Recent years have seen the rise of disciplines like continuous integration, test driven development, build/deployment automation, and more. All of these had a purpose to automate as many parts as possible of the lifecycle of a software product. However, the main focus often is the software itself, and the infrastructure on which the software runs is still quite often a “work of art.”

In a classic sense, infrastructure summarizes items such as operating systems, servers, switches, and routers (see Chapter 3). According to other definitions, infrastructure comprises all of the environments of an organization together with supporting services, such as firewalls and monitoring services (see Chapter 8). The use of the term *infrastructure as code* is widespread these days. In this context, infrastructure is often thought to include every part of the solution that is not the developed software application itself (although even the software is included occasionally). In that sense, infrastructure is meant to include the middleware (such as web servers with configuration files, software packages as part of the operating system, crontabs, technical users, and so on).

Infrastructure is set up and changed over time, before the software even goes into production. If you’re lucky, infrastructure is documented well, but often enough it would not be an easy task to rebuild your infrastructure from scratch if the need would arise.

“Perl was designed as a programming language for automating system administration. It didn’t take long for leading-edge sysadmins to realize that handcrafted configurations and non-reproducible incantations were a bad way to run their shops.”

---

1. [http://twitter.com/#!/devops_borat/status/41587168870797312](http://twitter.com/#!/devops_borat/status/41587168870797312)

Infrastructure as code has lately become popular to emphasize the need to handle the set up of your infrastructure in the same way you would handle development of your code: pick the right language or tool to do the job and start developing a solution that suits your needs, making it an executable specification that can be applied to target systems efficiently and repeatedly.

This chapter introduces the tools Vagrant, a tool for managing virtualized development environments, and Puppet, a tool for managing infrastructure items that are often also called configurations. Afterward, we’ll explore a real-world use that is based on how the development of the build server Jenkins is streamlined by Vagrant and Puppet.

Starting with Infrastructure as Code

Setup and maintenance of infrastructure were automated even before the rise of Agile software development and the DevOps movement. But there have often been handcrafted, scripted solutions, barely readable by someone other than the original author. In recent years, a few tools in the field of configuration management started to gain popularity to address these challenges. These tools help developers and operations to work together and enable more transparency on the infrastructure level. After all, in today’s world of ever more complex and distributed IT systems, there’s an increasing need for developers to know about operational things and vice versa. The infrastructure as code paradigm and its related tools can help to achieve this goal.

Before we get into more detail, there are two key questions to be addressed:

- Why should I adopt infrastructure as code?
- How should I do it? (which tools can I use?)

To answer these questions, let’s first think about the process involved in infrastructure setup when a typical web application gets developed.

Traditional Infrastructure Handling

In the first phase, as the architecture is not yet fixed, developers will try components, eventually ending up in a first draft of the setup. Now, each developer has set up his or her local development environment, with all the components running on the machine. At this point there might not even be a shared environment on which the software gets deployed continuously. That would be the next step. Depending on the organization’s structure, this might already be the borderline where the developers are not allowed full access to the target machines anymore and they have to provide the operations team with some sort of assistance on how to get things up. The same might happen again for QA, staging, and production environments. And as reality goes, at least the configuration of the components might change continuously along the way, if not new components being added to the infrastructure.

---