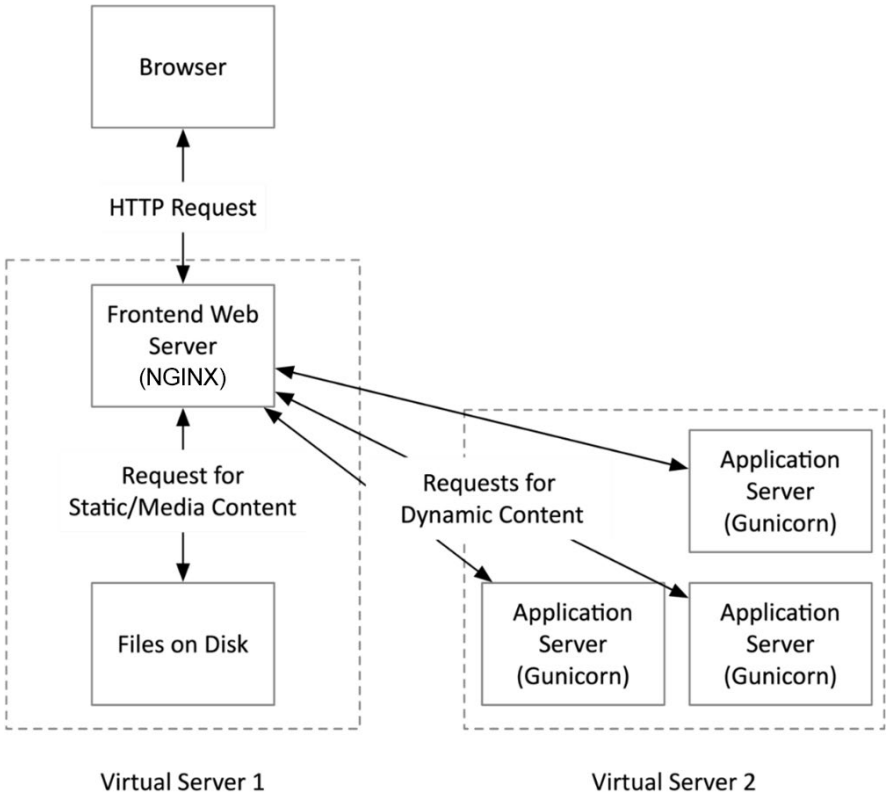

Figure 17.2: Architecture with multiple backends

Figure 17.2 shows how the server architecture is split across multiple servers. A single frontend server, **`Virtual Server 1`**, stores static files and serves them from its local disk. Requests for dynamic content are forwarded from **`Virtual Server 1`** to **`Virtual Server 2`**, which is running multiple **`Gunicorn`** instances. We could further expand this with multiple virtual servers that follow the same pattern as **`Virtual Server 2`**.

In this chapter, we will be using NGINX as the frontend server and Gunicorn as the backend (or application) server. We'll also switch to using PostgreSQL as the database server instead of SQLite. This is because SQLite is designed just for a single user and is fine for development purposes when you're running Django on your own computer. PostgreSQL is a database server designed for high loads that can scale to thousands of connections.

GUNICORN

Gunicorn is an easy-to-use application server. Its settings can be controlled with command-line flags or a configuration file. In its simplest usage, it can be run like this:

```
gunicorn bookr.wsgi:application
```

This starts Gunicorn using the Django WSGI.

> **NOTE**
>
> WSGI (often pronounced *wizz-gee*) stands for **Web Server Gateway Interface**. It describes a standard for Python to communicate with web servers. This means any WSGI Python script should be able to communicate with any WSGI server and any combination of frontend server and application server can be used together, provided they both "speak WSGI." For example, instead of using Gunicorn, we could use uWSGI. Its configuration would be different, but it serves the same purpose.

We will return to Gunicorn's configuration in *Chapter 18*, *Deployment of a Django Application (Part 2 – Configuration and Code Deployment)*.

> **NOTE**
>
> Prior to this chapter, the code and examples given would work on Windows, macOS, or Linux. In this chapter, the installation and setup guides assume an Ubuntu Linux server. This is because the paths to data and config files differ between operating systems. It is possible to run NGINX and Gunicorn on Windows or macOS too, but you will need to research how to install them yourself.
>
> You can use **VMware** or **VirtualBox** to configure a Linux virtual machine running on your computer or use a hosted virtual private server from a provider such as **Amazon Web Services** (**AWS**). If you use a virtual machine, you can shut it down when you are not working on it to free up resources on your computer.
>
> We will give instructions on how to set up an Ubuntu Linux virtual machine running inside VirtualBox on your computer, but if you're more familiar with other virtualization software or have a cloud provider that you would like to use, you can set up a virtual server in that environment instead. Provided you create a virtual server running the same version of Ubuntu, the instructions should be the same after setup is complete.
>
> Throughout this chapter, we will refer to this virtual machine as the "virtual machine." If a command is to be run on it, we will say "Run command … on your virtual machine" or "On your virtual machine, do…," and so on.

HOSTED VIRTUAL SERVER

A simple way to get your website online is to use a single virtual server that runs all the software your application needs. This will include the frontend web server, the application server, the database server, and your Django code. As your site grows in popularity or requires extra performance or features, you can start splitting out frontend and backend applications to be run on separate servers.

For now, we will discuss the basic steps to getting a Django website running on a new virtual machine. Since it will be running on your local machine, it will only be accessible from your computer and to others on your networks (for example, other people in your home or office). The process for setting up Django is the same for a hosted virtual server, and so once you are ready to take an application online, you can follow the same steps and deploy to a machine that the public can access.

> **NOTE**
>
> There are a number of virtual server providers that you can use to quickly set up a virtual server. These include **AWS** (https://aws.amazon.com), **DigitalOcean** (https://www.digitalocean.com/), **Linode** (https://www.linode.com/), and others. If you already have an account with a virtual server provider like this, you can choose to create a virtual server with them. Once you have the server set up and running, then the instructions should be the same.

Once the virtual server is set up, you will connect to it using **SSH** (**Secure Shell**). SSH is built into macOS and Linux, but a third-party SSH client should be installed for Windows. We will use PuTTY. We can then start installing the software packages necessary to run our Django website. These will not only be things like Python, but also the supporting software, such as the frontend web server (NGINX) and database server (PostgreSQL).

In *Chapter 18*, *Deployment of a Django Application (Part 2 – Configuration and Code Deployment)*, after the software is installed, we can create a PostgreSQL database and user. We'll then create the web server directory to hold the static and media files.

> **NOTE**
>
> During server setup, we will be using some basic Linux system administration commands (such as **`mkdir`**, **`chmod`**, and **`sudo`**). If you haven't used these before, don't worry. We will explain them as we use them.

Next, we will need to create a system user on the virtual server that has permission to read, write, and execute the Django project. We will then execute the Gunicorn process as the newly created user.

> **NOTE**
>
> We use a non-root user for security. If there are security issues in our Django code that expose access to other parts of the system, running the server as a non-root user will limit the access of an attacker only to our application. If running it as root, the attacker could access the entire server. The root user can also start other servers, modify configuration files, and delete data for other users. It is good practice to separate applications to be run by their own users so that if one gets compromised, other applications can't be accessed.

With that in place, we can upload our Django code. This is done over **Secure File Transfer Protocol** (**SFTP**). In this book, we will use **FileZilla**, a free, multi-platform SFTP client, but you can use any SFTP client if you are already familiar with another one. Before uploading the files, we should configure the production settings in **`settings.py`** and make a minor change to **`wsgi.py`** to enable the use of the **Django configurations** that were added in *Chapter 15*, *Using Third-Party Libraries with Django*. We should then make sure our library requirements are up to date in **`requirements.txt`**.

After uploading the Django project, we can create a new virtual environment. We'll then install the requirements listed in **`requirements.txt`** using **`pip3`**. We can then run **`collectstatic`** to copy all the static files into the web server directory. Finally, we'll do our configuration setup by editing/creating configuration files for the server. We'll need to create **`systemd`** scripts to start Gunicorn and edit the NGINX configuration so that it knows how to load static files and pass on other requests to the Gunicorn instance.

This sounds like a lot of steps, but we'll go through them one by one until Bookr is up and running on our own server.

The first thing to do is to get our own virtual server up and running. In the next exercise, we will download an Ubuntu Linux ISO (a virtual installation disk) and VirtualBox, a free virtualization program that runs on many different platforms. If you already have a different preferred virtual machine host or cloud provider that you wish to use, you can skip *Exercise 17.01*, *VirtualBox Installation and Virtual Machine Creation*, *Exercise 17.02*, *Ubuntu Linux Installation*, and *Exercise 17.03*, *VirtualBox Networking Configuration*.

EXERCISE 17.01: VIRTUALBOX INSTALLATION AND VIRTUAL MACHINE CREATION

In this exercise, you will create a virtual machine that is ready for Ubuntu setup. First, you will download an Ubuntu Linux installation ISO and VirtualBox installation software. You'll set up VirtualBox and a new virtual machine, and then configure it ready to install and run Ubuntu:

1. In a web browser, go to the Ubuntu Linux server download site at https://ubuntu.com/download/server. Under the **`Ubuntu Server 18.04.4 LTS`** header, click the **`Download`** button. While version 18.04 sounds old, it is a **Long-Term Support** (**LTS**) release, which means that it is mature and stable and will be supported for a long time (until 2028):

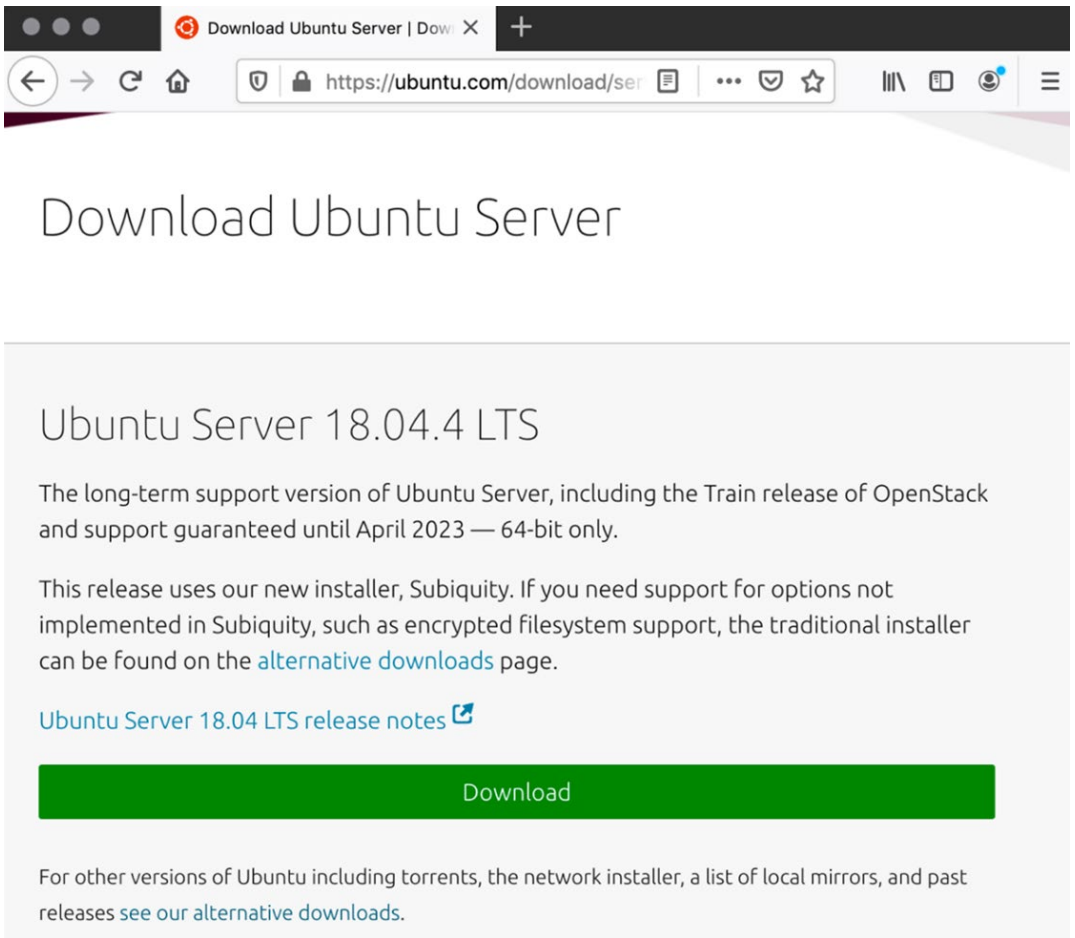

Figure 17.3: Ubuntu Server 18.04.4 download page

> **NOTE**
>
> Note that the "patch" portion of the version might change by the time you're reading this, that is, the version may have moved up to 18.04.5, 18.04.6, and so on. If the first two numbers are 18.04, then this doesn't matter.

You will be taken to the **`Thank You`** page and the ISO will begin downloading. It is around 870 MB, so it might take some time depending on the speed of your internet connection:

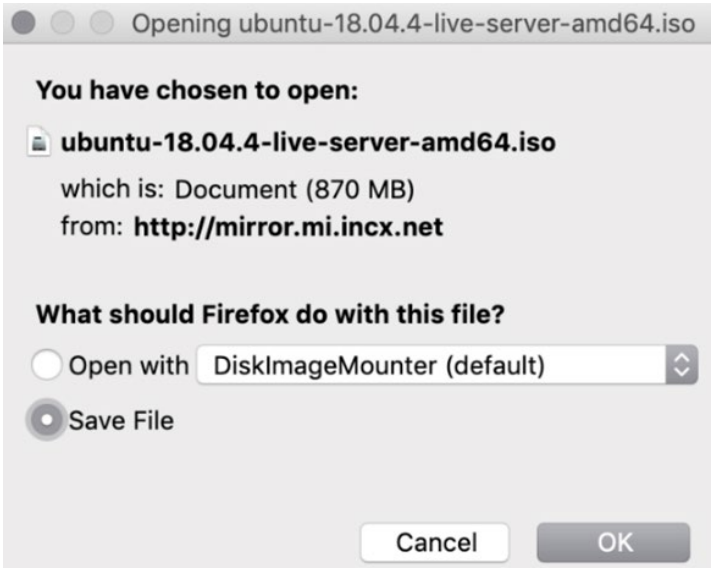

Figure 17.4: Ubuntu ISO download prompt in Firefox

2. While the Ubuntu ISO is downloading, you can also download and begin setting up VirtualBox. Go to the VirtualBox download page at https://www.virtualbox.org/wiki/Downloads. Click the **`Download`** link for your operating system under the **`VirtualBox 6.1.4 platform packages`** header. Please note again that this version might have changed by the time you read this. Click **`Windows hosts`** if you have a Windows computer, **`OS X hosts`** for macOS, and so on. If you're running Linux as your host operating system, you can click **`Linux distributions`** to find instructions for your particular Linux distribution:

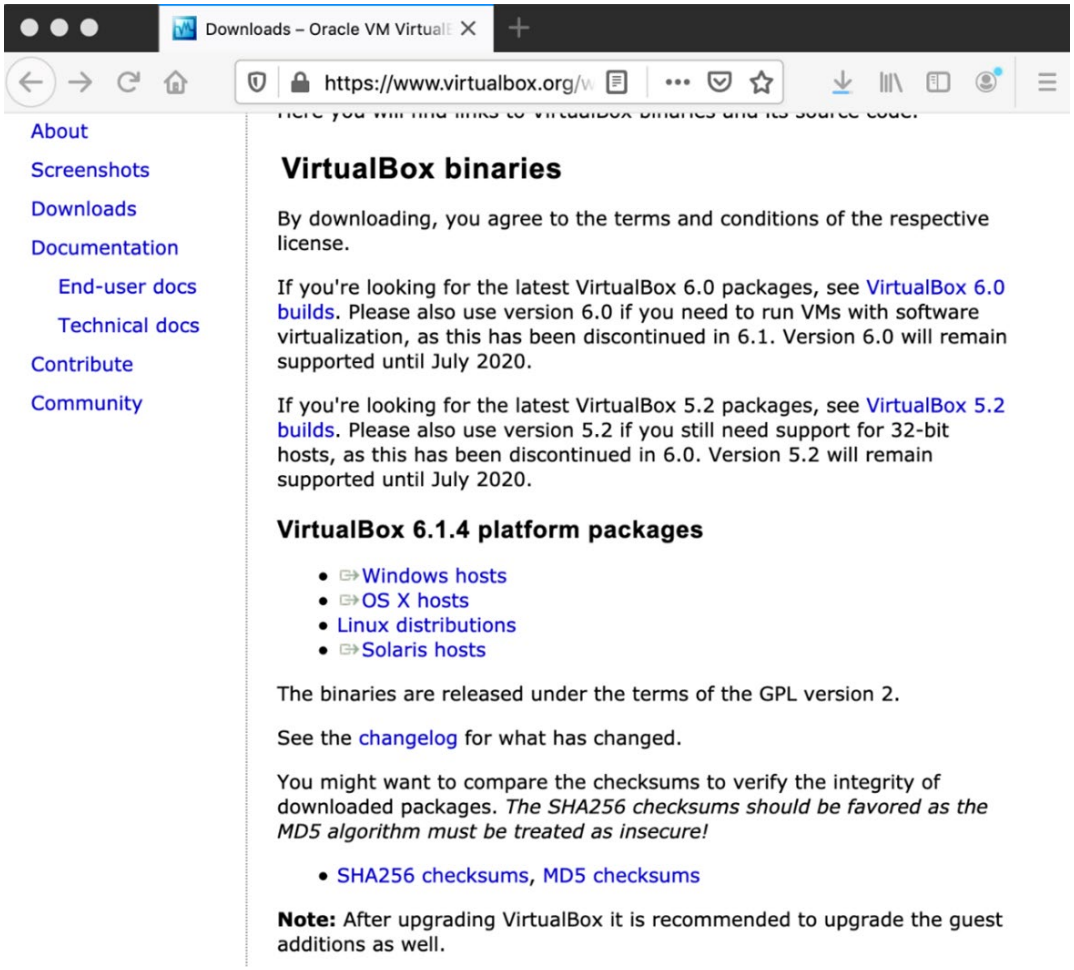

Figure 17.5: VirtualBox download page

After clicking a download link (Windows or macOS), the VirtualBox installer should start to download:

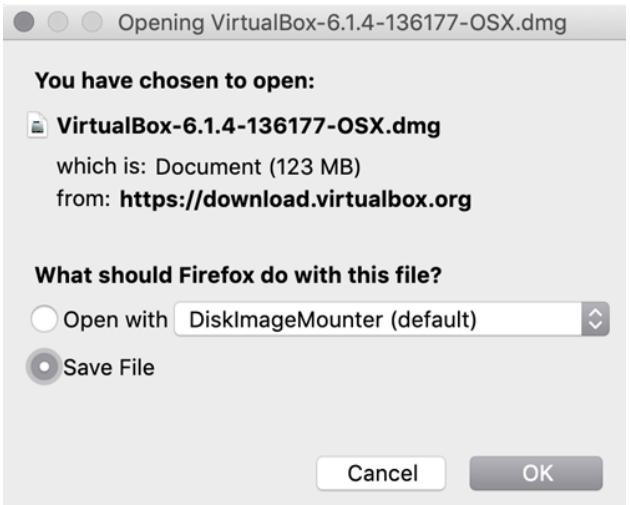

Figure 17.6: VirtualBox download prompt in Firefox

3. Once VirtualBox has downloaded, you should install it in the normal way that software is installed for your operating system.

> #### MACOS SECURITY SETTINGS
>
> In macOS, the system security settings may prevent VirtualBox from installing. You will get a message in the installer that says **`The Installation Failed`**, and macOS will prompt with an alert with the title **`System Extension Blocked`**. Click the **`Open Security Preferences`** button on the alert and quit the installer.
>
> In the System Settings app, the **`Security & Privacy`** pane should be open. Click the lock button in the bottom-left corner of the window, then enter your password. Next, click the **`Allow`** button next to the **`System software from developer "Oracle America, Inc." was blocked from loading`** message. After allowing the extension, rerun the installer and it should complete successfully.

4. Launch VirtualBox. Once again, on macOS, you may have to confirm some extra security settings (Linux/Windows users, continue to *step 5*). You may be presented with an **`Accessibility Access (Events)`** dialog box (*Figure 17.7*). This may not come up until you try to start the virtual machine, so refer back to these steps if this occurs then:

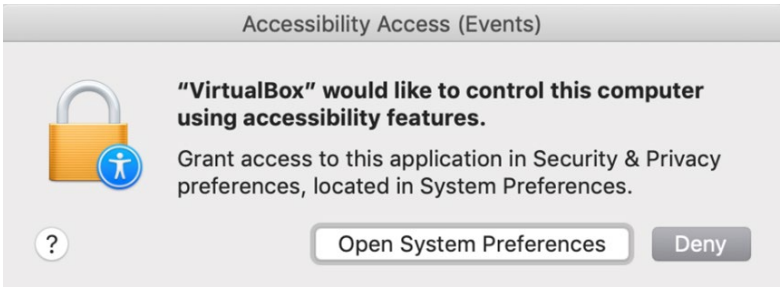

Figure 17.7: Accessibility Access (Events) dialog

Click **`Open System Preferences`**. The system preferences will open to the **`Security & Privacy`** pane. Click the lock icon in the bottom-left corner (*Figure 17.8*). You will be prompted to enter your password:

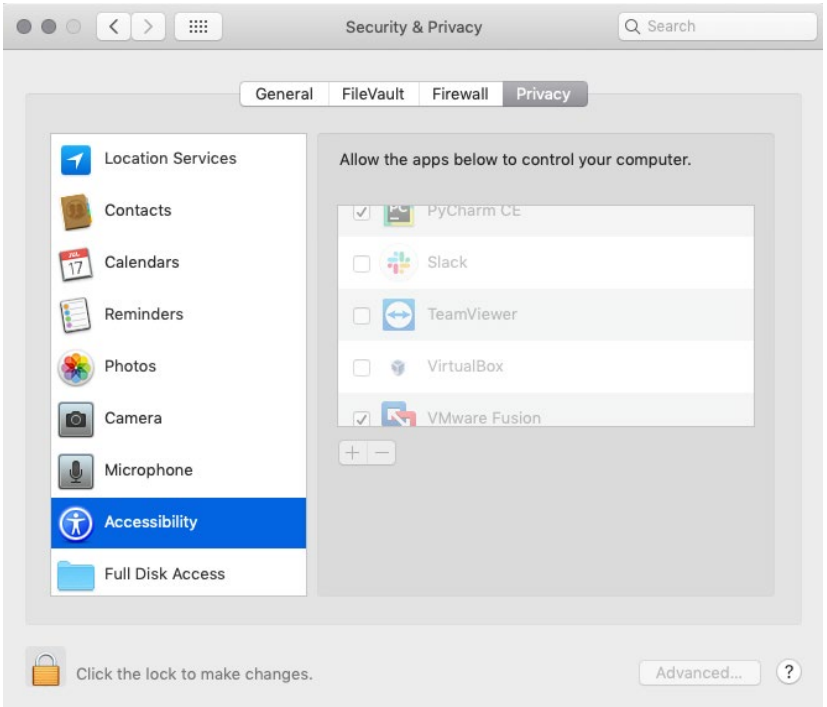

Figure 17.8: Security & Privacy pane (macOS)

Select **`Accessibility`** on the left pane if it isn't already selected. Then, check **`VirtualBox`** on the right-hand-side **`Allow the apps below to control your computer`** pane (*Figure 17.9*):

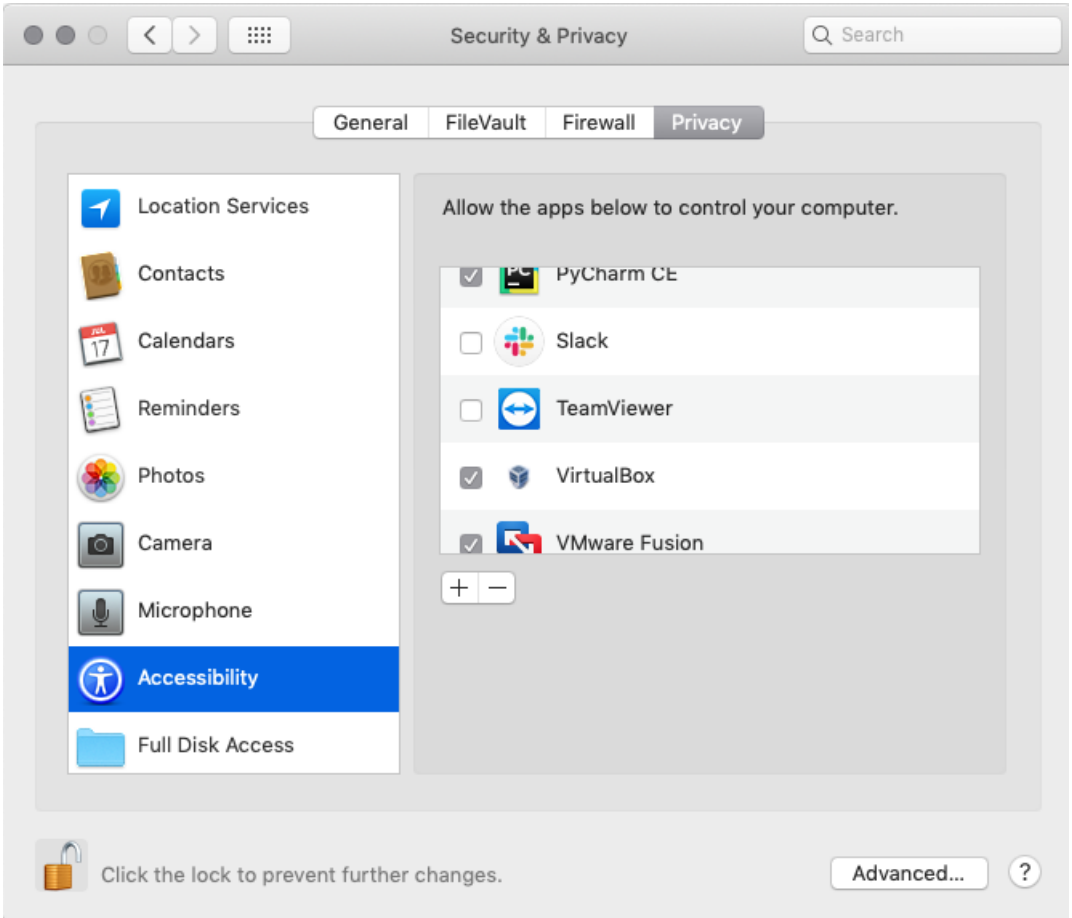

Figure 17.9: VirtualBox enabled in the Accessibility pane

You can now quit the system preferences.

5. Once VirtualBox is up and running, you'll see the **`Oracle VM VirtualBox Manager`** window:

Figure 17.10: Oracle VM VirtualBox Manager

6. Click the **`New`** button at the top of the window to create a new virtual machine.

   You will be stepped through the setup wizard. The first step is the **`Name and operating system`** settings. You will give the virtual machine a name, select where to save it on your computer, and tell VirtualBox what operating system you are installing:

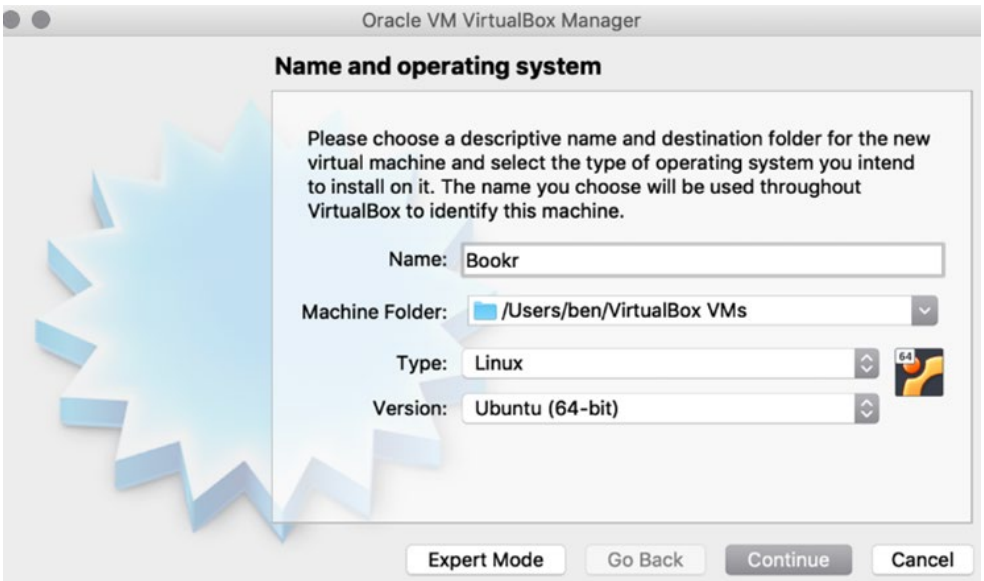

Figure 17.11: VirtualBox setup wizard step 1

The name is used to identify the machine as yours, but it's not very important what you pick. Just enter something such as **`Bookr`**.

`Machine Folder` is where VirtualBox stores the virtual machine's data. The default location is usually fine. The virtual machine's virtual hard drive can take up a few gigabytes of space, so you might choose to store it on another disk if you require more space and have one available.

Select **`Linux`** for **`Type`** and **`Ubuntu (64-bit)`** for **`Version`**, then click **`Continue`**.

7. The next part of the wizard is **`Memory size`**, where you set the amount of memory to allocate to your virtual machine:

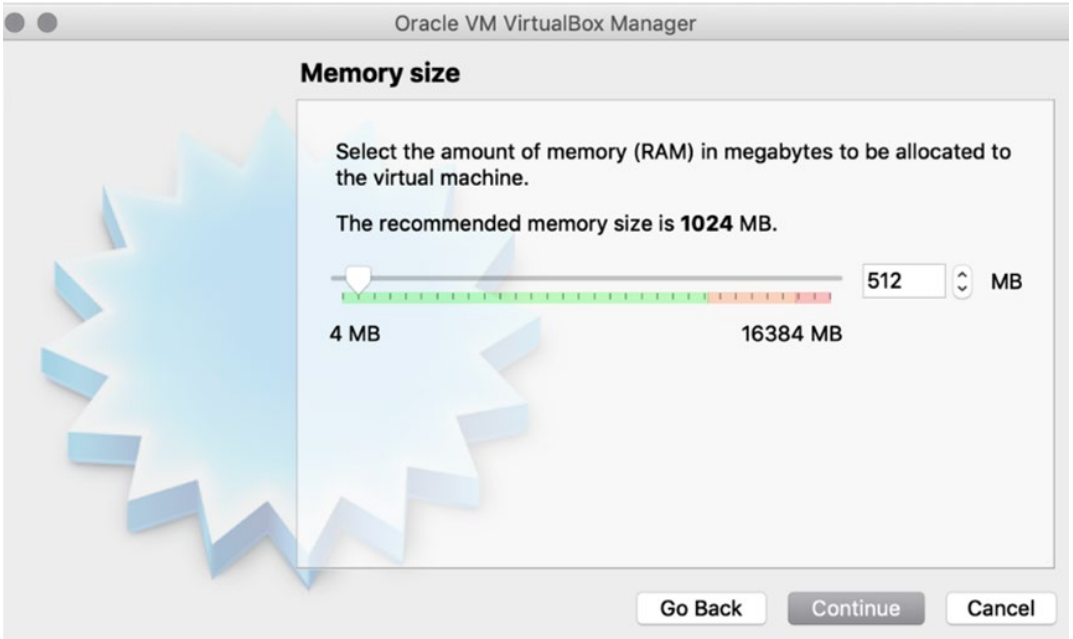

Figure 17.12: Memory size

The more you allocate to it, the less is available to your host computer and it may cause other applications you're running to slow down. However, if you don't allocate enough, your virtual machine might slow down or the applications it's running may crash.

The recommended minimum for Ubuntu is 1,024 MB (1 GB). You can leave this at the default, at least during installation. After installation, this can be decreased if it is affecting the host system's performance.

> **NOTE**
>
> Choosing the amount of memory for a server can be complicated and depends on how many users you have and how complicated your site and database are. With a real hosted virtual server, you will need to perform testing and scale up or down based on load and cost requirements.

Click **`Continue`**.

Next, set up the virtual hard disk in the **`Hard disk`** settings. The default **`10.00 GB`** option is plenty. Just make sure **`Create a virtual hard disk now`** is

Figure 17.13: Hard disk settings

Then, click **`Create`**.

8. You should then specify the hard disk file type:

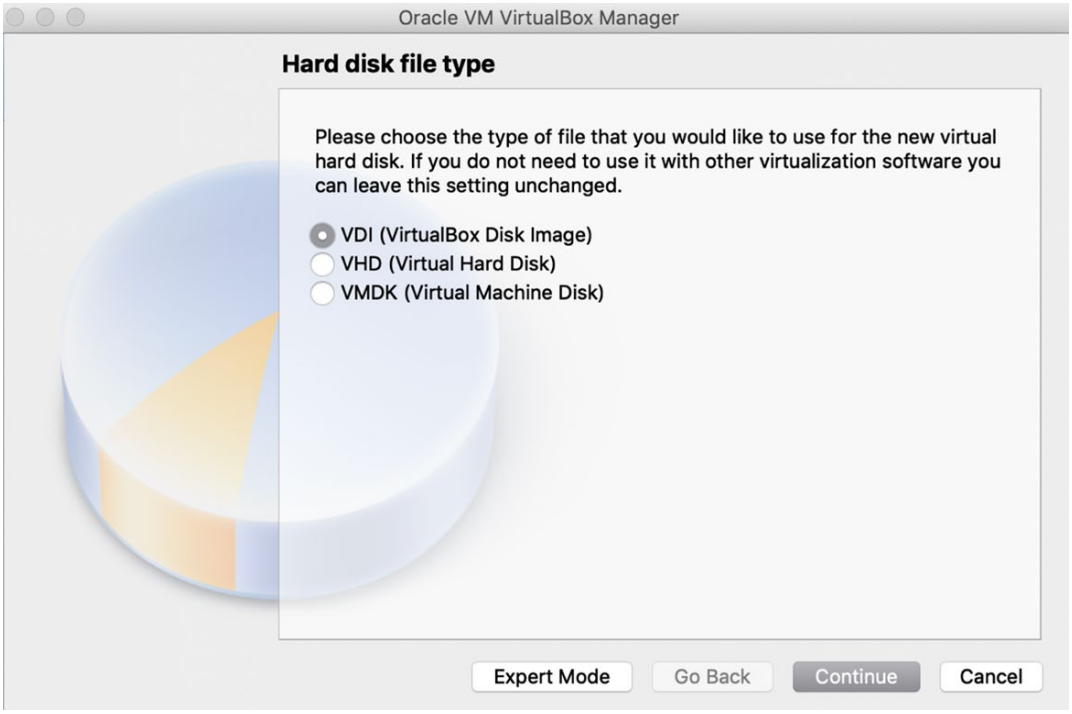

Figure 17.14: Hard disk file type

Leave the default **`VDI (VirtualBox Disk Image)`** setting selected (see *Figure 17.14*) and click **`Continue`**.

9. In the next step, **`Storage on physical hard disk`**, you will need to select how the virtual hard drive is allocated. You can choose **`Dynamically allocated`**, which means that the virtual hard disk file on your computer will only take up as much space as you have stored inside it. For example, if you store 1 GB of data on your virtual machine (counting all your data and the operating system), then the file on your host computer will be 1 GB too (actually a little bit bigger, due to some overhead, because VirtualBox needs to store a bit of data about the hard disk). The more data you store, the bigger the file will get, up to a maximum of 10 GB (plus a little overhead).

   If you choose **`Fixed size`**, a 10 GB file will be created on your host computer straightaway but won't get any bigger as you store more data in the virtual machine.

As the message says, **`Fixed size`** is a little faster but will take longer to set up initially. For our purposes, we won't notice a difference, so leave it on **`Dynamically allocated`**:

Figure 17.15: Storage on physical hard disk

Then, click **`Continue`**.

10. The next step is for the virtual disk's file location and size. This is where the virtual disk is stored, and by default, it's inside the directory selected in *step 6*. This default location is fine (*Figure 17.16*):

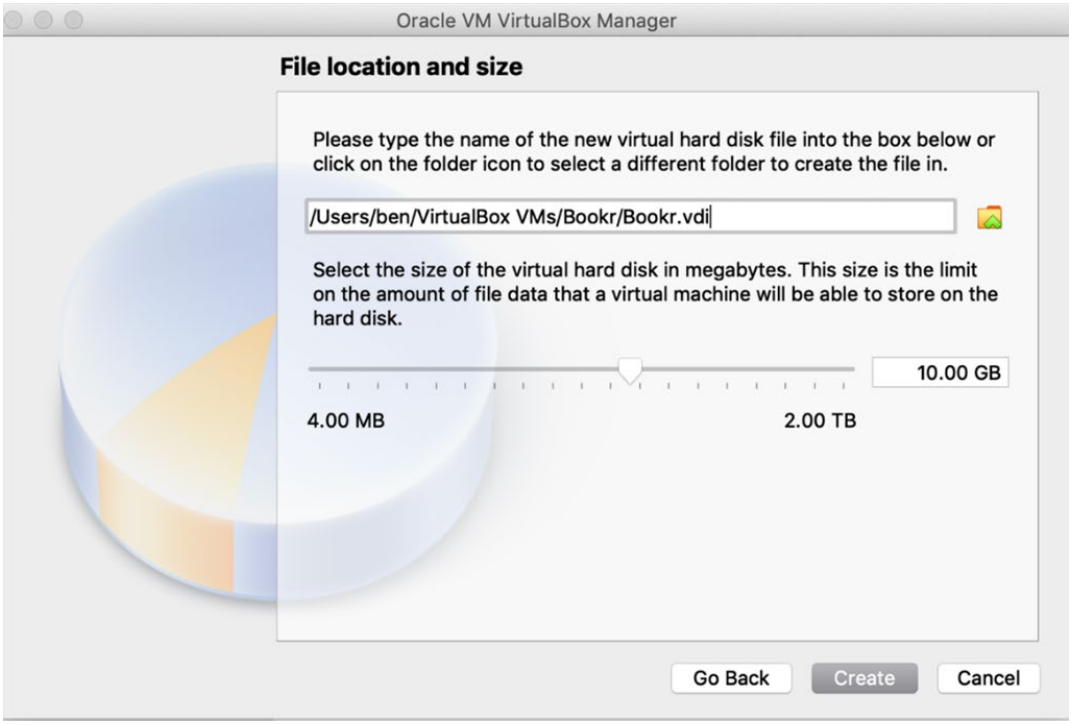

Figure 17.16: File location and size

Leave the size at **`10.00 GB`** – since it's *dynamically allocated* in *step 10*, we won't use this much storage on the host disk unless we fill up the VM. Click **`Create`**.

11. The **`Oracle VM VirtualBox Manager`** window should now show all the information about the virtual machine we just created:

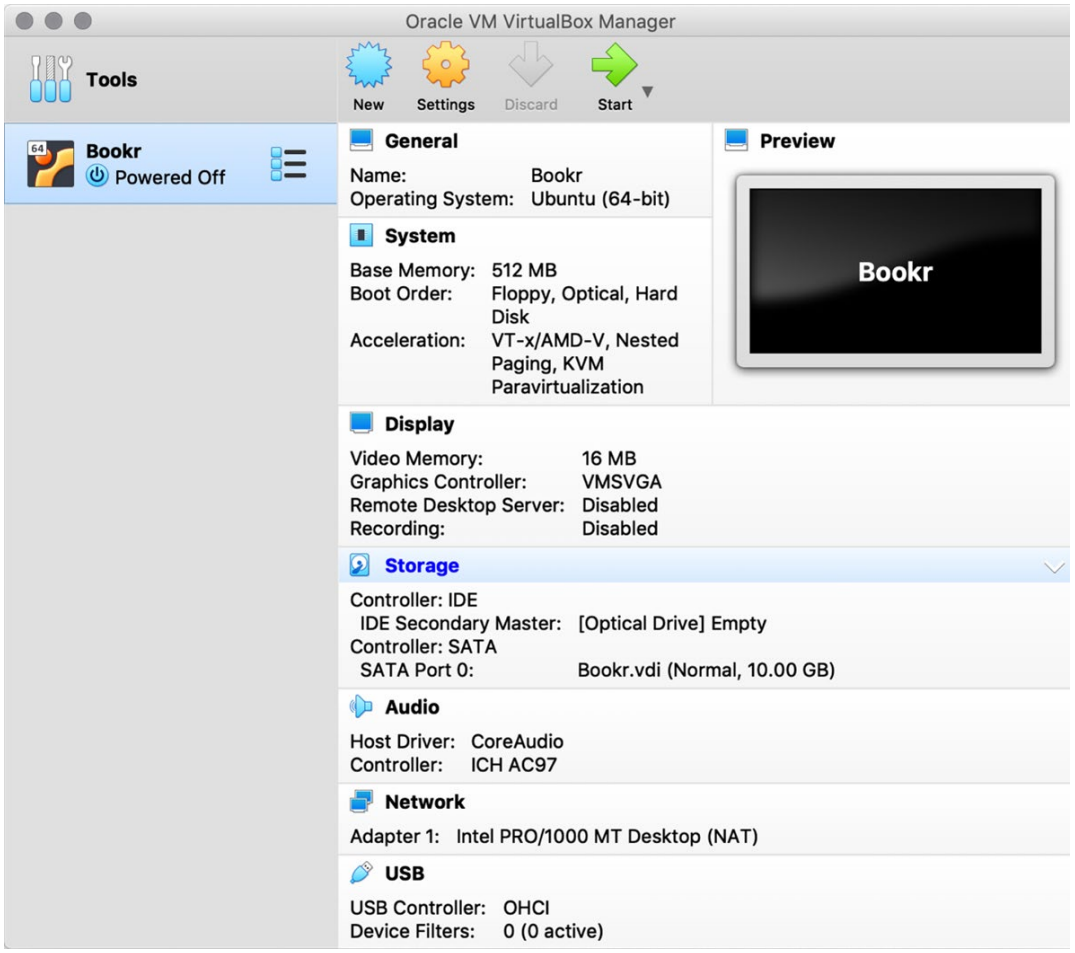

Figure 17.17: Bookr virtual machine summary

You now need to attach the Ubuntu ISO that you downloaded to the virtual machine. This is like inserting a CD into a CD drive or attaching a USB flash disk to a computer.

Click on the **`Storage`** header text in the virtual machine detail view to bring up the storage settings for the virtual machine:

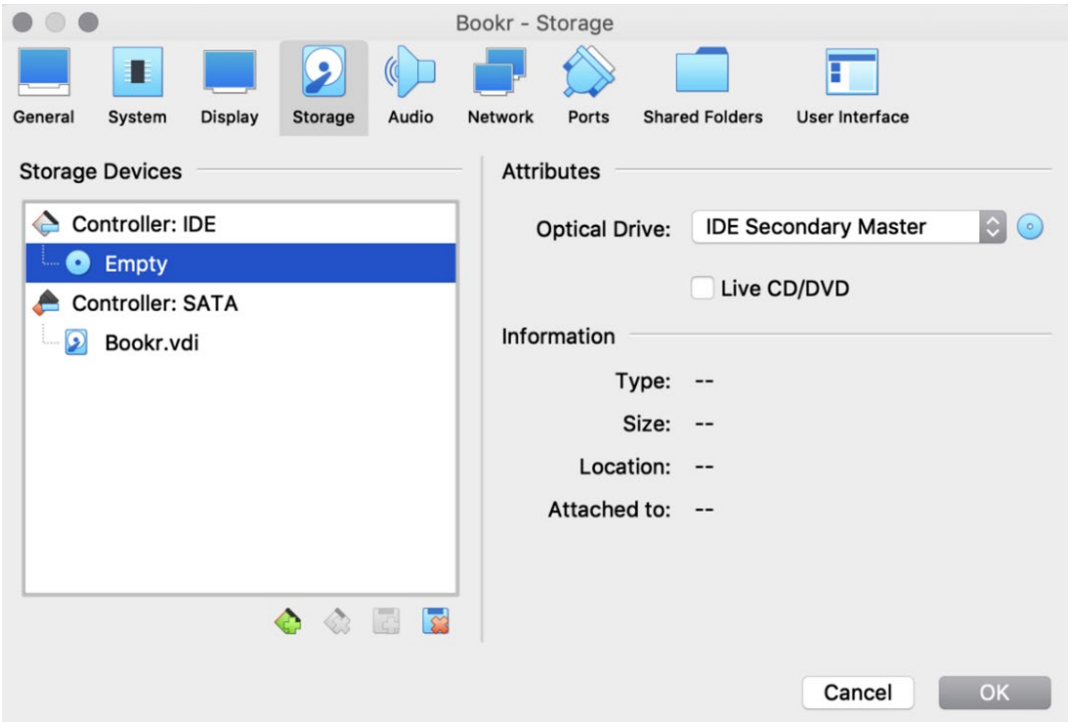

Figure 17.18: Storage settings

In the **`Storage`** window, select the CD/DVD drive under the **`Controller: IDE`** section. It currently says **`Empty`**.

Next, click the CD icon under the **`Attributes`** section to the right of the window. This will bring up a context menu where you can select what type of disk to attach:

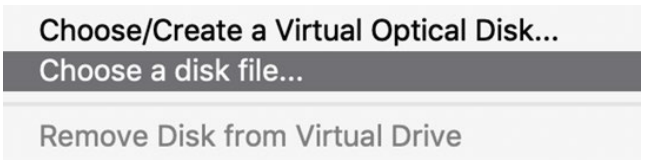

Figure 17.19: Attributes context menu

Select **`Choose a disk file`**....

This will bring up a file selector dialog. You should navigate to the downloaded Ubuntu Linux ISO, select it, then click **`Open`**:

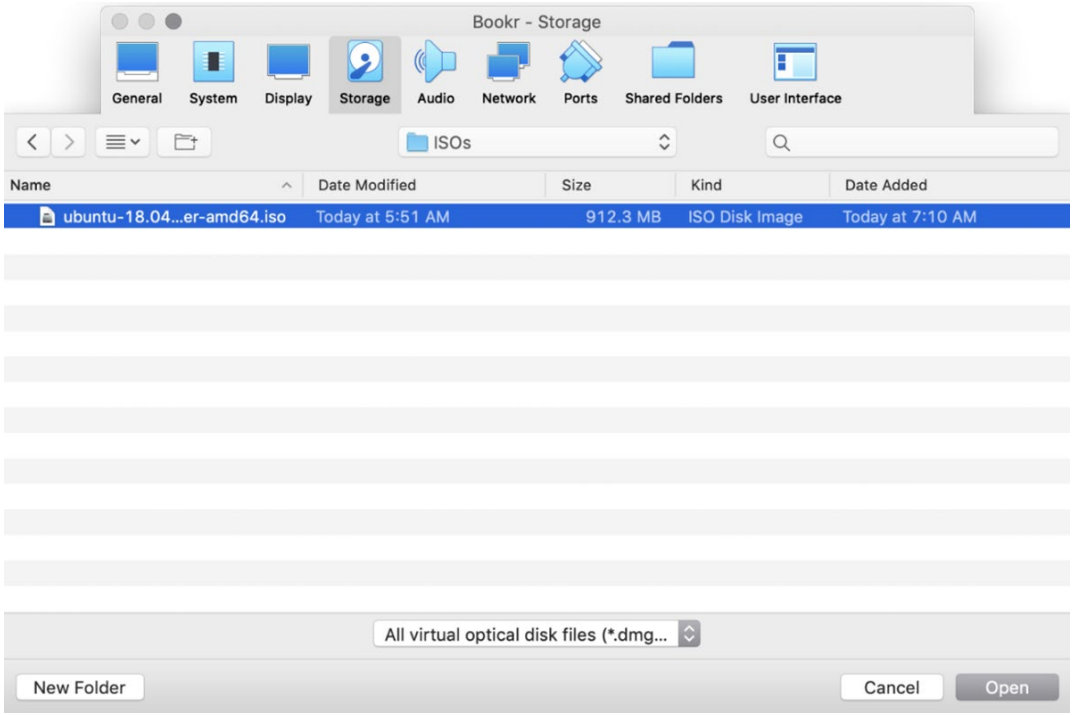

Figure 17.20: Select the Ubuntu ISO that was downloaded

The **`Storage`** window should now display the selected ISO in the **`Location`** information:

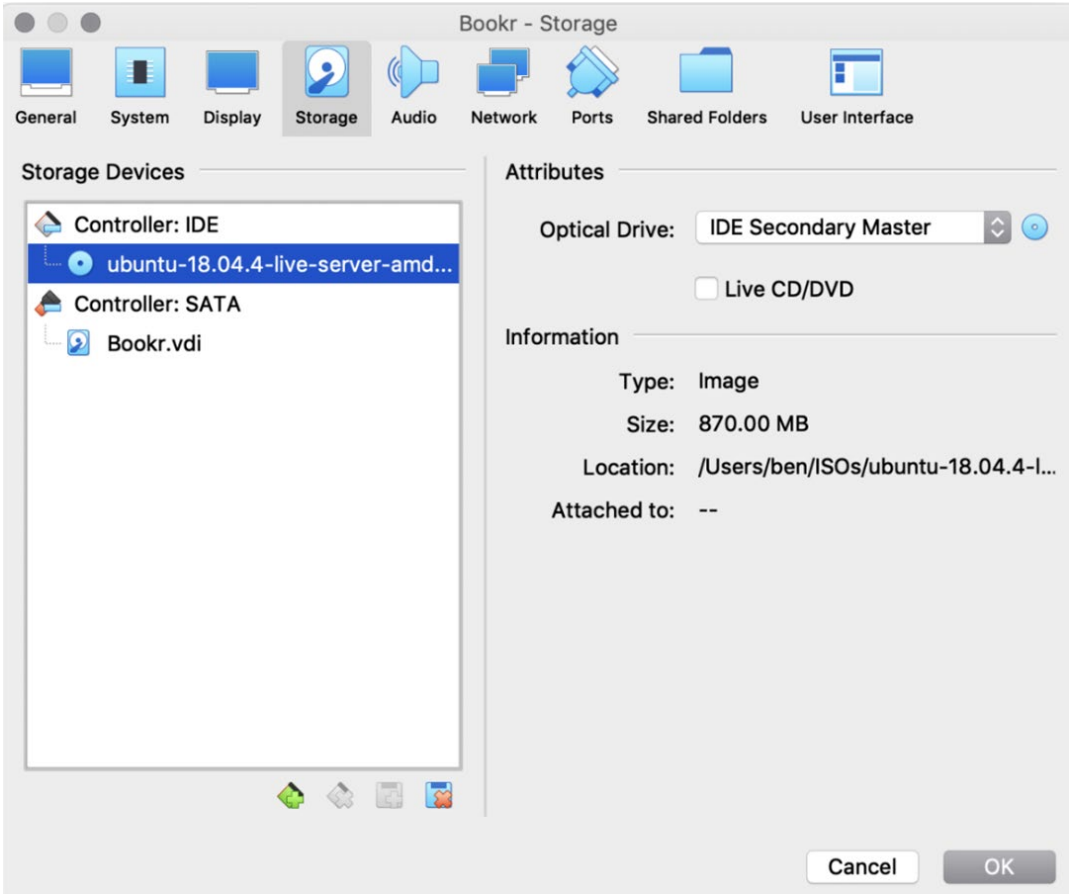

Figure 17.21: Location of the Ubuntu ISO displayed in the Storage window

Click **`OK`** to close the settings window.

> **NOTE**
>
> If you're running VirtualBox on macOS with a Retina display, the virtual machine display will be very small. Open the display preferences for the virtual machine by clicking the **`Display`** section header. Then, set the scale factor to **`200%`** and click **`OK`** (refer to *Figure 17.22*).

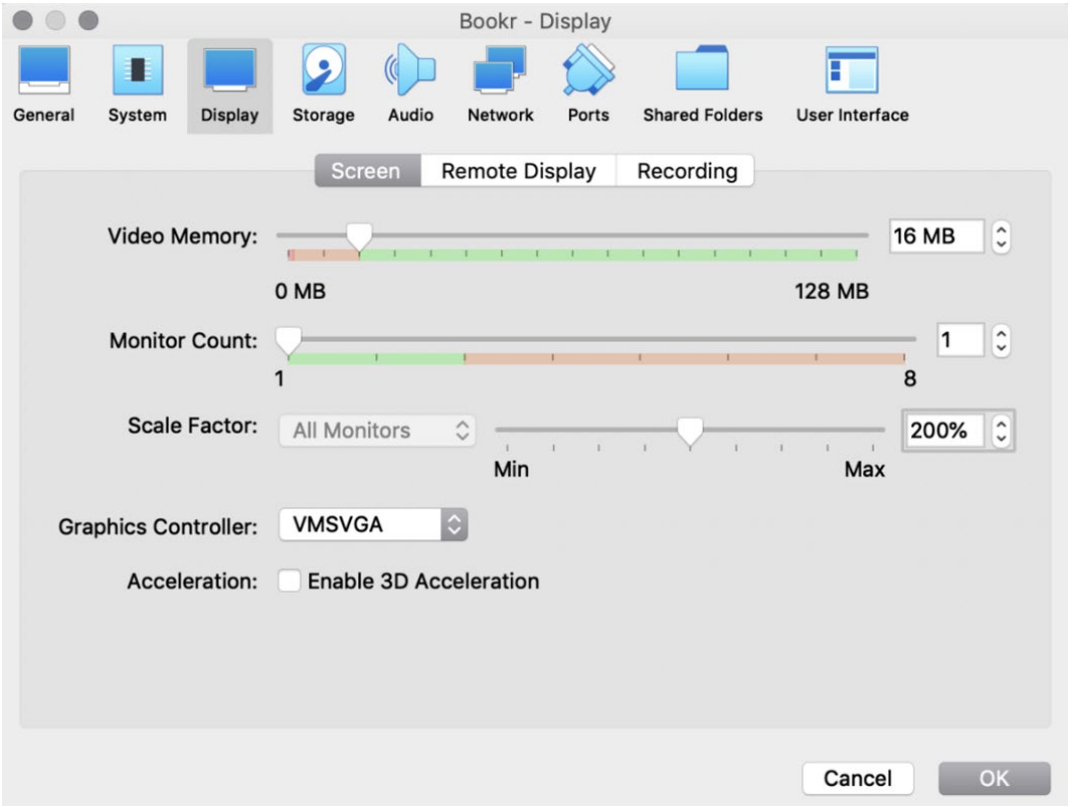

Figure 17.22: Display scale settings

We're now ready to start the virtual machine, which will begin the Linux installation process. But we will continue that in the next exercise.

In this exercise, you downloaded VirtualBox and an Ubuntu Linux ISO in preparation to set up your virtual server. You created a new virtual machine and configured it with enough RAM and hard drive space to run a basic Ubuntu server. It's now ready to boot up and begin the installation.

In the next exercise, you will start up the virtual machine and begin the Ubuntu Linux installation. You can follow these instructions if you have a virtual machine ready to go in VirtualBox or some other virtualization software (such as VMware or Parallels). Or, if you already have a virtual server (or a real Linux server) set up with a cloud provider, you can skip this exercise and use that instead, and move on to *Exercise 17.04*, *Package Update and Installation*.

EXERCISE 17.02: UBUNTU LINUX INSTALLATION

This exercise continues on from *Exercise 17.01*, *VirtualBox Installation and Virtual Machine Creation*. In the previous exercise, you created a virtual machine (the virtual hardware, so to speak), and in this exercise, you will install the operating system (Ubuntu Linux) on it. You should mostly be able to follow the installation instructions even if you use different virtualization software. If you have an account with a cloud provider that lets you start up a prebuilt Ubuntu Linux server, you can use that instead, and skip this exercise:

> **NOTE**
>
> Throughout this exercise, you will be prompted to press *Enter*. On macOS, you can press *Return* instead.

1. Launch VirtualBox if it's not already running. You should see the **`Oracle VM VirtualBox Manager`** window.

2. Click the green **`Start`** button at the top of the **`Oracle VM VirtualBox Manager`** window. It will boot from the ISO you attached in *Exercise 17.01*, *VirtualBox Installation and Virtual Machine Creation*. A new window will open showing what the virtual machine's screen is displaying. You will be prompted to select an ISO (*Figure 17.23*) but since we selected one in *Exercise 17.01*, *VirtualBox Installation and Virtual Machine Creation*, it will already be selected, so just click **`Start`**:

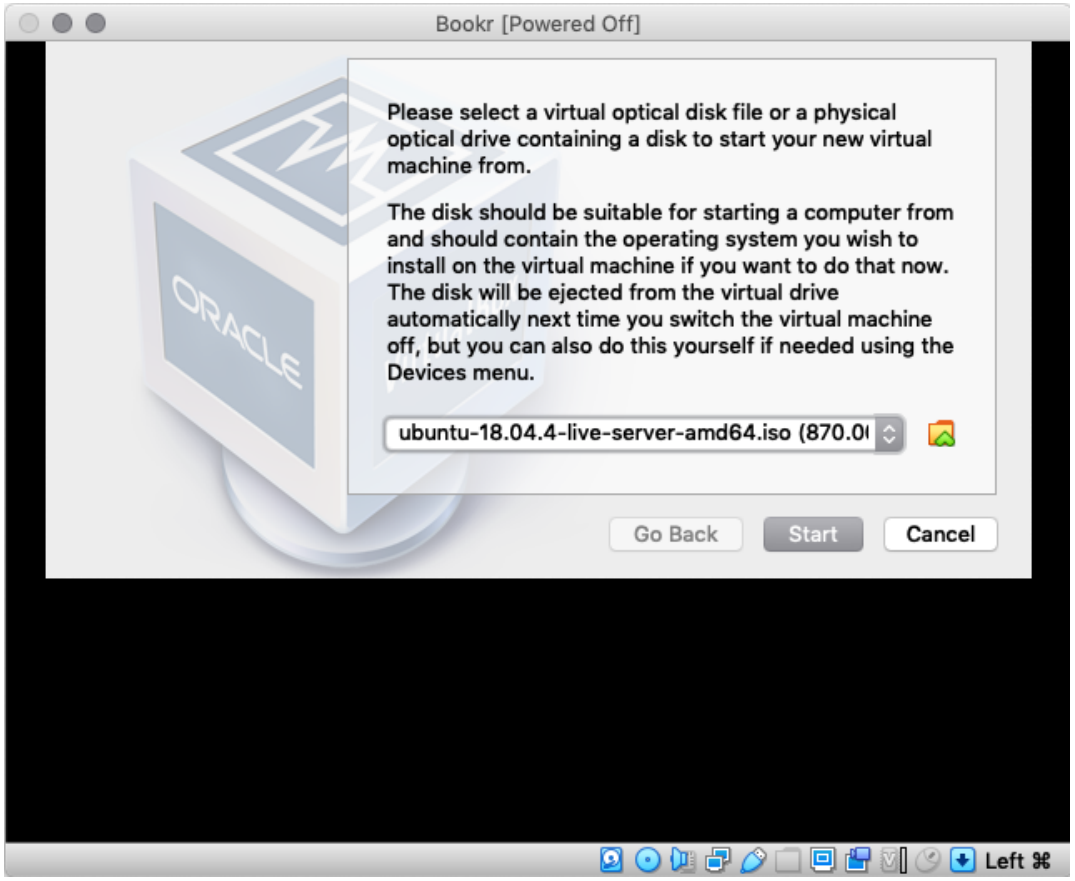

Figure 17.23: ISO file selection

3. Ubuntu will start booting up on the virtual screen; you will see some log messages and within a minute or two, you will get the language selection screen:

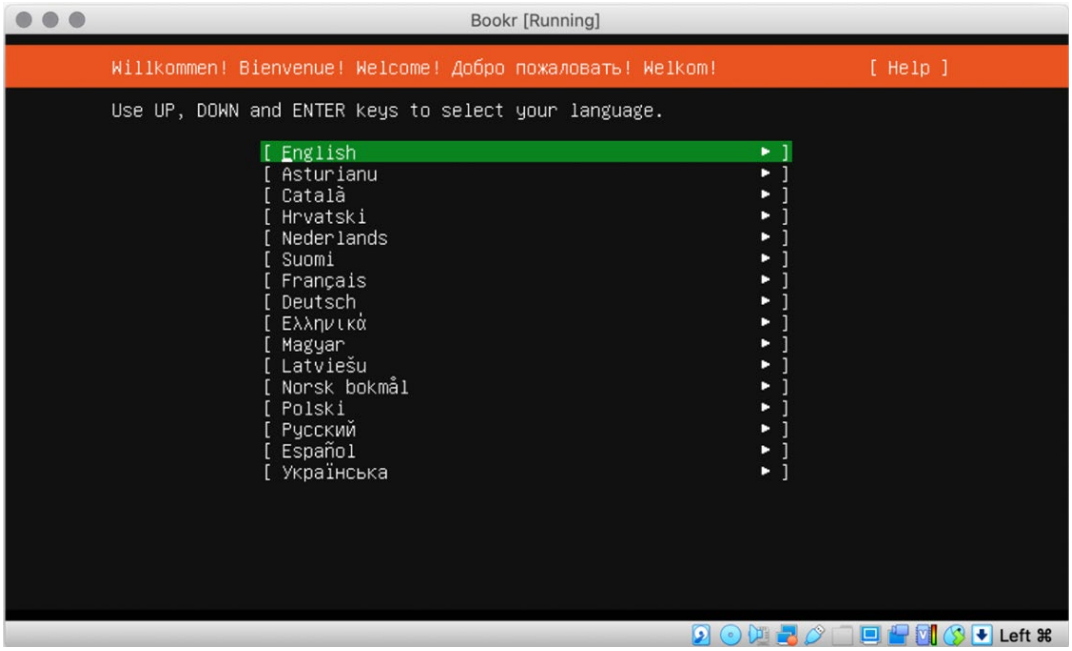

Figure 17.24: Ubuntu language selection screen

Click into the window and VirtualBox will capture your mouse and keyboard input. You might get a message as in *Figure 17.25*:

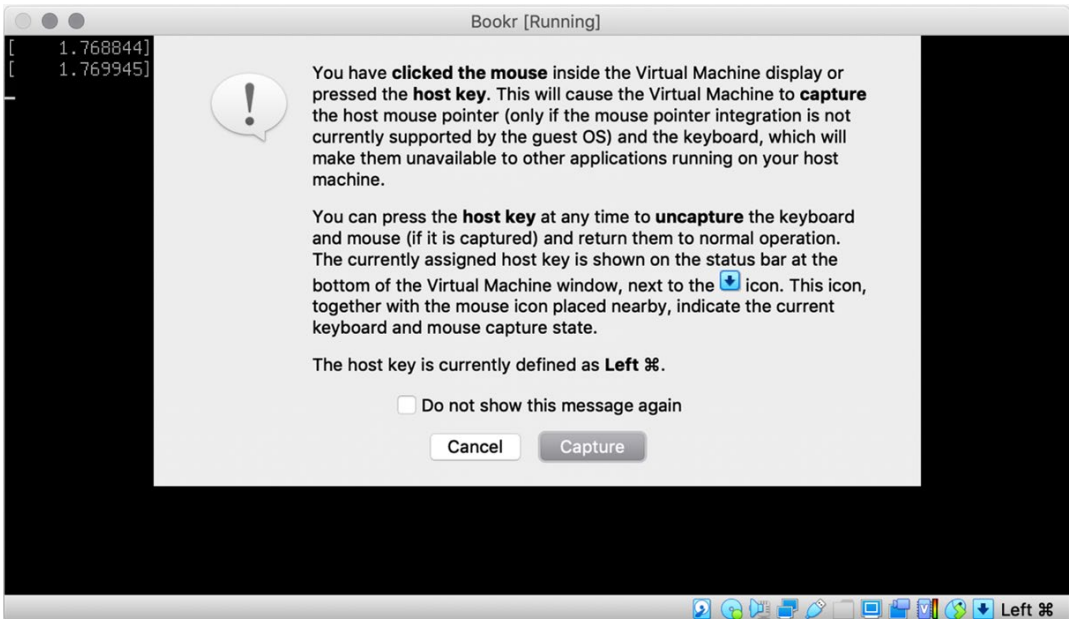

Figure 17.25: Message on the screen after clicking the window

You will have to press the *Command* key (macOS) or the *Windows* key (Windows) to release the capture.

4. Use the up and down cursor keys to select your preferred language, then press *Enter* to continue.

5. In the next step, you will be asked whether you want to update the installer before continuing:

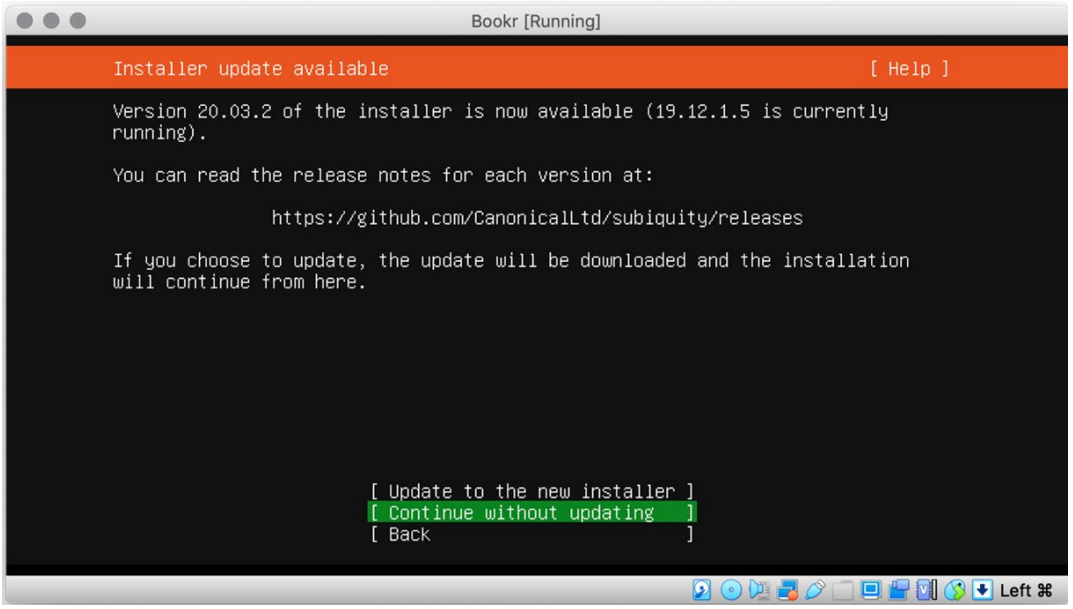

Figure 17.26: Installer update selection

We don't need to do that, so use the cursor keys to select **`Continue without updating`**, then press *Enter*.

6. The installer automatically selects the keyboard layout it thinks you are using:

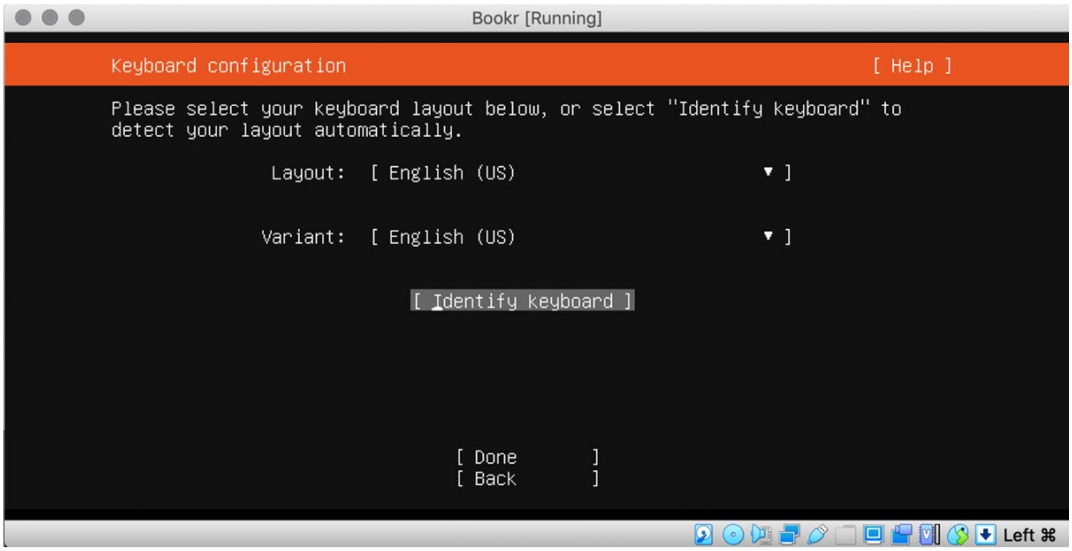

Figure 17.27: Keyboard identification screen

If it doesn't seem to be correct, you can choose **`Identify keyboard`**, then press *Enter*, then follow the steps to select a keyboard layout (this involves answering a few questions about your keyboard). When you're happy with the keyboard selection, select **`Done`** and press *Enter*.

7. You'll need to select your network interface in the next step:

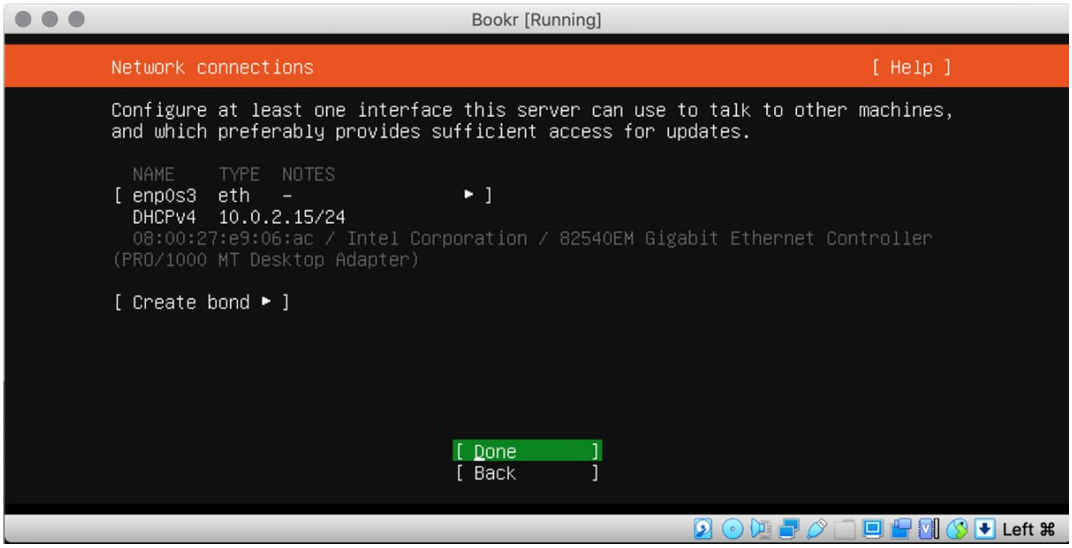

Figure 17.28: Network connection setup

Since VirtualBox provides standard virtual hardware with only one network interface, just select **Done** and press *Enter*.

8. On the next screen (*Figure 17.29*), you can enter a proxy server:

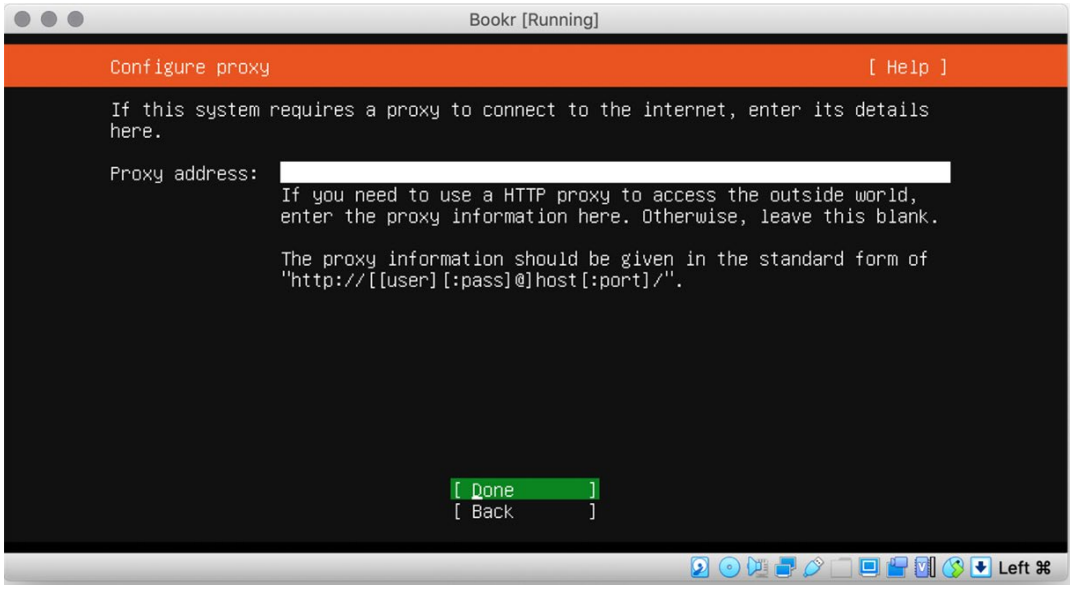

Figure 17.29: Proxy server configuration

You should type in a proxy server if you need to use one to access the internet. If not, just select **Done** and press *Enter*.

9. Next, the Ubuntu installer will automatically pick the mirror server to download packages from; this is based on your location:

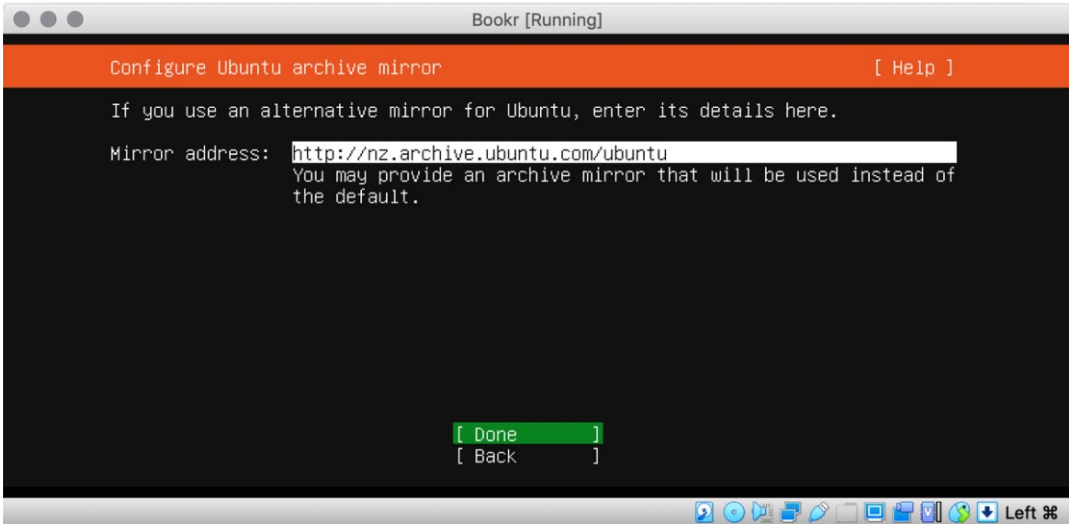

Figure 17.30: Mirror server selection (auto selected)

You can leave this as the default (it is selected automatically based on your region), then select **Done** and press *Enter*.

10. The next step is to partition the virtual hard drive and format it. The installer guides us through this process, and since we're using the entire virtual disk, it's very simple:

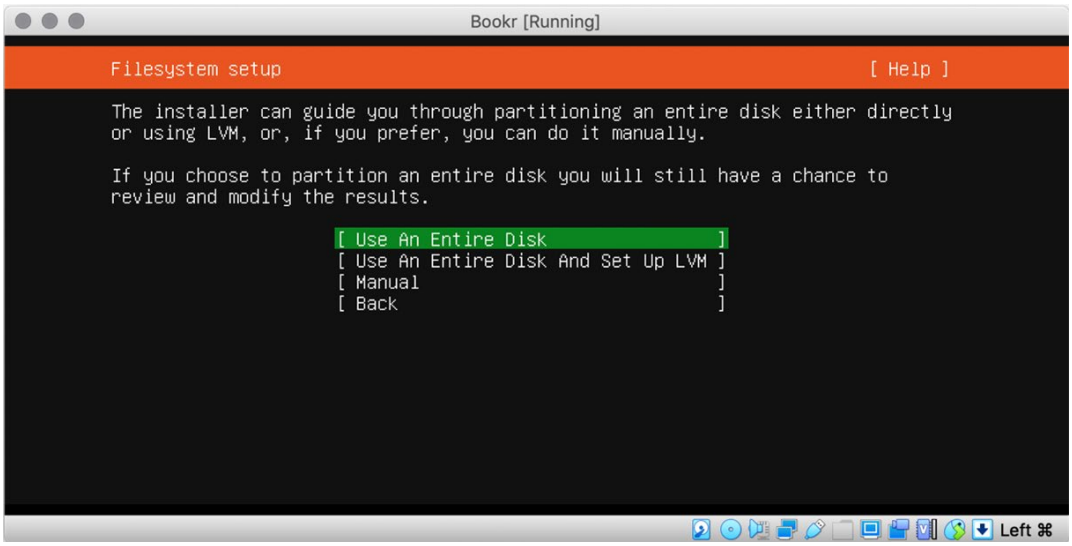

Figure 17.31: Filesystem setup

Select **Use An Entire Disk**, then press *Enter*.

11. Then, select the hard disk to use. Since there's only one, select it from the list (it will start with **VBOX_**; see *Figure 17.32*), and then press *Enter*:

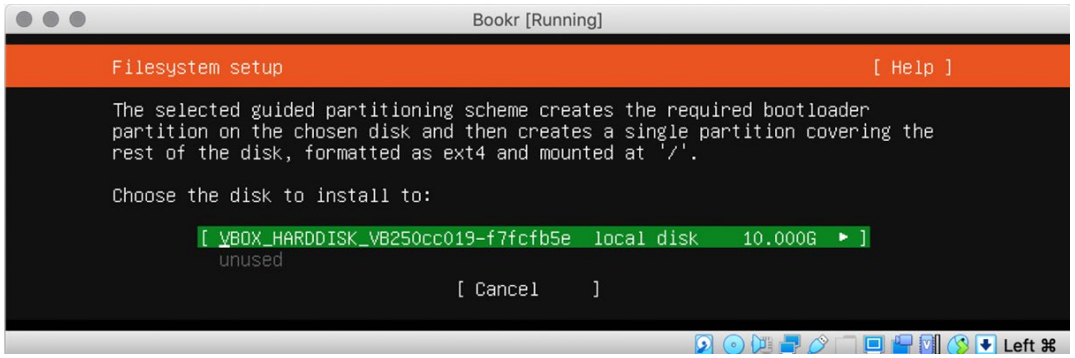

Figure 17.32: Hard disk selection

It should be noted here that this applies to your virtual hard disk. No changes will be made to the formatting of the host computer's disk. Even if something goes wrong during the installation, you can just delete the virtual disk and start again.

12. On the next screen (*Figure 17.33*), the installer automatically generates a partition scheme. Just select **Done** and press *Enter*:

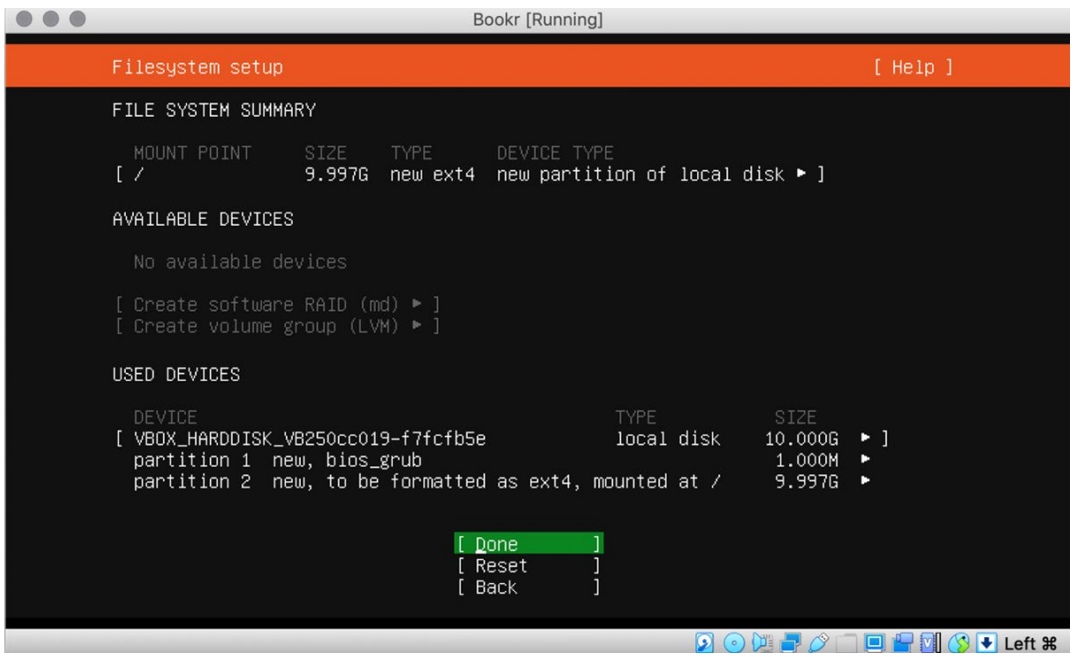

Figure 17.33: Partition entry

13. You will be asked to confirm that you want to format the disk. Use the arrow keys to select **`Continue`**, then press *Enter*:

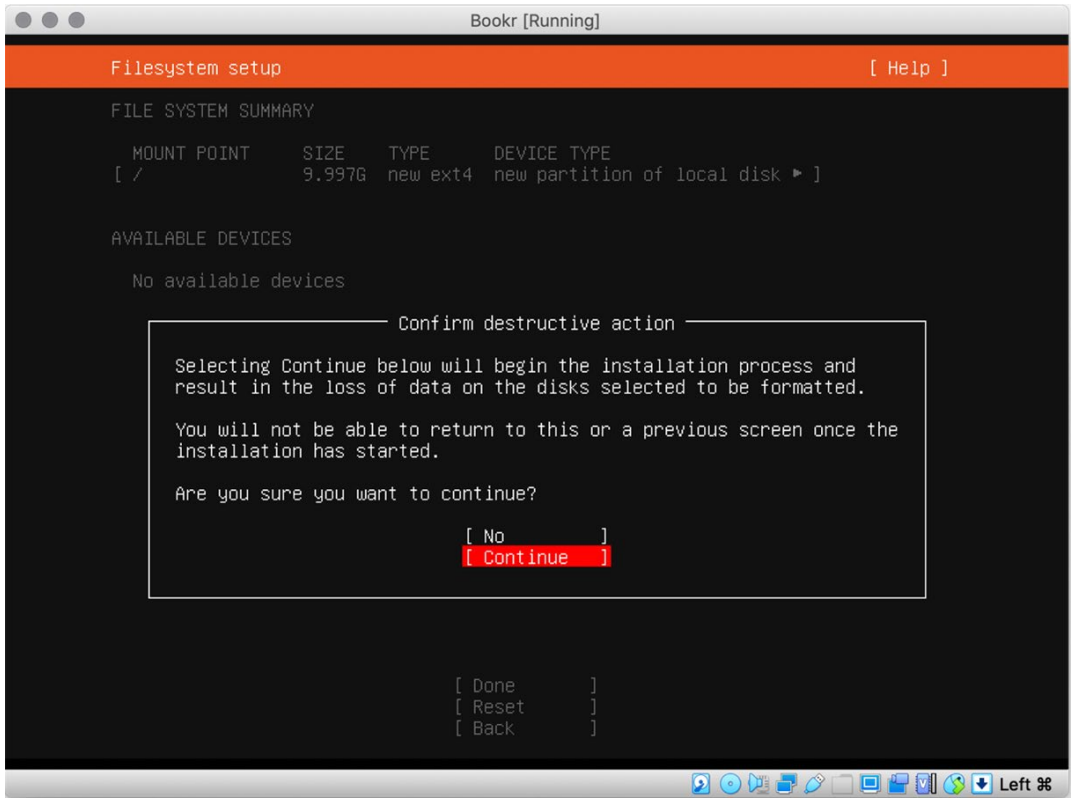

Figure 17.34: Confirm disk formatting

14. After the formatting is complete, on the next screen, you will need to enter your details. After typing in each field, use *Tab* to move to the next field. See *Figure 17.35* for an example:

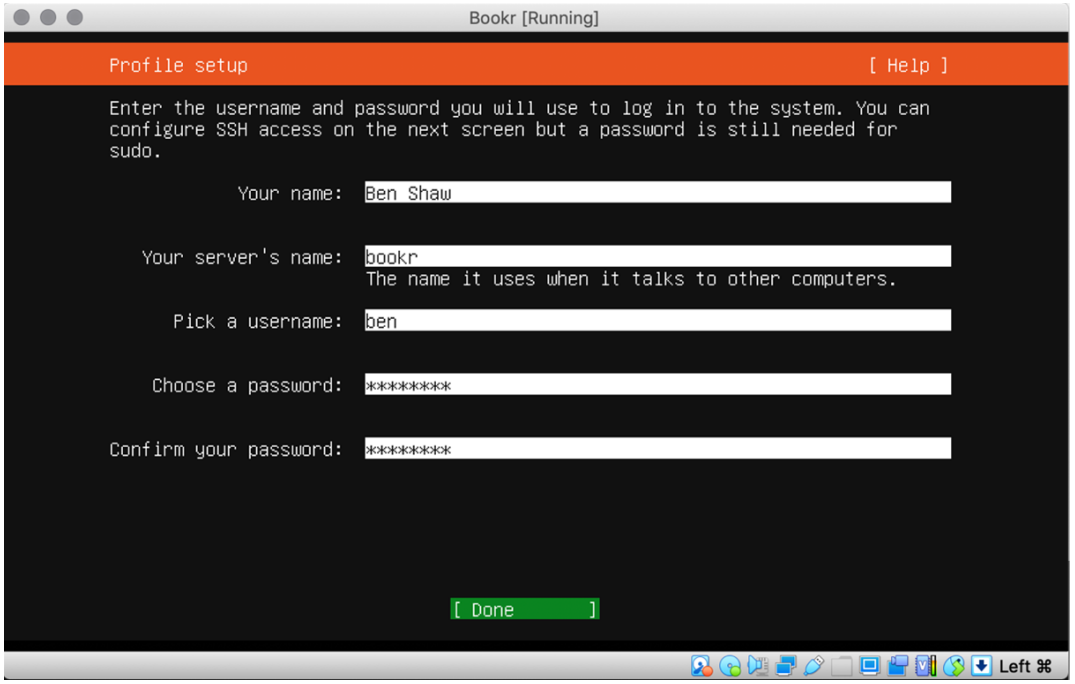

Figure 17.35: Profile setup

Type in the following:

- Your name.
- For **`Your server's name`**, enter **`bookr`**.
- For your username, you can choose your first name or any short username you like. This is the name of your personal user – we will create a specific Bookr user later, after the server is set up.

- Enter a password for your user account, and the same in the **`Confirm your password`** field.

  Press *Tab* again to select **`Done`**, then press *Enter*.

15. On the next screen, we want to enable SSH access on the server so that we can connect to it using SSH and SFTP:

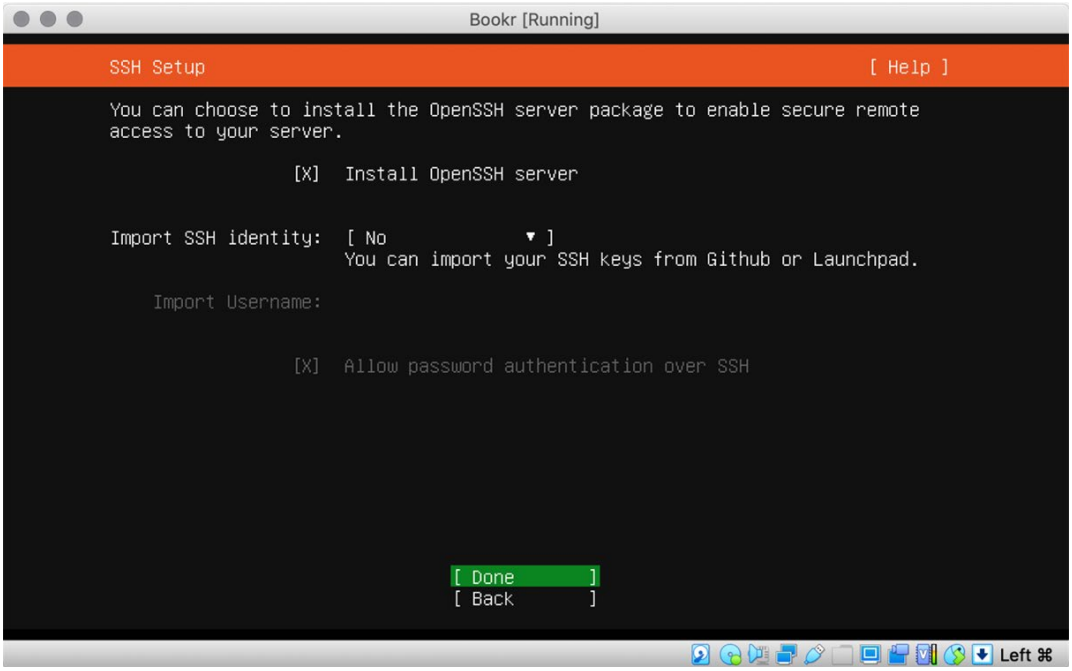

Figure 17.36: SSH setup

Press the spacebar to check the **`Install OpenSSH server`** option. Then, move the cursor down, select **`Done`**, and press *Enter*.

16. The next screen allows you to select packages to pre-install:

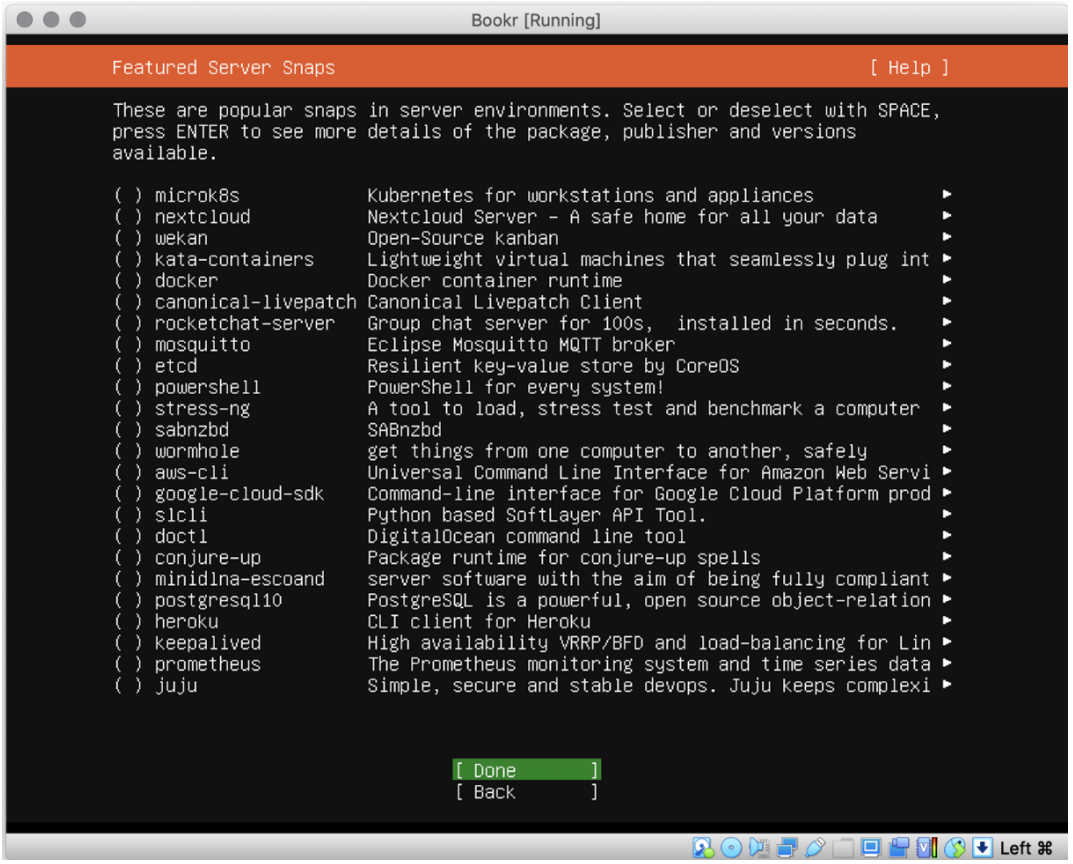

Figure 17.37: Package selection

You might notice **`postgresql10`** in this list. While we could install PostgreSQL at this step, for consistency, we will install it later with all the other packages we need. Since we don't want to install anything now, move down to select **`Done`** and press *Enter*.

17. The next screen will show a log of the activity, and the installer will perform the base installation, then automatically install the latest security updates:

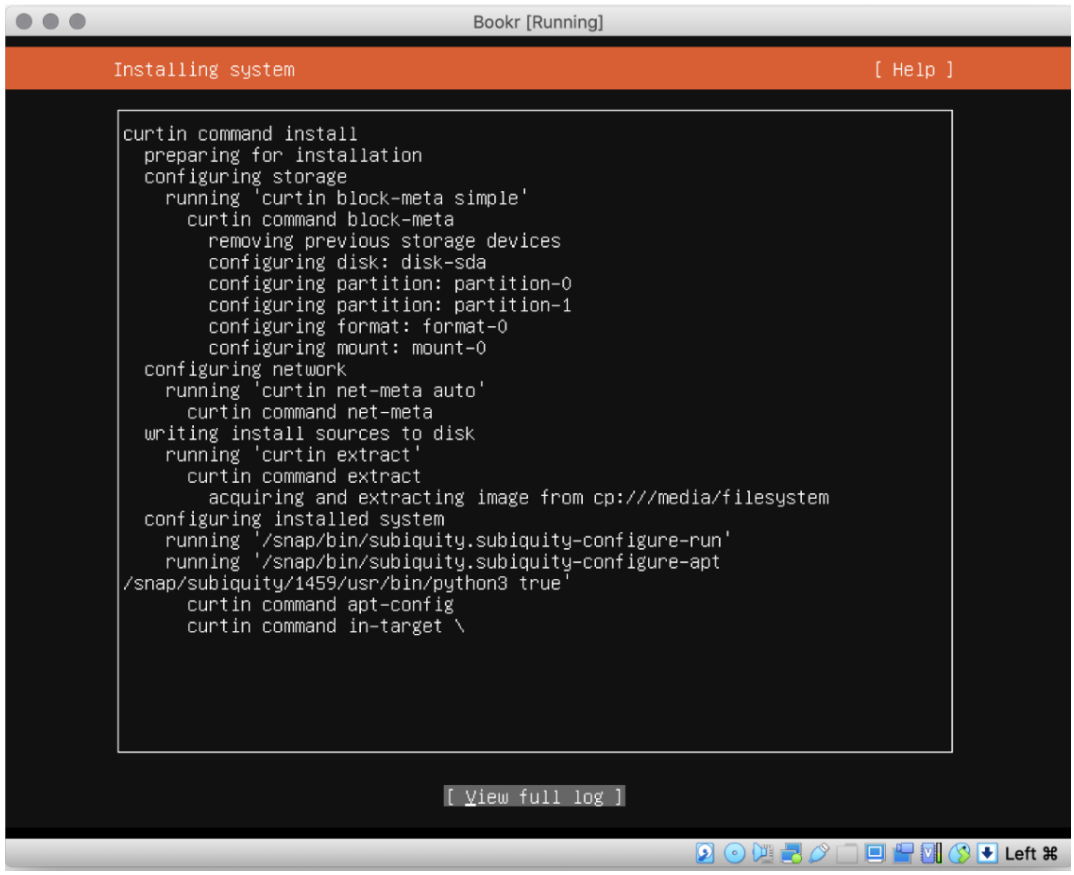

Figure 17.38: Installation screen

This can take some time, so you will need to just wait for it to finish.

18. When it's finished, the options at the bottom will change to **`View full log`** and **`Reboot`**:

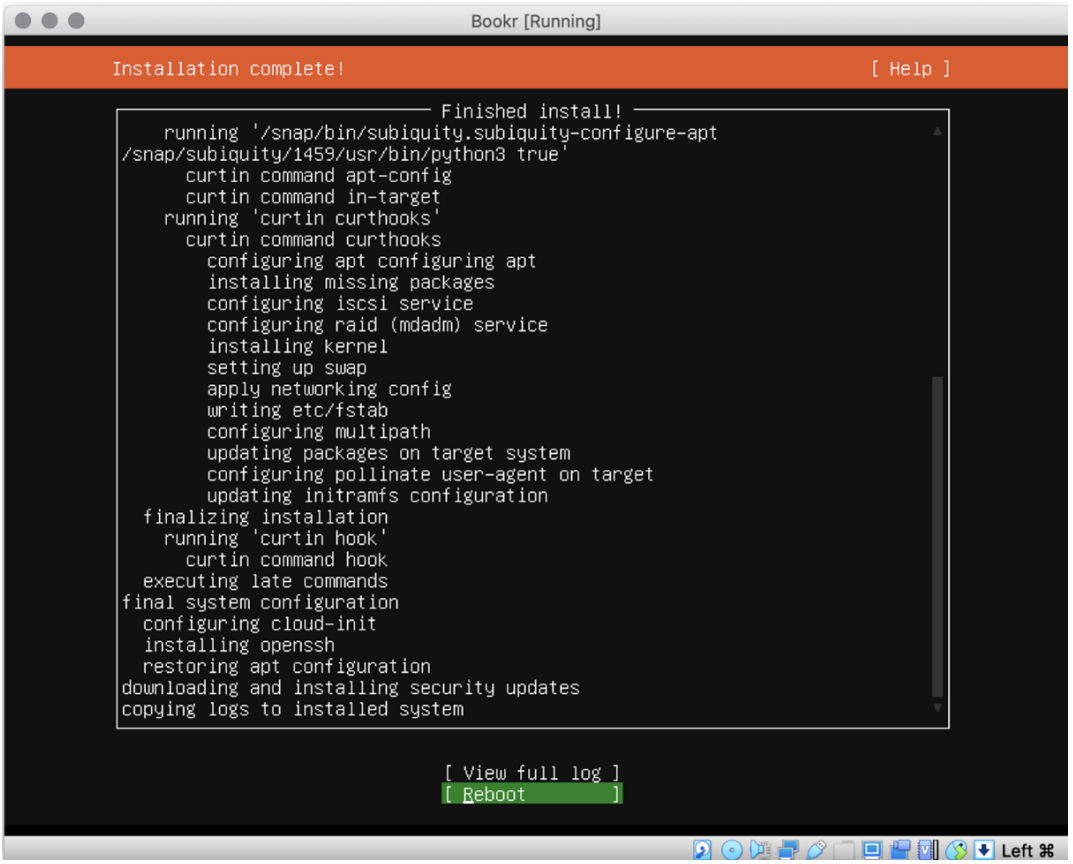

Figure 17.39: Installation complete

Select **`Reboot`** and press *Enter*.

19. The screen will update and start showing more log messages, then eventually stop and show a **`Please remove the installation medium, then press ENTER`** message:

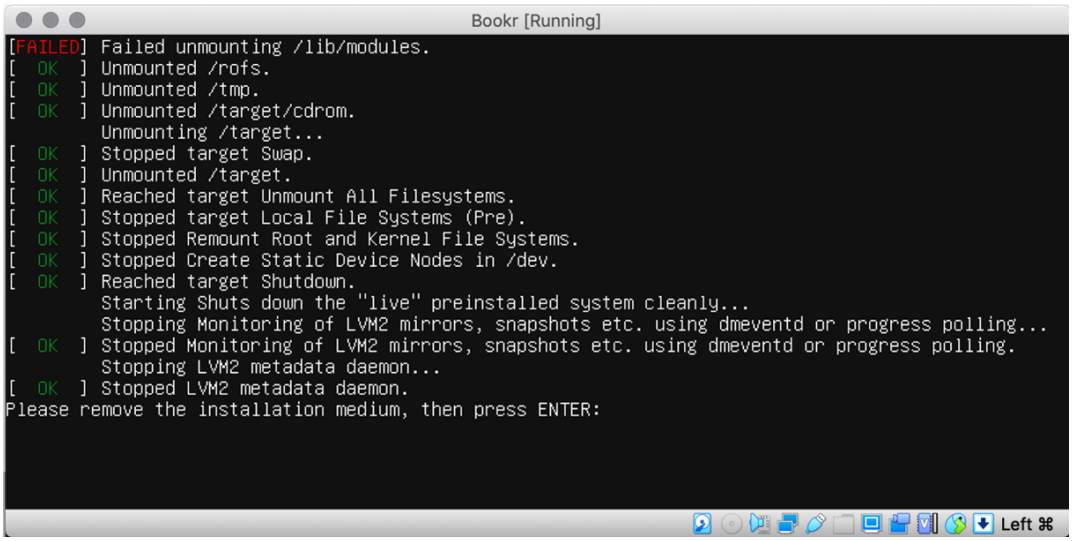

Figure 17.40: Post-installation screen

Note that you might have a **`FAILED`** message, but that's OK.

To "eject" the virtual CD, go back to the **`Oracle VM VirtualBox Manager`** window and click the **`Storage`** text header. This will open the **`Storage`** preferences window, which we saw before, in *step 12*. Once again, click on the CD icon under the **`Controller: IDE`** heading on the left (this time, it should have the name of the Ubuntu ISO). Then, click on the CD in the right-hand-side **`Attributes`** section. From the menu that appears (*Figure 17.41*), choose **`Remove Disk from Virtual Drive`**, then click **`OK`** to close and save the storage settings:

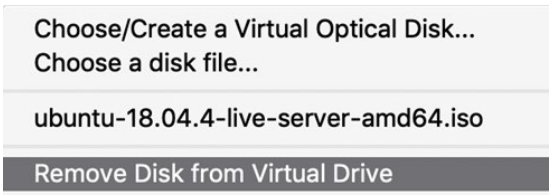

Figure 17.41: Remove Disk from Virtual Drive

20. Switch back to the Bookr virtual display and press *Enter*. Your virtual machine will reboot, and you should see some log messages as it starts up:

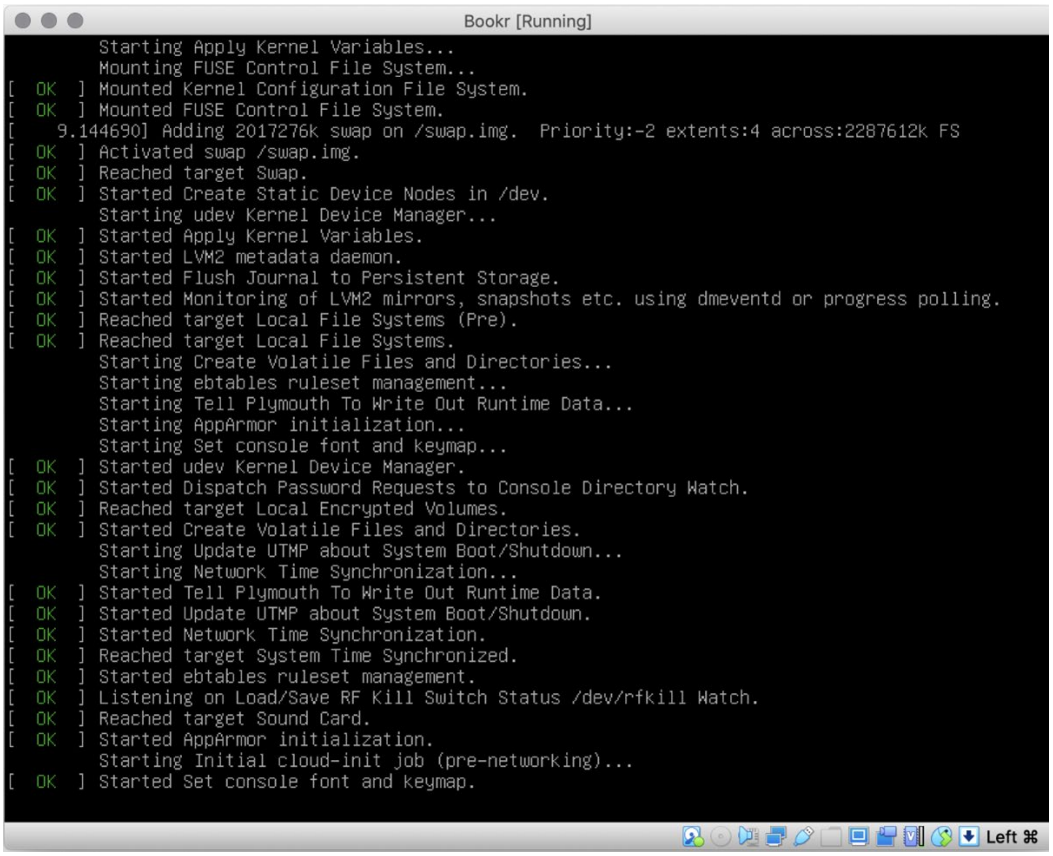

Figure 17.42: Log messages when booting Ubuntu

After a minute or so, you will get the Ubuntu login screen.

Note that if you wait too long before logging in, you might see some **`cloud-init`** messages printed to the screen, such as in *Figure 17.43*:

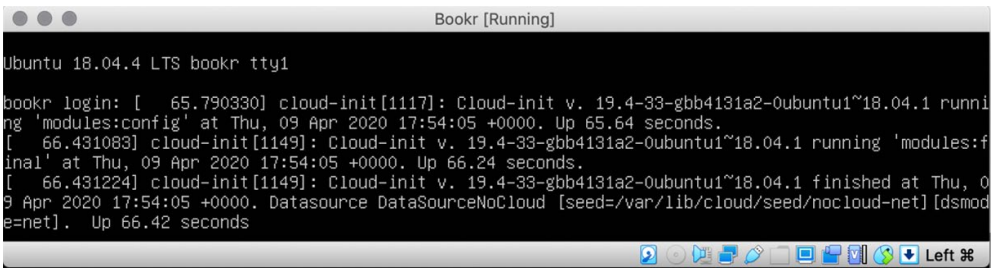

Figure 17.43: cloud-init log messages

This is OK; just press *Enter* to get the login prompt back.

21. Type in the username you created in *step 14*, then press *Enter*. Type in your password (again from *step 14*) – note that you won't see any characters appear as you type. Then, press *Enter* to log in. You will see an output of some system information, and then a command prompt, such as **`yourusername@bookr:~$`**:

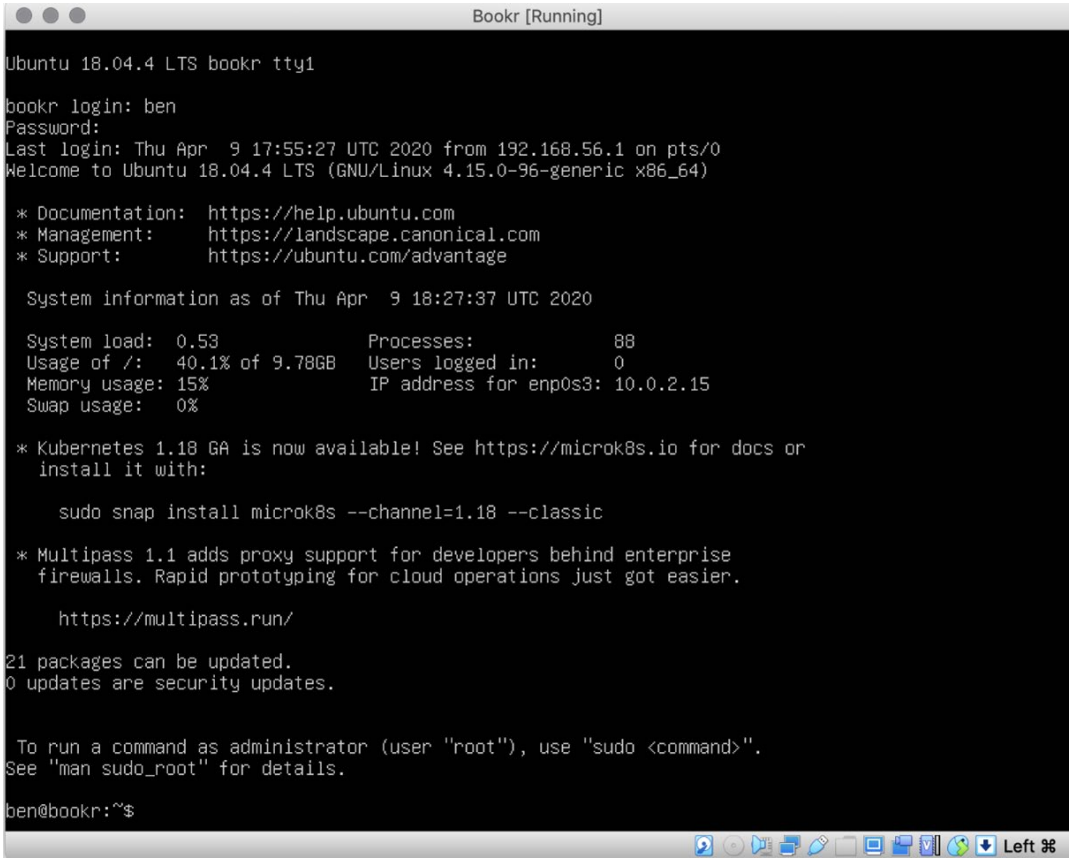

Figure 17.44: Linux command prompt

You've successfully set up an Ubuntu virtual machine. We'll shut it down from inside the virtual machine, and then come back to it later. To shut the machine down, just type the following:

```
shutdown now
```

The next screenshot shows how your virtual screen should look, with the **`shutdown now`** command typed in (but before you've pressed *Enter*):

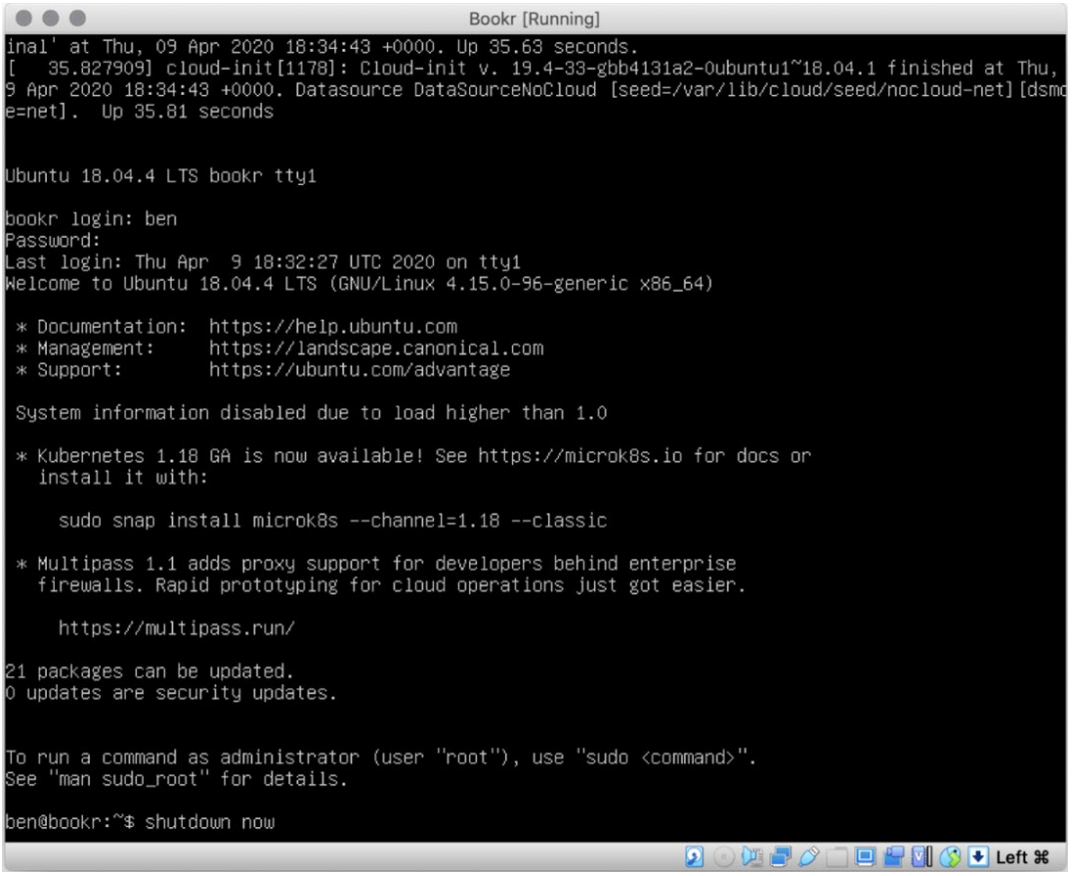

Figure 17.45: The shutdown now command entered

Then, press *Enter*.

This instructs the machine to power off right now. You could also run **`shutdown`** with no argument, which would have the machine turn off in 1 minute, or you could specify a later time.

You'll see some shutdown log messages and then the virtual machine will shut down and its virtual screen will close. In the **`Oracle VM VirtualBox Manager`** window, you'll see that the Bookr virtual machine is now in the **`Powered Off`** state:

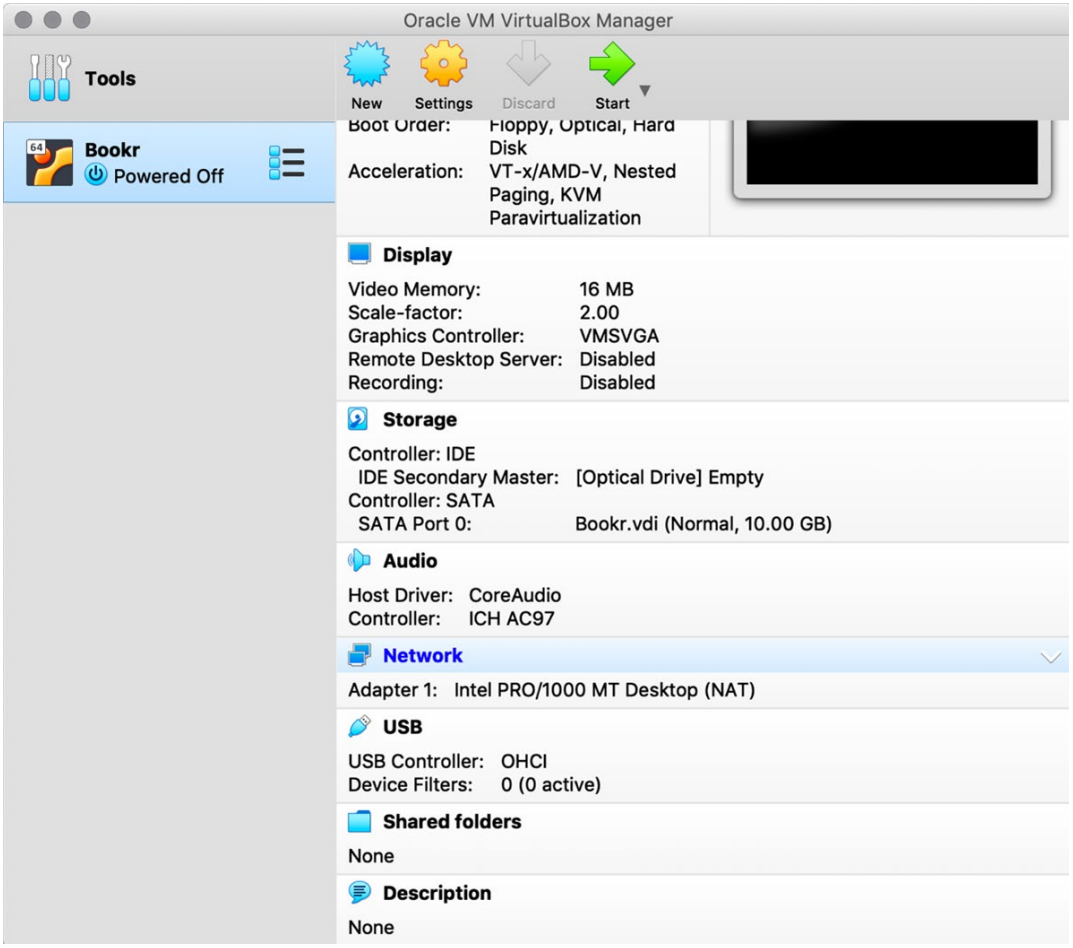

Figure 17.46: The Bookr machine in the Powered Off state

In this exercise, you installed Ubuntu Linux on a VirtualBox virtual machine. This exercise and the previous one were quite long, but you now have an Ubuntu Linux virtual machine that will mimic that from a cloud provider. You will use it in the rest of the exercises and activities.

SUDO

Later, in *Exercise 17.03*, *VirtualBox Networking Configuration*, we will run a command on the virtual machine using the **sudo** command. First, let's do a brief rundown of some common Linux permissions. We created a Linux user during the virtual machine setup in *Exercise 17.02*, *Ubuntu Linux Installation*. This is a normal, non-administrative user. There are some commands and files that this user can't access and that are only available to the *superuser*. The superuser has the username **root**, and it's more common to refer to this user as *root*; for example, "this command needs to be run as root" or "only the root user can access these files." The root user has its own password.

Without going into too much detail, the **sudo** command allows you to execute (**do**) commands as though you were the superuser (**su**). You authenticate using your own password instead of root's password. This is more secure than having a root password that is shared among different users, and more fine-grained access can be given to certain users and/or commands.

To run a command as a superuser, you simply prefix it with **sudo**. For example, the **whoami** command simply tells you the current user executing the command (highlighted):

```
ben@bookr:~$ whoami
ben
ben@bookr:~$
```

Next, we can try executing **whoami** using **sudo**. We will be prompted to enter our password. Then, after hitting *Enter*, we'll see it output **root**:

```
ben@bookr:~$ sudo whoami
[sudo] password for ben:
root
ben@bookr:~$
```

After executing a command as **sudo**, our authentication is cached in the current terminal for 5 minutes, so if we execute a command using **sudo** again, we aren't prompted for our password. This applies even if it's a different command.

For example, if we then run **sudo id** (**id** is like **whoami** but shows more information about your user ID and the groups to which you belong), we see root's information without being prompted for a password:

```
ben@bookr:~$ sudo id
uid=0(root) gid=0(root) groups=0(root)
ben@bookr:~$
```

To expire our authentication, we can do any of the following:

- Log out then log back in.
- Wait 5 minutes.
- Run **sudo -k**:

```
ben@bookr:~$ sudo -k
ben@bookr:~$ sudo whoami
[sudo] password for ben:
root
ben@bookr:~$
```

That's all you need to know about **sudo** for now. We will use it in the next exercise to shut down the virtual machine.

BUILT-IN MANUALS (THE MAN COMMAND)

Linux (and other Unix-like operating systems, such as macOS) has a built-in manual system. It can be accessed in Terminal using the **man** command. It accepts an argument that is the program or function whose manual page you want to read.

For example, to read information about the **whoami** command, use the following:

```
man whoami
```

The following figure shows this code running in Terminal:

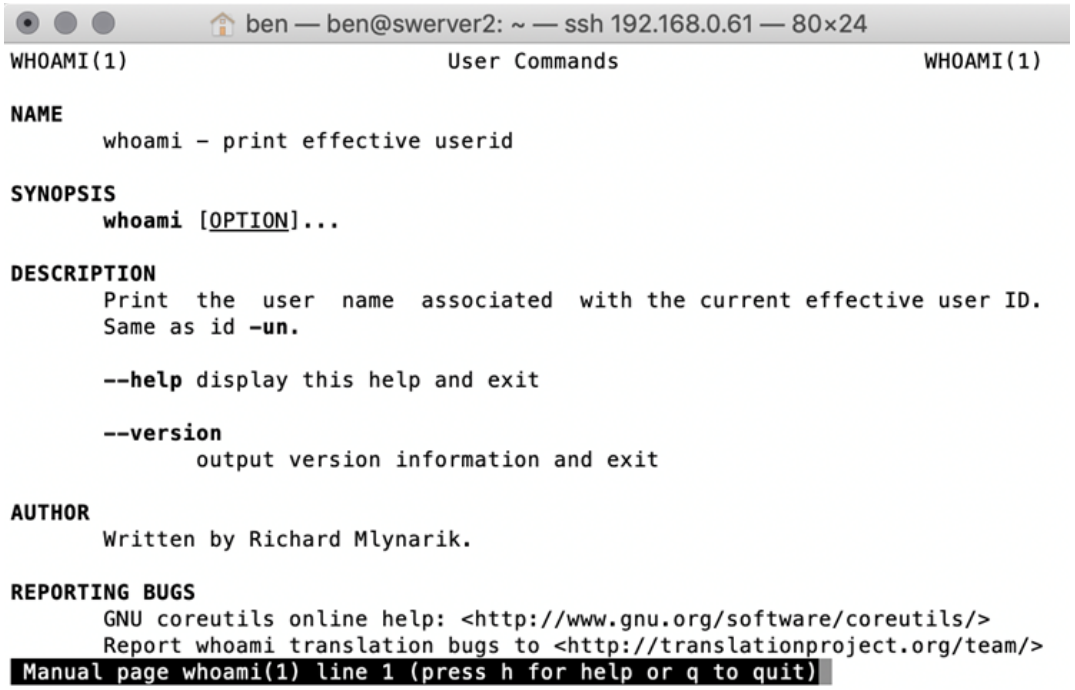

Figure 17.47: Screenshot of man whoami running in Terminal

The information that is displayed is known as a manual page, or "man page." We would say that *Figure 17.47* shows the man page for **`whoami`**. You can even run **`man man`** to read the man page for **`man`** itself.

SSH

SSH, which stands for **secure shell**, is a protocol for connecting to a remote computer and running commands on it from a local system. Using the terminal interface we've been working with in this chapter, SSH allows us to run commands on a remote server, such as a virtual machine or a virtual server in the cloud.

So far, when working with our virtual machine, we have done so using the virtual screen. This is fine for running simple commands, but the virtual screen has some limitations. Copy and paste, text selecting, and scrolling all don't work very well. We also can only have one visible session at a time, which can slow down our workflow. When we begin to work with a virtual server, the virtual screen may be even more difficult to work with, and our provider may offer limited functionality. For these reasons, we will use SSH to communicate with our virtual machine from now on. We already set up an SSH server on the virtual machine when we installed Ubuntu; we just need to get an SSH client set up and connected to it. On macOS and Linux, SSH is built in. On Windows, we will have to install an SSH client. We will use an SSH client called **PuTTY**.

VirtualBox offers different methods of networking, which you can change depending on what works for your situation. When our virtual machine was set up, VirtualBox defaulted it to using the **NAT** (**Network Address Translation**) networking type. This allows connections out from the virtual machine but does not allow new connections in. This is the default because it is the most likely to work in all configurations.

Since we need to connect to the virtual machine, we will need to make some changes to the network settings. The easiest method is to switch the virtual machine to use the **`Bridged Adapter`** setting. This will make the virtual machine behave as if it were just another computer on our network. It will have its own IP address that is separate from our computer. The only issue with this approach is that your network must have its own DHCP server that will automatically assign an IP address to computers that are attached. If it does, then this is the preferred method as it will most closely mimic the setup of a remote virtual server.

The other option is to leave the network in NAT mode and to forward the ports we want to use into the virtual machine. For example, SSH runs on port **`22`**, and HTTP on port **`80`**, so both of these ports should be forwarded into the virtual machine. Then, any connections to a host computer's IP address on those ports will be forwarded to the virtual machine. This has a few more steps to set up, and it can interfere with those ports if we already have those services running on our host machine.

In the next exercise, you will make these settings changes and then test whether you can connect to your SSH server running on the virtual machine from your host machine. If you're familiar with how your virtual server's network is configured already or are using different virtualization software or a server from a cloud provider, you can skip this exercise.

EXERCISE 17.03: VIRTUALBOX NETWORKING CONFIGURATION

In this exercise, you will make changes to the VirtualBox configuration in order to get its networking set up. This will allow you to connect to the virtual machine using SSH, and later with a web browser when you're running Django on it. Depending on your operating system and which networking settings work, you may be able to skip some steps. At the end of the exercise, you will know how to connect to your virtual machine using SSH:

1. You first need to make sure you have an SSH client installed. On macOS or Linux, you should already have one installed with the system. To confirm it, open a terminal and type **`ssh`**, then press *Enter*. You should see output like the following:

```
$ ssh
usage: ssh [-46AaCfGgKkMNnqsTtVvXxYy] [-B bind_interface]
           [-b bind_address] [-c cipher_spec] [-D [bind_address:]
port]
           [-E log_file] [-e escape_char] [-F configfile] [-I pkcs11]
           [-i identity_file] [-J [user@]host[:port]] [-L address]
           [-l login_name] [-m mac_spec] [-O ctl_cmd] [-o option] [-p
port]
           [-Q query_option] [-R address] [-S ctl_path] [-W
host:port]
           [-w local_tun[:remote_tun]] destination [command]
```

   This shows all the options available to configure how SSH connects.

   If you are on Windows, you will need to download and install PuTTY from https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html.

> **NOTE**
>
> If you have WSL set up on Windows 10, you can install the SSH client provided as part of WSL and then execute it by running the **`ssh`** command, the same as for macOS or Linux. You can then follow the instructions as if you were on Linux and skip the PuTTY installation and setup.

2. Now, we can make changes to the VirtualBox configuration. Launch VirtualBox if it's not already running. You should see the **`Oracle VM VirtualBox`** window with the Bookr virtual machine selected in the left panel. The virtual machine should still be shut down from the previous exercise.

   In the right panel, click the **`Network`** header to open the network settings.

   In the network settings window that opens, click the **`Attached to`** menu, which should currently be set to **`NAT`**. Select **`Bridged Adapter`** (*Figure 17.48*):

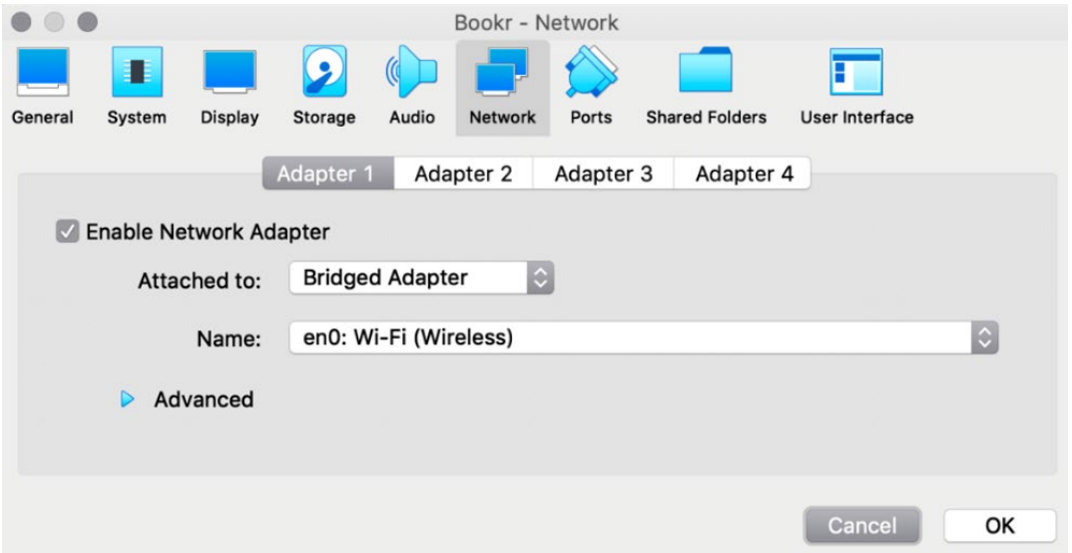

Figure 17.48: Bridged Adapter and the host interface selected

   The settings will update to now also show a **`Name`** field, which allows you to select the physical network device to bridge to.

   If your computer has multiple network adapters, select the correct one from this menu. Then, click **`OK`**.

3. The virtual machine manager window will be shown again, and the **`Network`** settings will have been updated to show that it is now in **`Bridged Adapter`** mode:

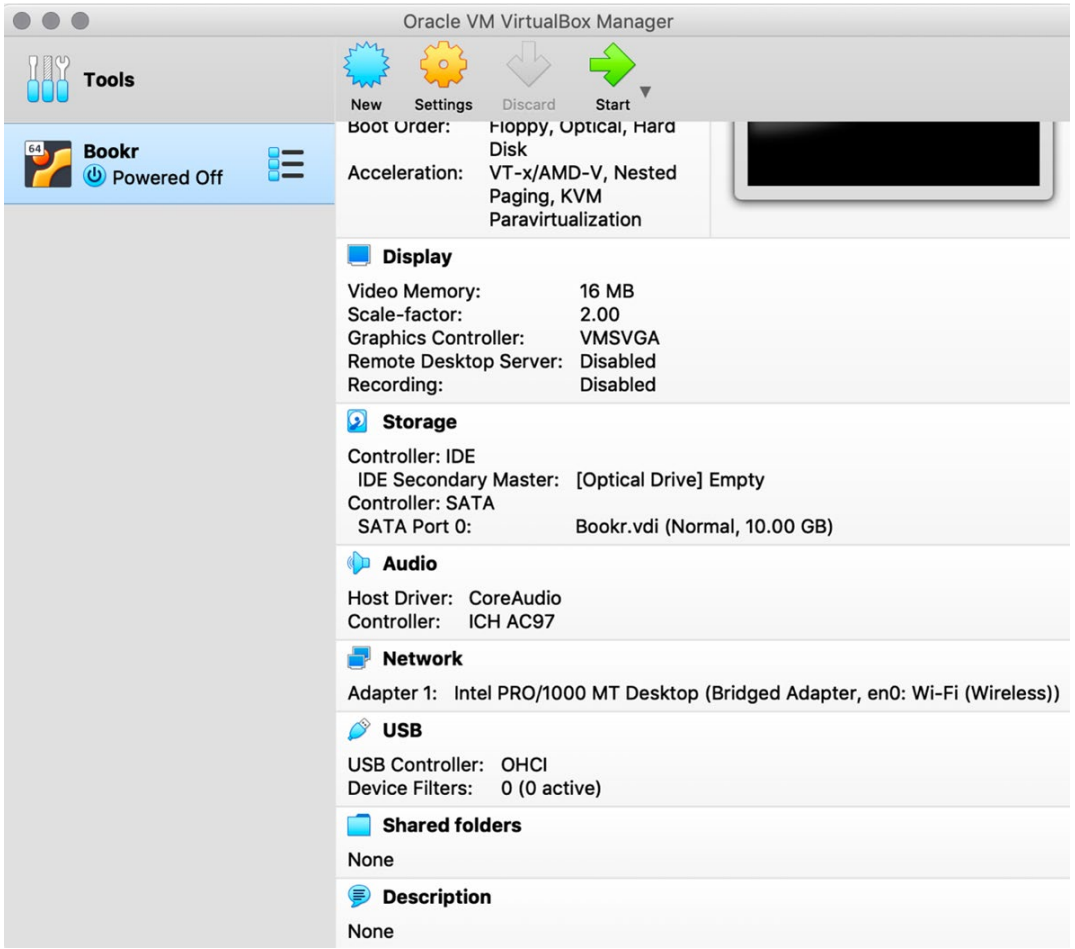

Figure 17.49: Network settings displaying Bridged Adapter

Click **`Start`** at the top of the window to start the virtual machine.

4. After the virtual machine starts up, log in to it using your username and password, as you did in *Exercise 17.02*, *Ubuntu Linux Installation*. You'll need to find out the IP address that the virtual machine now has. When you have a command prompt, run the **`ifconfig`** command (short for (network) interface configuration) to see the virtual machine's IP address.

   Type **`ifconfig`** and then press *Enter*. You should see an output as in *Figure 17.50*:

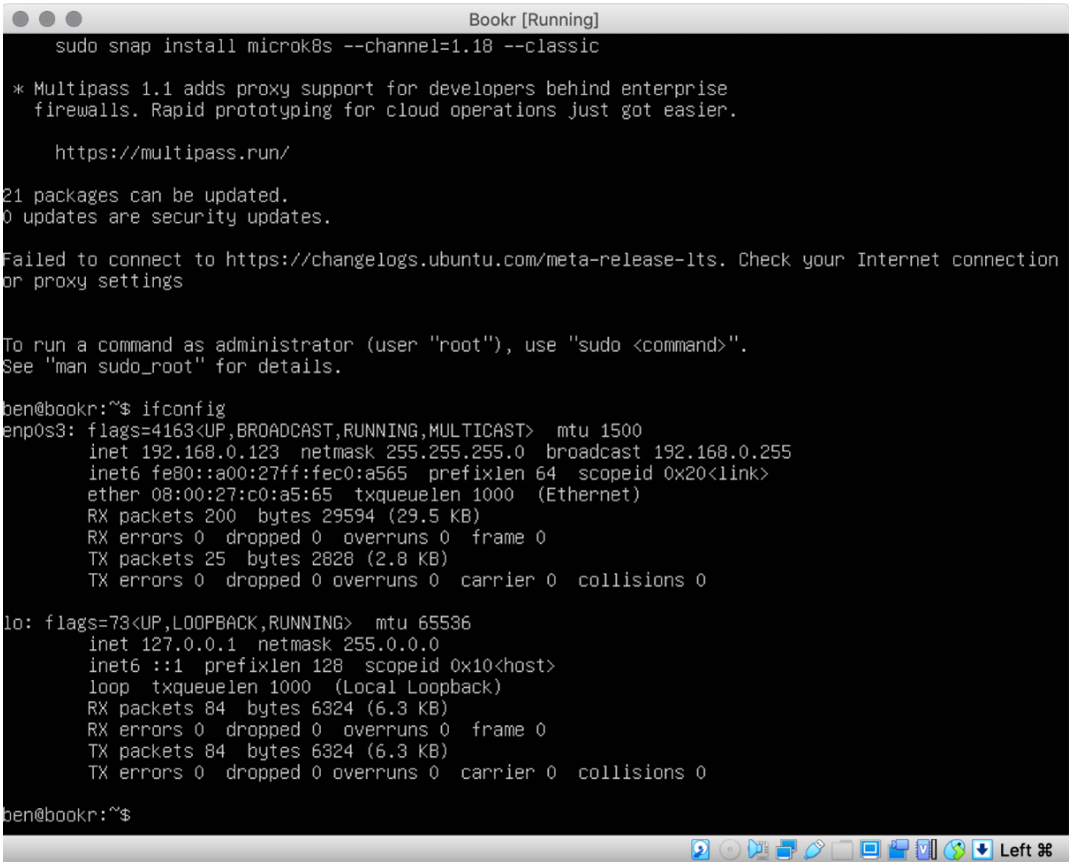

Figure 17.50: ifconfig output

   The IP address is listed to the right of **`inet`** on the second line of the output. It should be something like **`192.168.x.y`**, **`10.x.y.z`**, or **`172.x.y.z`**. If you do not see any of these listed, then it is possible that your computer's connection is not configured with DHCP and you will need to use NAT with port forwarding. You can skip forward to *step 9*, where you'll configure NAT.

5. If your virtual machine does have an IP address, you can now try connecting to it using SSH. On macOS or Linux, switch to Terminal and run the following:

```
ssh <username>@<ip address>
```

Here, **<username>** is the short username you used when setting up the virtual machine, and **<ip address>** is the IP address of your virtual machine that you got in the previous step. For the virtual machine set up in this example chapter, it would be the following:

```
ssh ben@192.168.0.123
```

> **NOTE**
>
> Note that the username (highlighted) does not need to be provided if the username on your host machine matches that of your virtual machine. In that case, you can just do **ssh <ip address>** – for example, **ssh 192.168.0.123**.

After you press *Enter*, **ssh** will try to connect to the virtual machine. You will be asked whether you trust the private key from the server, which is displayed as follows:

Figure 17.51: Server private key trust prompt

This is only done the first time you connect to your virtual machine on this IP address (if the IP address changes, you will be prompted again).

Type **`yes`**, then press *Enter*.

You will be prompted for your password, which you created in the previous exercise (*step 14* in *Exercise 17.02*, *Ubuntu Linux Installation*). Type it in (you won't see any output while you type), then press *Enter* (*Figure 17.52*):

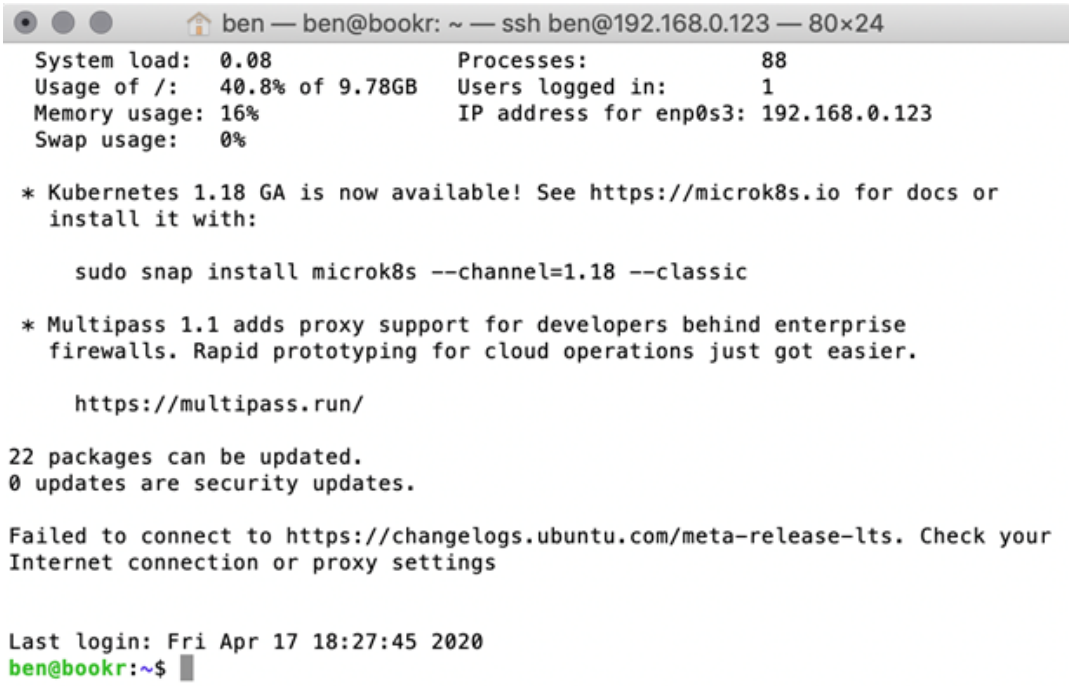

Figure 17.52: After a successful SSH connection

After logging in successfully, you will see the same output as when you logged in using the virtual screen. Then, you'll get a command prompt.

6. If you're on Windows, open PuTTY. You'll see the **`PuTTY Configuration`** window. Enter the IP address you found in *step 4* into the **`Host Name (or IP address)`** field. Then, click **`Open`** at the bottom of the window:

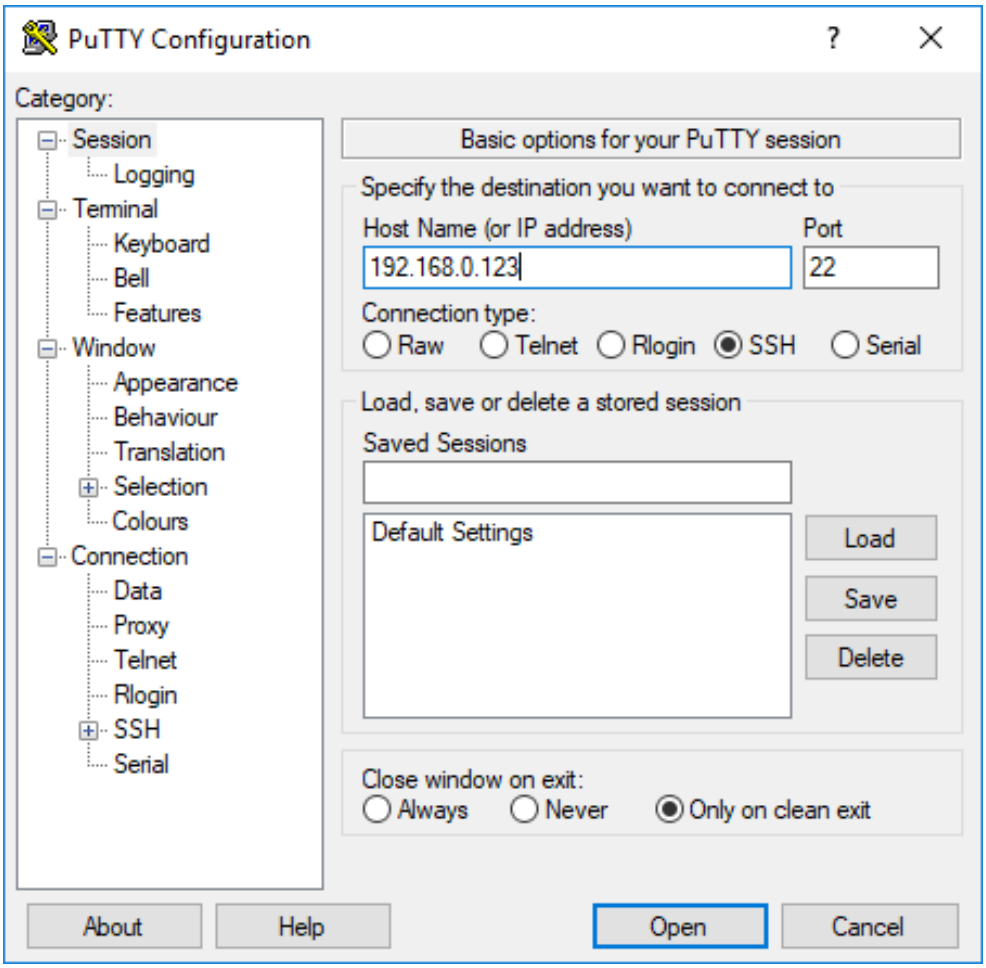

Figure 17.53: PuTTY Configuration window

7. Since this is the first time connecting to the virtual machine, you will see the **`PuTTY Security Alert`** window, which displays the server's key:

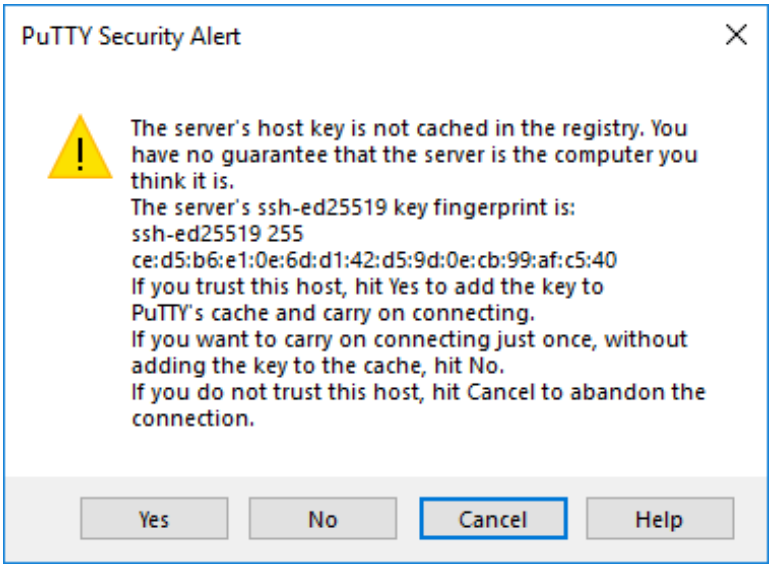

Figure 17.54: PuTTY Security Alert window

Click **`Yes`**.

You'll then see a terminal window that will first prompt you who to log in as. Enter the username you created in the previous exercise, then press *Enter*.

It will then prompt you for your password. Type your password (you won't see anything on screen as you type), then press *Enter* again:

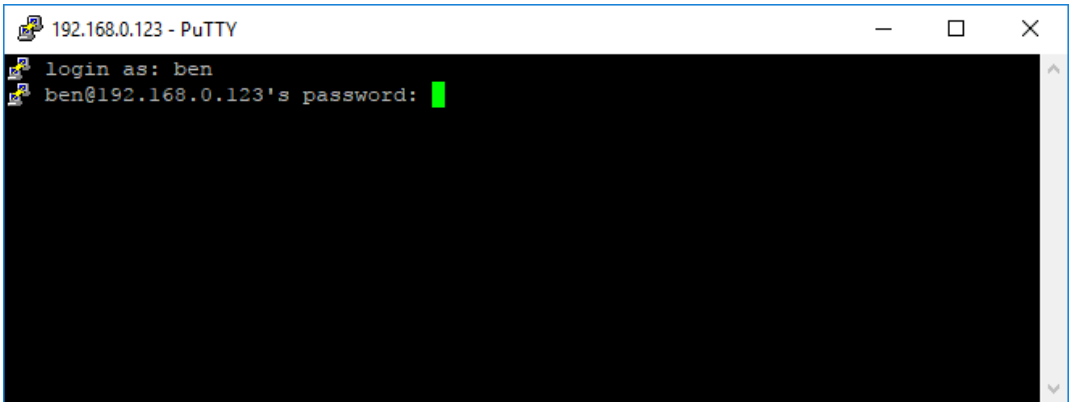

Figure 17.55: PuTTY login/password prompt

After logging in successfully, you will see the same output as when you logged in using the virtual screen. Then, you'll get a command prompt as shown in the following figure:

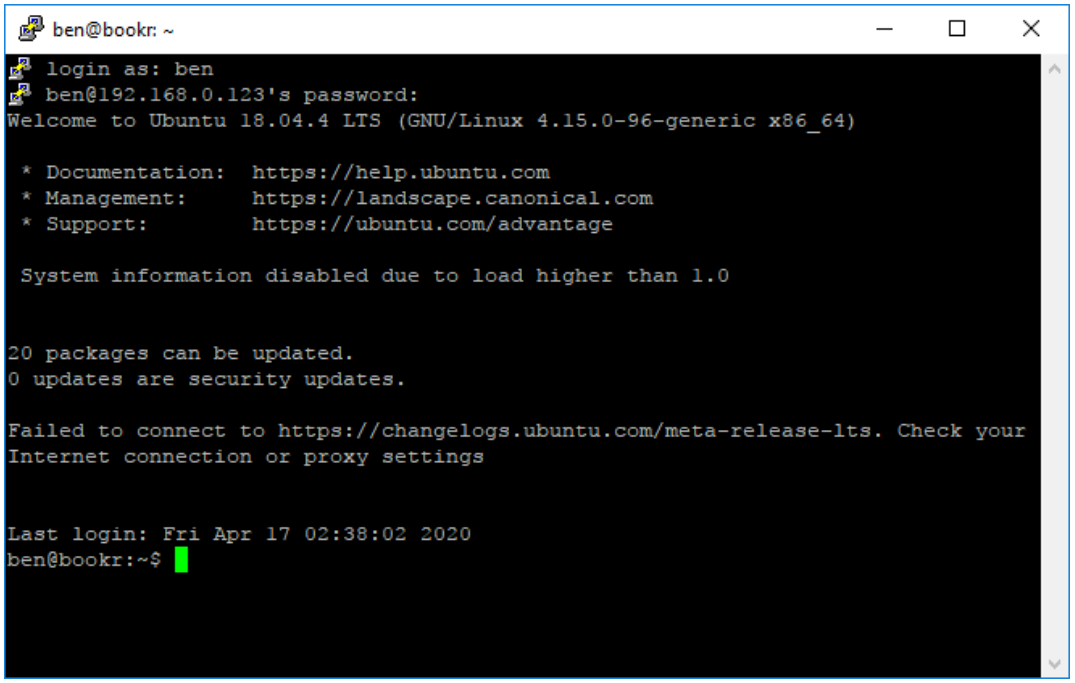

Figure 17.56: Prompt after logging in using PuTTY

8. Regardless of your operating system, if you were able to log in and see a command prompt, then this means everything is working fine. Make sure you record the IP address of the virtual machine because you'll need it every time you have to log in to the server, but for now, we know that it works. You can skip ahead to the final step, *step 17*.

9. If for some reason the virtual machine did not get an IP address or the SSH connection failed, you will need to try to connect using NAT. Switch back to the VirtualBox virtual screen, where you should still be at the command prompt where you ran the **`ifconfig`** command. Shut down the virtual machine by running the **`shutdown now`** command.

10. In the **`Oracle VM VirtualBox Manager`** window, click the **`Network`** heading text (as you did in *step 2*).

11. In the **Network** settings window, change the **Attached to** setting back to **NAT** (not **NAT Network**):

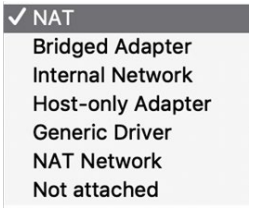

Figure 17.57: Attached to changed back to NAT

12. You now need to forward ports so that connections to your computer will go through to the virtual machine. Click the **Advanced** arrow to expand the advanced settings section:

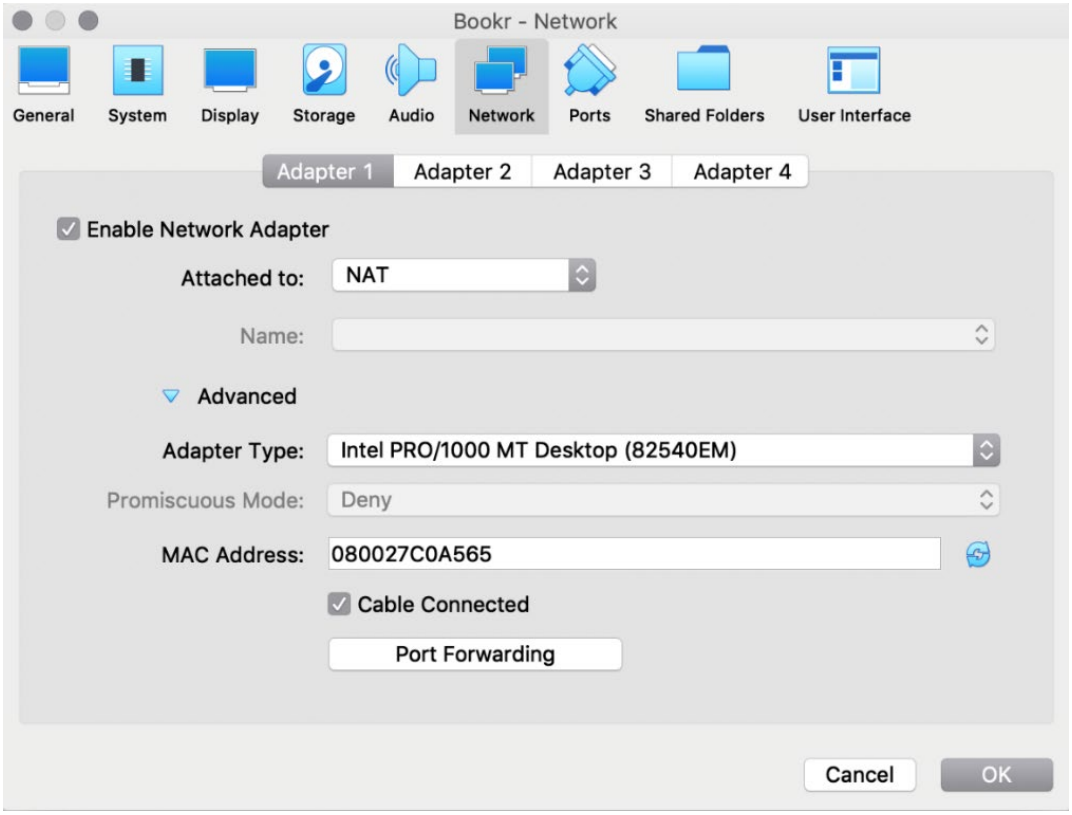

Figure 17.58: Advanced network settings opened

13. Click the **`Port Forwarding`** button to open the port forwarding settings. Then, click the **`Add`** button in the top-right corner. You will see a blank port forwarding configuration row displayed.

    Refer to *Figure 17.59* of *step 14* to see what this window looks like. Enter the following values:

- **`Name`**: **`SSH`**.
- **`Protocol`**: Select **`TCP`**.
- **`Host Port`**: **`22`**.
- **`Guest Port`**: **`22`**.

    The other fields (**`Host IP`** and **`Guest IP`**) should be left blank.

    This will enable the forwarding of SSH.

14. Repeat *step 13* to add another port forwarding line. This time enter **`HTTP`** for **`Name`**, and **`80`** for both **`Host Port`** and **`Guest Port`**. When you've completed *Step 13* and this step, the port forwarding settings should look as in *Figure 17.59*:

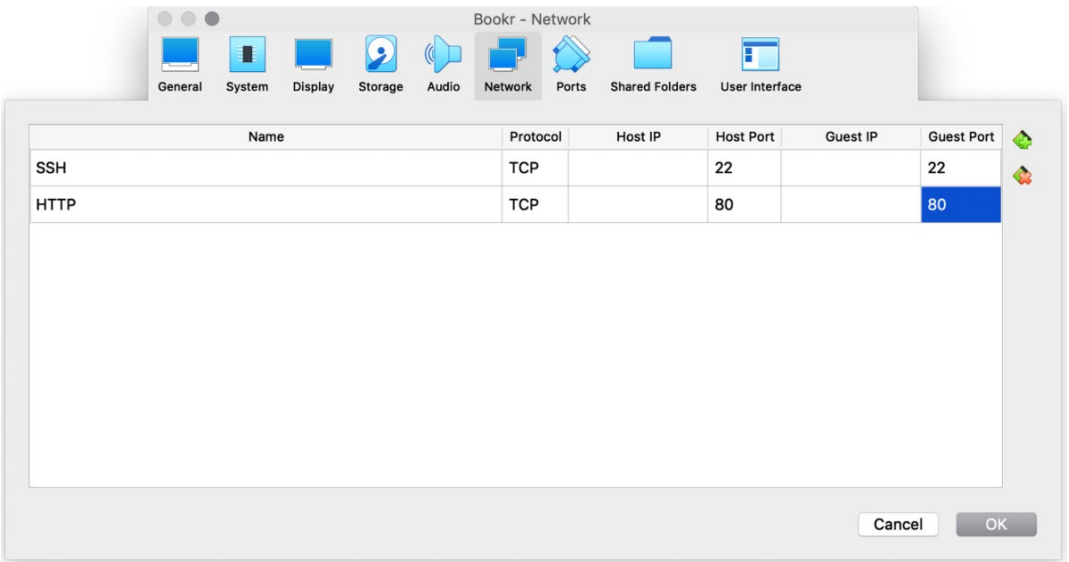

Figure 17.59: Port forwarding settings

Click **OK** to close and save the port forwarding settings, then click **OK** in the **Network** settings window to save and close it.

15. Start the virtual machine by clicking **Start** in the **Oracle VM VirtualBox Manager** window.
16. Once it has started up, you don't need to log in to the virtual screen; instead, switch back to Terminal (macOS/Linux) or PuTTY (Windows) and try to connect (repeat *steps 5* or *6*) using the **localhost** address. This won't connect to your host computer; since the port is forwarded, it will go to the virtual machine instead.

    You should now be logged in and see the command prompt as in *Figure 17.52* (macOS/Linux) or *Figure 17.56* (Windows).
17. You can now try out the **sudo** command to shut down the virtual machine.

> **NOTE**
>
> When we were using the virtual screen inside VirtualBox, we could run the **shutdown** command without using **sudo**. This is because it was like we were sitting down in front of a real computer and using a keyboard and monitor connected to it. Anyone in this position can run **shutdown** because even if they did not have software access, they still have hardware access to just turn the computer off with a power switch.
>
> When we connect over SSH, we are not physically at the machine and so the **shutdown** command is limited to root.

Shut down your virtual machine now by typing **`sudo shutdown now`**, then press *Enter*. You will be prompted to enter your password. Type it in (nothing will be shown on screen), then press *Enter*:

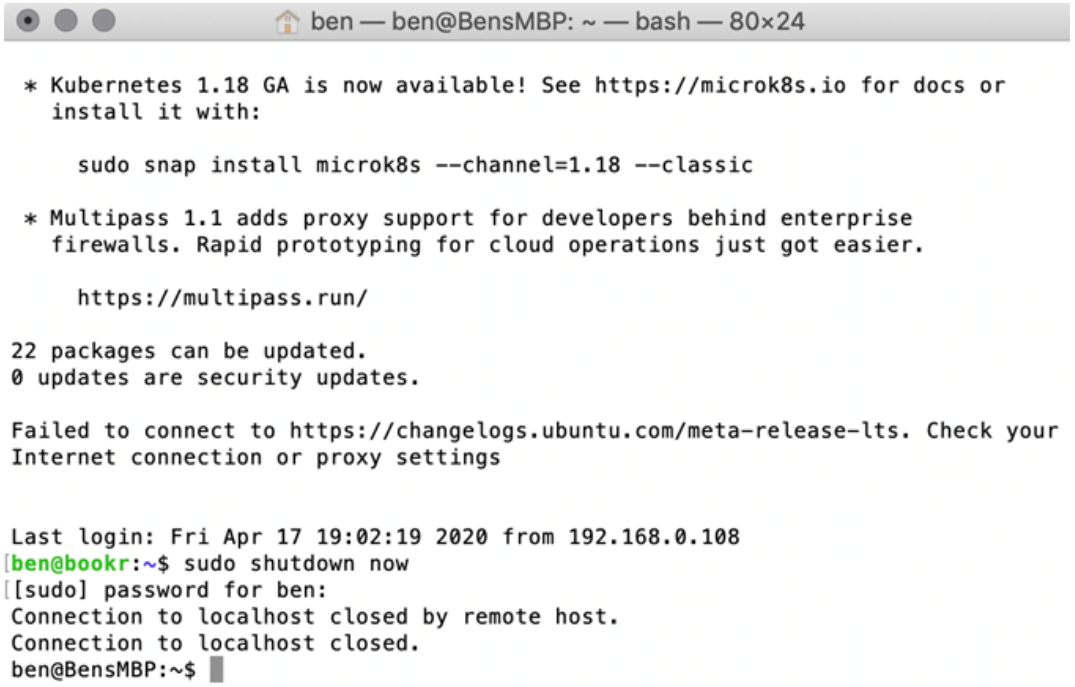

Figure 17.60: The shutdown command executed over SSH

The virtual machine will shut down and your SSH session will be disconnected.

In this exercise, you configured networking for the virtual machine and used SSH to connect to it. Your virtual machine may use either bridged or NAT networking depending on which setting worked.

If you had set up a virtual server with a cloud provider, it would be in this current state, so the steps and exercises going forward apply to both your local virtual machine and the remote virtual private server hosted in the cloud.

In the next section, you will learn how to install packages and keep your system up to date using **`apt`**. You'll understand the various system packages that you need and what they do. Then, you'll log back into your virtual machine and install them.

PACKAGES AND UPDATES

Different distributions of Linux have different systems for installing and updating packages. Ubuntu (and other distributions that are also based on Debian) uses a system called **APT** (**Advanced Package Tool**). This is a system for installing and updating packages, as well as automatically installing dependencies and default configuration files. It is also able to run pre- and post-install scripts to perform extra setup.

There are a few different ways of interacting with the APT system; we will use the **`apt`** command-line tool.

Since **`apt`** needs to make changes to the system, it must be run using **`sudo`**. Here's a quick rundown of a few common commands:

- **`install`**: This is used to install a particular package:

```
sudo apt install <packagename>
```

  This will also first install the packages on which the installed package(s) depend. You can also install multiple packages at once, by listing them all, for example, **`sudo apt install package1 package2...`**.

- **`remove`**: This will uninstall a package:

```
sudo apt remove <packagename>
```

  It can take multiple package names, as the **`install`** command does. This will not remove the package's dependencies (see **`autoremove`**).

- **`purge`**: When **`apt`** removes a package (using the **`remove`** command), it won't delete any configuration files that the package installed, since you might have made changes to them that you don't want to lose. Or you might need to remove an old version of a package and reinstall a new one that uses the same config files. Whatever the reason, in case you do also want to clean up everything, **`apt`** provides the **`purge`** command. **`purge`** is used in the same way as **`remove`** but will also delete configuration files:

```
sudo apt purge <packagename>
```

- **`update`**: This refreshes the list of available packages, including new packages and upgrades. It does not take any arguments:

```
sudo apt update
```

  The package list must be updated before trying to upgrade packages; otherwise, the local system will not know that upgrades exist.

- **`upgrade`**: This command upgrades all the installed packages to their latest version:

```
sudo apt upgrade
```

  This does not take any arguments; that is, you can't use it to upgrade just one installed package. Instead, just **`apt install`** a package again, and only that package will be upgraded.

- **`full-upgrade`**: Similar to **`upgrade`**, however, when using **`upgrade`**, new dependencies are not installed. For example, you might install **`package-a`**, which is at version **`1.0`** and has no dependencies. Later, a new version, **`2.0`**, is released, which now has a dependency of **`package-b`**. Running **`upgrade`** will not upgrade **`package-a`** because it must install the **`package-b`** dependency. Running **`full-upgrade`** will first install **`package-b`**, and then version **`2.0`** of **`package-a`**.

  Like **`upgrade`**, it takes no arguments:

```
sudo apt full-upgrade
```

- **`autoremove`**: When **`apt`** removes a package (using the **`remove`** command), it leaves that package's dependencies installed. The **`autoremove`** command will remove all the packages that have been installed as dependencies but whose dependent packages have been removed.

  This command takes no arguments:

```
sudo apt autoremove
```

In this book, we will only be using the **`update`** and **`install`** commands, although knowing the others that have been covered will give you a good basis for continuing to administer the Ubuntu system.

Ubuntu comes with some of the software that we need already installed, for example, Python 3. To run a Django app, we'll need some other supporting packages:

- **`nginx`**: This package installs the NGINX web server for serving static files and proxying requests to Gunicorn.
- **`postgresql`**: The SQL server that replaces the single-file SQLite database we were using during development.

- **`python3-virtualenv`**: This adds the Python **`virtualenv`** package to the system. Once this is installed, we'll be able to create Python virtual environments. Further Python packages (such as Django) will be installed into the virtual environment using **`pip`**.
- **`libpq-dev`**, **`python3-dev`**, and **`build-essential`**: These provide header files for PostgreSQL, Python 3, and the system. **`build-essential`** also includes compilers and build tools. We'll need these to build the Python PostgreSQL client library.

In the next exercise, we will upgrade the packages that are already installed on the virtual machine. Then, we'll install the packages we just mentioned.

EXERCISE 17.04: PACKAGE UPDATE AND INSTALLATION

In this exercise, you will SSH into your virtual machine, then update the list of available packages/versions using the **`apt update`** command. The installed packages will then be upgraded using **`apt upgrade`**. Finally, we'll use **`apt install`** to install **`nginx`**, **`postgresql`**, **`python3-virtualenv`**, **`libpq-dev`**, **`python3-dev`**, and **`build-essential`**:

1. Start VirtualBox and then start up your virtual machine. Let it finish booting up – you'll know it's finished because you'll see the login prompt.
2. SSH into your virtual machine using the IP address you found in *Exercise 17.03*, *VirtualBox Networking Configuration* (or **`localhost`**).
3. Once you have connected, use the **`apt update`** command to update the list of available packages. The command must be run using **`sudo`**, like this:

```
sudo apt update
```

   You will be prompted to enter your password to authenticate the **`sudo`** process:

```
ben@bookr:~$ sudo apt update
[sudo] password for ben:
```

   Type in your password and then press *Enter*.

   You will see the list of repositories from which **`apt`** is fetching updates, and you should see that several packages are available to be upgraded:

```
ben@bookr:~$ sudo apt update
[sudo] password for ben:
Hit:1 http://nz.archive.ubuntu.com/ubuntu bionic InRelease
Hit:2 http://nz.archive.ubuntu.com/ubuntu bionic-updates InRelease
```

```
Hit:3 http://nz.archive.ubuntu.com/ubuntu bionic-backports InRelease
Hit:4 http://nz.archive.ubuntu.com/ubuntu bionic-security InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
28 packages can be upgraded. Run 'apt list --upgradable' to see them.
ben@bookr:~$
```

4. Upgrade all the installed packages to their latest versions using the **`apt upgrade`** command. Again, this must be run using **`sudo`**:

```
sudo apt upgrade
```

Since you just authenticated to **`sudo`** in the previous step, you shouldn't have to enter your password again (unless it's been more than 5 minutes since you executed *step 3*).

After entering the command, **`apt`** will check which packages are upgradeable and show them to you. It will then ask the following:

```
Do you want to continue? [Y/n]
```

You'll need to then press *Enter* again to confirm that you want to upgrade. **`apt`** will then download and install all the available upgrades (note that your list of upgrades might not exactly match what is shown here, and for brevity, a lot of output has been removed):

```
ben@bookr:~$ sudo apt upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages will be upgraded:
  bsdutils distro-info-data …
28 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 7344 kB of archives.
After this operation, 238 kB of additional disk space will be used.
Do you want to continue? [Y/n]
Get:1 http://nz.archive.ubuntu.com/ubuntu bionic-updates/main amd64
bsdutils amd64 1:2.31.1-0.4ubuntu3.6 [60.3 kB]
Get:2 http://nz.archive.ubuntu.com/ubuntu bionic-updates/main amd64
libuuid1 amd64 2.31.1-0.4ubuntu3.6 [20.1 kB]
…
Fetched 7344 kB in 4s (1720 kB/s)
Preconfiguring packages ...
```

```
(Reading database ... 102499 files and directories currently
installed.)
Preparing to unpack .../bsdutils_1%3a2.31.1-0.4ubuntu3.6_amd64.deb
...
Unpacking bsdutils (1:2.31.1-0.4ubuntu3.6) over (1:2.31.1-
0.4ubuntu3.5) ...
Setting up bsdutils (1:2.31.1-0.4ubuntu3.6) ...
…
Processing triggers for initramfs-tools (0.130ubuntu3.9) ...
update-initramfs: Generating /boot/initrd.img-4.15.0-99-generic
ben@bookr:~$
```

5. Now that all existing packages are up to date, install the packages you need: **`nginx`**, **`postgresql`**, **`python3-virtualenv`**, **`libpq-dev`**, **`build-essential`**, and **`python3-dev`**. This is done using **`apt install`**, and all the package names can be provided to the command at once:

```
sudo apt install nginx postgresql python3-virtualenv libpq-dev build-
essential python3-dev
```

`apt` will calculate the package dependencies and show you a list of all the packages that will be installed. Some of the output has been truncated for brevity:

```
ben@bookr:~$ sudo apt install nginx postgresql python3-virtualenv
libpq-dev build-essential python3-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  dh-python …
Suggested packages:
  g++-multilib …
The following NEW packages will be installed:
  build-essential …
0 upgraded, 34 newly installed, 0 to remove and 19 not upgraded.
Need to get 61.4 MB of archives.
After this operation, 135 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Once again you will be asked whether you want to continue, so press *Enter* to begin the installation.

Depending on the speed of your internet connection, the installation might take a few minutes or more. You will see a lot of output. First will be messages starting with **`Get`**, as follows:

```
Get:1 http://nz.archive.ubuntu.com/ubuntu bionic-updates/main amd64
libjpeg-turbo8 amd64 1.5.2-0ubuntu5.18.04.3 [110 kB]
Get:2 http://nz.archive.ubuntu.com/ubuntu bionic/main amd64 fonts-
dejavu-core all 2.37-1 [1041 kB]
```

This indicates the packages that are being fetched. Next, the packages will be unpacked, with lines like this:

```
Selecting previously unselected package libjpeg-turbo8:amd64.
(Reading database ... 102527 files and directories currently
installed.)
Preparing to unpack .../00-libjpeg-turbo8_1.5.2-0ubuntu5.18.04.3_
amd64.deb ...
Unpacking libjpeg-turbo8:amd64 (1.5.2-0ubuntu5.18.04.3) ...
Selecting previously unselected package fonts-dejavu-core.
Preparing to unpack .../01-fonts-dejavu-core_2.37-1_all.deb ...
```

Next, the packages are set up:

```
Setting up python-pip-whl (9.0.1-2.3~ubuntu1.18.04.1) ...
Setting up libjbig0:amd64 (2.1-3.1build1) ...
Setting up fonts-dejavu-core (2.37-1) ...
```

For some packages (**`postgresql`** in particular), there will be a lot of output about how they are being configured. When **`apt`** installs **`nginx`** and **`postgresql`**, it also sets them up to start when the machine boots, and then starts them after the installation is complete.

The last few lines of output will be the processing of triggers, which are the final scripts to set up the packages:

```
Processing triggers for systemd (237-3ubuntu10.39) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
```

When the installation is finished, you will see your command prompt again.

6. The final step in this exercise is to check that NGINX was correctly installed. Open a web browser on your host machine and go to the address of your virtual machine, for example, **`http://192.168.0.123/`** or **`http://localhost/`** (if using NAT). You should see the default NGINX welcome page:

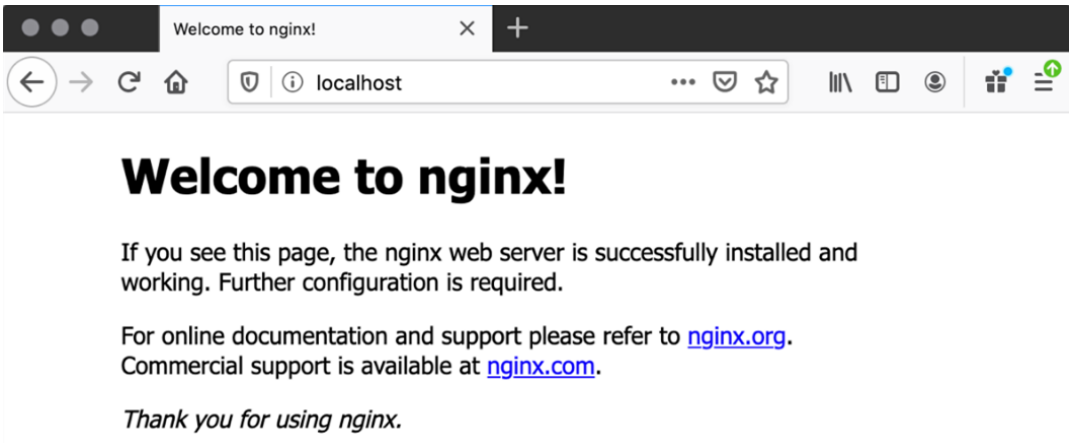

Figure 17.61: Default NGINX screen

   It is not so easy to test that PostgreSQL was set up correctly but unless you saw any errors in the installation output, you should assume it was successful. You'll use PostgreSQL in the first exercise in the next chapter.

7. We will return to the virtual machine in the next chapter. You can shut it down now by running the **`sudo shutdown`** command in the SSH terminal. Or, if you will be continuing on with the next chapter soon, you can leave the session running and return to it.

In this exercise, you first updated the list of available packages and then installed all the available upgrades. Then, you installed the NGINX web server, the PostgreSQL database server, Python 3's **`virtualenv`**, and other supporting build packages. This was all done using the **`apt`** program. You then tested that the NGINX web server was running and that you could connect to it using a web browser.

ACTIVITY 17.01: INSTALLING GLANCES – A SYSTEM-MONITORING UTILITY

In the activity for this chapter, you will reinforce what you have learned by connecting to your virtual machine, then installing and using the **`glances`** system monitoring application all on your own.

`glances` is a terminal-based utility that shows you information about your virtual machine at a glance (hence the name). It can be installed with **`apt`** and has several command-line options. To get you more familiar with using SSH, **`apt`**, and Linux in general, in this activity, you will update your **`apt`** package list, install **`glances`**, then check its command-line options using **`man`**.

These steps will help you complete this activity:

1. Start up your virtual machine and connect to it using SSH.
2. Make sure its list of **`apt`** packages is up to date.
3. Install **`glances`** using **`apt`**.
4. After it's installed, run **`glances`** and have a quick look at its interface. Then, type *Ctrl* + *C* or *Esc* to exit.
5. There are two things you might have noticed when running **`glances`**.

   First, if your terminal has a light background, some of the text can be difficult to read. **`glances`** doesn't have a white theme to fix this.

   Second, there is a lot of information displayed, and you can disable some of the information that is less interesting to you. In this case, as an example, you want to disable the memory swap module.

   Use the **`man`** command to read about the command-line options for **`glances`**. You will be able to find out how to disable the memory swap module and, optionally, set it to the white theme (only if your terminal has a light background).

6. Re-run **`glances`** with the command-line flags to set the options you found out about in *step 5*.

   Once **`glances`** is up and running with the correct options, it should look similar to *Figure 17.62* (although it will look different if your terminal has a dark background):

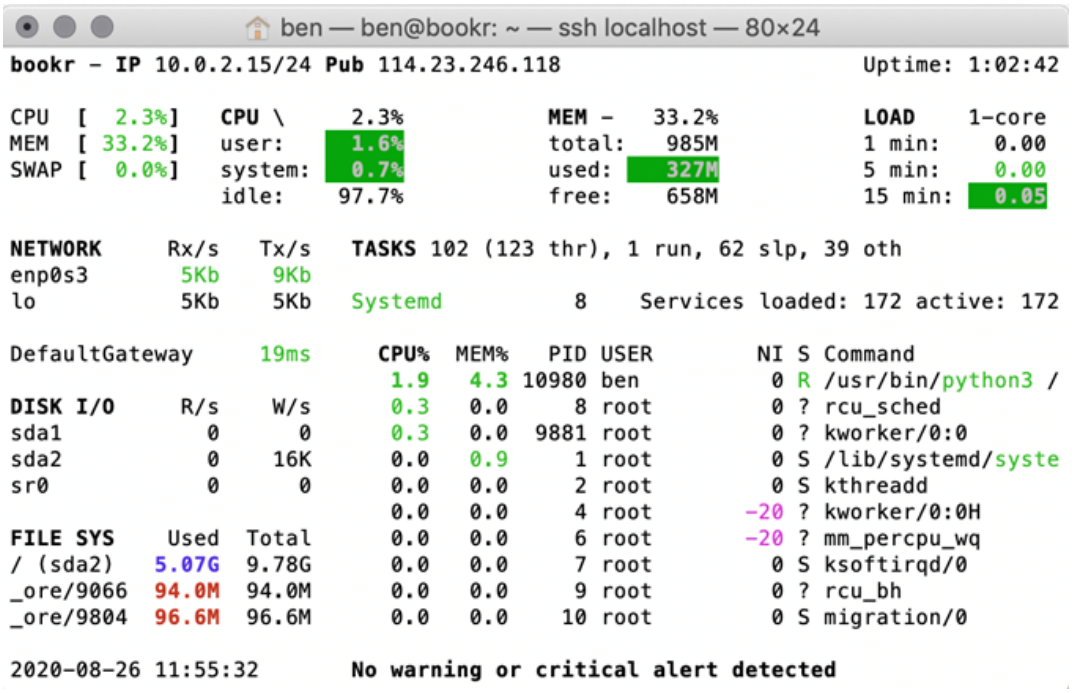

Figure 17.62: glances with a white theme and memory swap module disabled

> **NOTE**
>
> The solution to this activity can be found at http://packt.live/2Nh1NTJ.

SUMMARY

In this chapter, we learned about the architecture used in a production web server, specifically the roles of a frontend server (NGINX), an application server (Gunicorn), and databases (PostgreSQL) in Django hosting. We studied how these three work together to receive, process, and serve requests that the server gets.

We then got Ubuntu Linux up and running on a virtual server, either on a personal computer or through a cloud provider. Later, we connected to this server remotely using SSH. Then, we learned about the **`apt`** command and how it comes in handy when we need to install or remove packages and keep them up to date. Finally, we put all the knowledge we'd gained so far in the chapter into practice as we installed NGINX, Gunicorn, and PostgreSQL, plus supporting packages, on our virtual server remotely using SSH.

The next chapter follows straight on from this one. We will configure our PostgreSQL database and users, configure permissions on our server, upload our application code and production settings, and finish with Bookr up and running on our virtual machine. You can download the next chapter from the GitHub repository for this book at http://packt.live/3pbmqyU.