```
# Update the apt-get cache
sudo apt-get update

# Install PHP and Apache
sudo apt-get install -y php apache2

# Copy the code from the repository
sudo git clone https://github.com/brikis98/php-app.git
/var/www/html/app

# Start Apache
sudo service apache2 start
```

The great thing about ad hoc scripts is that you can use popular, general-purpose programming languages, and you can write the code however you want. The terrible thing about ad hoc scripts is that you can use popular, general-purpose programming languages, and you can write the code however you want.

Whereas tools that are purpose-built for IaC provide concise APIs for accomplishing complicated tasks, if you're using a general-purpose programming language, you need to write completely custom code for every task. Moreover, tools designed for IaC usually enforce a particular structure for your code, whereas with a general-purpose programming language, each developer will use their own style and do something different. Neither of these problems is a big deal for an eight-line script that installs Apache, but it gets messy if you try to use ad hoc scripts to manage dozens of servers, databases, load balancers, network configurations, and so on.

If you've ever had to maintain a large repository of Bash scripts, you know that it almost always devolves into a mess of unmaintainable spaghetti code. Ad hoc scripts are great for small, one-off tasks, but if you're going to be managing all of your infrastructure as code, then you should use an IaC tool that is purpose-built for the job.

Configuration Management Tools

Chef, Puppet, and Ansible are all *configuration management tools*, which means that they are designed to install and manage software on existing servers. For example, here is an *Ansible role* called *web-server.yml* that configures the same Apache web server as the *setup-webserver.sh* script:

```
- name: Update the apt-get cache
  apt:
    update_cache: yes

- name: Install PHP
  apt:
    name: php

- name: Install Apache
  apt:
    name: apache2

- name: Copy the code from the repository
  git: repo=https://github.com/brikis98/php-app.git
dest=/var/www/html/app

- name: Start Apache
  service: name=apache2 state=started enabled=yes
```

The code looks similar to the Bash script, but using a tool like Ansible offers a number of advantages:

Coding conventions

> Ansible enforces a consistent, predictable structure, including documentation, file layout, clearly named parameters, secrets management, and so on. While every developer organizes their ad hoc scripts in a different way, most configuration management tools come with a set of conventions that makes it easier to navigate the code.

Idempotence

> Writing an ad hoc script that works once isn't too difficult; writing an ad hoc script that works correctly even if you run it over and over again is much harder. Every time you go to create a folder in your script, you need to remember to check whether that folder already exists; every

time you add a line of configuration to a file, you need to check that line doesn't already exist; every time you want to run an app, you need to check that the app isn't already running.

Code that works correctly no matter how many times you run it is called *idempotent code*. To make the Bash script from the previous section idempotent, you'd need to add many lines of code, including lots of if-statements. Most Ansible functions, on the other hand, are idempotent by default. For example, the *web-server.yml* Ansible role will install Apache only if it isn't installed already and will try to start the Apache web server only if it isn't running already.

Distribution

Ad hoc scripts are designed to run on a single, local machine. Ansible and other configuration management tools are designed specifically for managing large numbers of remote servers, as shown in Figure 1-2.

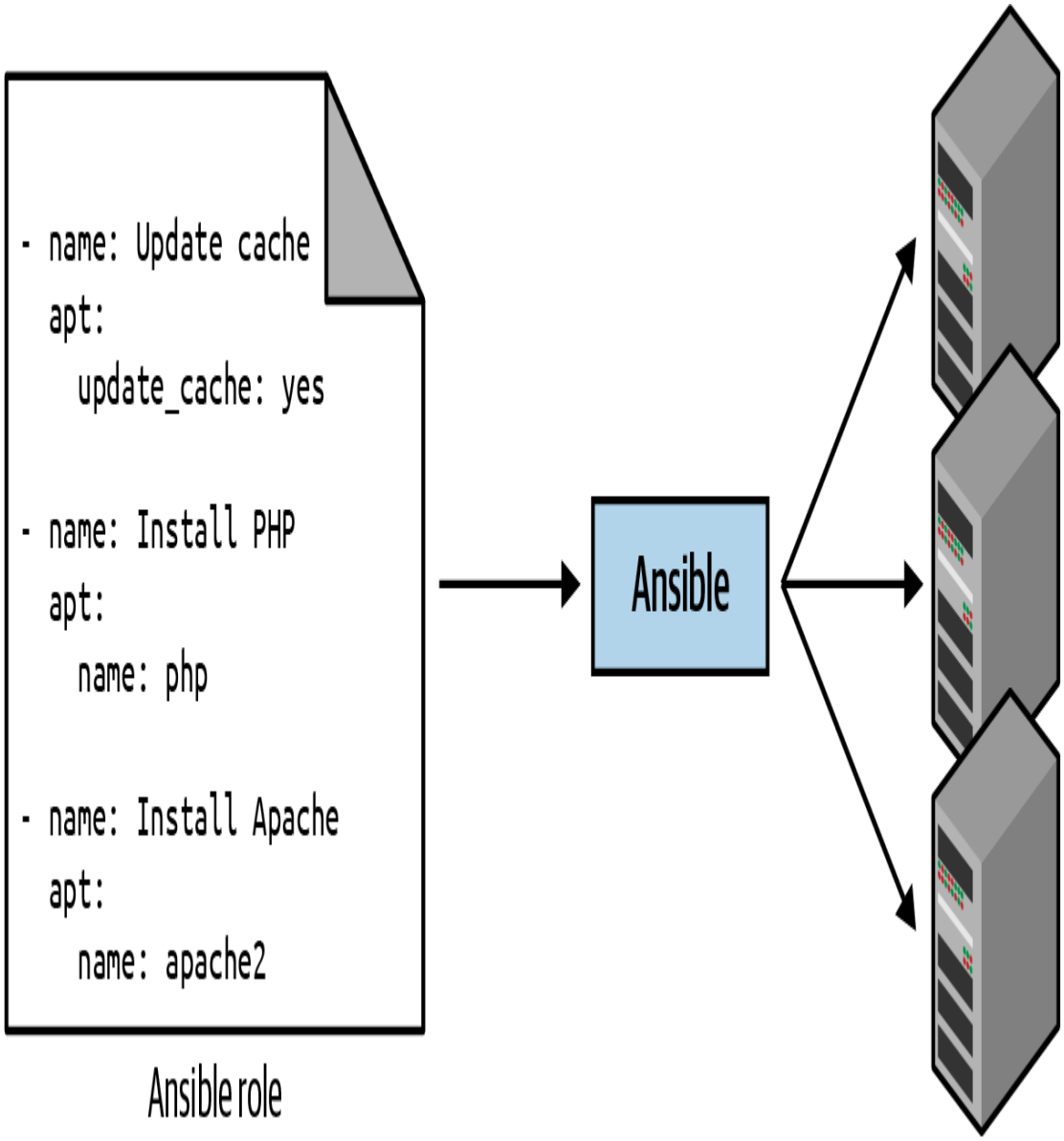

Figure 1-2. A configuration management tool like Ansible can execute your code across a large number of servers.

For example, to apply the *web-server.yml* role to five servers, you first create a file called *hosts* that contains the IP addresses of those servers:

```
[webservers]
11.11.11.11
11.11.11.12
11.11.11.13
```