O'REILLY®

Learning Puppet 4

A GUIDE TO CONFIGURATION MANAGEMENT AND AUTOMATION

Jo Rhett

Learning Puppet 4

A Guide to Configuration Management and Automation

Jo Rhett

O'REILLY®

Beijing • Boston • Farnham • Sebastopol • Tokyo

Learning Puppet 4

by Jo Rhett

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (http://safaribooksonline.com). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

- Editor: Brian Anderson
- Production Editor: Kristen Brown
- Copyeditor: Rachel Monaghan
- Proofreader: Jasmine Kwityn
- Indexer: Judy McConville
- Interior Designer: David Futato
- Cover Designer: Karen Montgomery
- Illustrator: Rebecca Demarest
- April 2016: First Edition

Revision History for the First Edition

- 2016-03-22: First Release

See http://oreilly.com/catalog/errata.csp?isbn=9781491907665 for release details.

978-1-491-90766-5

[LSI]

Foreword

I first met Jo Rhett online in the early 2000s. We were both active on the CFengine mailing lists, trying to administer production systems with methodologies that would come to be known as *infrastructure as code*. At the time, the concept of repeatable, convergent automation tools was fairly edgy, and the community that arose around the space also had its sharp edges. I seem to recall standing up at a LISA config management birds-of-a-feather session circa 2003, pointing at Mark Burgess, CFengine's author, and shouting, "You ruined my life! But you *saved* my life!"

Luke Kanies was also a CFengine user at the time, and was consistently exhorting the community to introspect its shortcomings and evolve to the next level. His edginess led to some pretty epic—thoug *mostly* collegial—debates, borne out in papers such as Steve Traugott's "Turing Equivalence in Systems Administration" and Luke's "ISConf: Theory, Practice, and Beyond."

Around 2005, Luke began writing a new tool in Ruby that was intended to address the problems with usability and flexibility he was running into with the rest of the ecosystem. Originally called Blink, the thing we know today as Puppet began its infancy as a CFengine "module" (actually a plugin, thus perpetuating an unfortunate relationship with English names that Puppet continues to this day), and J and I, along with the rest of the nascent configuration management community, followed its development with interest.

Fast-forward several years through a few iterations of Moore's law, and witness the resultant explosion in processing power, distributed systems complexity, and their attendant burden on operations folk. Puppet had matured to become the dominant *lingua franca* of infrastructure; the "too wars" of the mid-2000s had achieved détente and our focus turned to culture, process, and the wider problems of tying infrastructure operations to business value: in a word, DevOps.

I'd done tens of thousands of infrastructure buildouts on Puppet and eventually came to work at Puppet Labs, the company Luke formed around supporting and productizing Puppet. Jo was consultin at large companies around Silicon Valley, pushing Puppet to its limits and feeding bug fixes and feature requests into the community. As Puppet's new product manager, this relationship was a little different (and frankly, sometimes far less comfortable, on my end!) than working as peers on the sam project, but over the past several years it's turned out to be hugely positive.

The thing I appreciate most about Jo, which I think shines through in the book you're about to read, i his sincere desire to help others. This is a core principle of the DevOps movement, usually stated as "sharing" or "empathy," and Jo's embodied it since the early days of CFengine. He consistently advocates for the "tough right thing" for the users, and while he will describe a range of possibilities with deep technical acumen and clear-eyed candor, in the end he'll steer us, the readers, in the direction that's going to work out the best in the long term.

Puppet's an amazing tool, but as the saying goes: "With great power comes great responsibility." Jo' depth of experience and empathy for other operations engineers make him the right person to show u how to use Puppet's power responsibly. This book is the result, and I'm deeply grateful to him for writing it.

Eric Sorenson
Technical Product Manager

Preface

Twenty years ago, it was common for a single server to provide services for hundreds of users. A system administrator was often responsible for as few as 10 servers. Most people used only one computer.

In 2015, it is common for a normal, everyday person to own and utilize more than five highly advanced and capable computers. Think about the devices you use on a daily basis: every one of them —the phone on your desk, the cell phone on your hip, the tablet you read from, your laptop, and even the car you drive—is thousands of times more powerful and capable than the large, room-sized server used a few generations ago.

We live today in the midst of an information revolution. Systems capable of powerful computation that once required server rooms to contain are now able to be held in your hands. Explosive growth in the adoption and capabilities of modern technology has created a world full of computers. More and more devices every day contain powerful small computers that participate in the Internet of Things.

When I started my career, it was difficult to convince managers that every worker needed his or her own computer. Today, the workers are outnumbered by computers almost 20:1, and in certain industries by as much as 100:1. Advanced computing capability combined with cheap memory have revolutionized what businesses can accomplish with data. Even small teams of people utilize and depend upon thousands of computers. Every one of these devices needs to be managed. It’s simply not possible to do it all by hand.

For this, we use Puppet.

Who This Book Is For

This book is primarily aimed at system administrators and operations or DevOps engineers. If you ar responsible for development or production nodes, this book will provide you with immediately usefu tools to make your job easier than ever before. If you run a high-uptime production environment, you're going to learn how Puppet can enforce standards throughout the implementation. Soon you'll wonder how you ever got along without it.

No matter what you call yourself, if you feel that you spend too much time managing computers, the this book is for you. You'd like to get it done faster so you can focus on something else. You'd like t do it more consistently, so that you don't have to chase down one-off problems in your reports. Or you've got some new demands that you're looking for a way to solve. If any of these statements fit, Puppet will be one of the best tools in your toolbox.

What to Expect from Me

This book will not be a heavy tome filled with reference material irrelevant to the day-to-day system administrator—exactly the opposite. Throughout this book we will never stray from one simple goal: using Puppet to manage real-life administration concerns.

This book will never tell you to run a script and not tell you what it does, or why. I hate trying to determine what an installation script did, and I won't do this to you. In this book you will build up th entire installation by hand. Every step you take will be useful to you in a production deployment. You'll know where every configuration file lives. You'll learn every essential configuration paramet and what it means.

By the time you have finished this book, you'll know how Puppet works inside and out. You will hav the tools and knowledge to deploy Puppet seamlessly throughout your environment.

What You Will Need

You may use any modern Linux, Mac, or Windows system and successfully follow the hands-on tutorials in this book.

While there are some web dashboards for Puppet, the process of configuring and running Puppet will be performed through the command line. We will help you install all necessary software.

A beginner to system administration can follow every tutorial in this book. Any experience with scripts, coding, or configuration management will enhance what you can get out of this book, but is not necessary. It is entirely possible to deploy Puppet to manage complex environments without writing a single line of code.

Part II documents how to build custom modules for Puppet. You will create modules using the Puppe configuration language that will be taught in this book. When you've become an expert in building Puppet modules, you may want to add new extensions to the Puppet configuration language. Some extensions are currently only supported in the Ruby language:

Faces

Add-on subcommands for `puppet`

Providers
: Implementation methods for resource types

Syntax checkers
: A tool for validating input

Types
: New resources not built into Puppet

Reference materials such as Michael Fitzgerald's *Learning Ruby* can be helpful when creating extensions for a custom Puppet module.

What You'll Find in This Book

The Introduction provides an overview of what Puppet does, how it works, and why you want to use i

Part I will get you up and running with a working Puppet installation. You will learn how to write declarative Puppet policies to produce consistency in your systems. This part will also cover the changes to the language in Puppet 4.

Part II will introduce you to Puppet modules, the building blocks used for Puppet policies. You will learn how to find and evaluate Puppet modules. You'll learn how to distinguish *Puppet Supported* an *Puppet Approved* modules. You'll learn tips for managing configuration data in Hiera. Finally, you'll learn how to build, test, and publish your own Puppet modules.

Part III will help you install the new Puppet Server and the deprecated but stable Puppet master. You'll learn how to centralize the certificate authority, or use a third-party provider. You will configure an *external node classifier* (*ENC*). You'll find advice and gain experience on how to scale Puppet servers for high availability and performance.

Part IV will review dashboards and orchestration tools that supplement and complement Puppet. The web dashboards provide a way to view the node status and history of changes made by Puppet. The orchestration tools enable you to interact instantly with Puppet nodes for massively parallel orchestration events.

Every step of the way you'll perform hands-on installation and configuration of every component. There are no magic scripts, no do-it-all installers. You'll see how easy it is to deploy Puppet from scratch, and experience firsthand the power of the tools it provides. You'll finish this book with everything you need to build out a production service.

Throughout this book you'll find commentary and practical advice that is based on years of experien deploying, scaling, and tuning Puppet environments. You will find advice about managing small shops, large commercial enterprises, and globally distributed teams. You'll also learn several ways t scale Puppet to thousands of nodes.

How to Use This Book

This book provides explicit instructions for configuring and using Puppet from the command line without the use of external tools beyond your favorite text editor.

The book will help you create Puppet *manifests* and Puppet *modules* that utilize every feature of Puppet. You will create configuration policies to handle your specific needs from the examples in thi book.

Everything you learn in this book can be done entirely from your laptop or workstation without any impact on production environments. However, it will teach you everything necessary to deploy Pupp in real environments, and will include numerous tips and recommendations for production use.

IPv6 Ready

Every example with IP addresses will include both IPv4 and IPv6 statements. If you're only using on of these protocols, you can ignore the other. Puppet will happily use any combination of them. Specific advice for managing Puppet in dual-stack IPv6 environments can be found in multiple parts of the book.

SSL is now TLS

You are likely familiar with the term *SSL* when referring to transport layer security and encryption. You're also likely aware that SSL v3 was renamed TLS 1.0 when it became an IETF standard. At this time, all versions of SSL and up to TLS 1.1 are subject to known exploits. Rather than constantly refe to both terms (SSL/TLS) throughout this book, I will refer to it only by the new name (TLS). This is technically more accurate, as Puppet 4 requires TLS 1.2 and will not accept SSL connections.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`

Shows commands or other text that should be typed literally by the user.

`Constant width italic`

Shows text that should be replaced with user-supplied values or by values determined by context.

TIP

This element signifies a tip or suggestion.

NOTE

This element signifies a general note.

WARNING

This element indicates a warning or caution.

Safari® Books Online

NOTE

Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of plans and pricing for enterprise, government, education, and individuals.

Members have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and hundreds more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

- O'Reilly Media, Inc.
- 1005 Gravenstein Highway North
- Sebastopol, CA 95472
- 800-998-9938 (in the United States or Canada)
- 707-829-0515 (international or local)
- 707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at *http://bit.ly/learningPuppet4*.

To comment or ask technical questions about this book, send email to *bookquestions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at http://www.oreilly.com.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

I owe a deep debt of gratitude to Nova, Kevin, Andy, Lance, and far too many other friends whom I have neglected while adding “just one more essential thing” (more than a hundred times) to this book I lack the words to express how thankful I am that you are in my life, and have apparently forgiven m for being a ghost over the last year.

Growing and improving happens only by surrounding yourself with smart people. The following individuals have provided incalculable feedback and insight that has influenced my learning process, and deserve both accolades for their technical efforts and my sincere appreciation for their advice:

- Corey Quinn
- R.I. Pienaar
- Chris Barbour
- William Jiminez
- Rob Nelson

There are far too many Puppet Labs employees to list who have accepted, rejected, and negotiated my suggestions, feedback, and patches to Puppet and related tools. Thank you for your assistance over th years. We all appreciate your efforts far more than we remember to share with you.

I owe a drink and many thanks to many people who provided input and feedback on the book during the writing process, including but definitely not limited to the technical reviewers:

- Eric Sorenson, Puppet Labs
- Anna Kennedy, Redpill-Linpro
- Nick Fagerlund, Puppet Labs

If you find that the examples in this book work well for you, it’s likely due to the helpful feedback provided by numerous readers who posted errata on the book page or comments on the Safari book that helped me address concerns I would have missed. Their insights have been invaluable and deeply appreciated.

And finally, I’d like to thank my O’Reilly editor, Brian Anderson, who gave me excellent guidance o the book and was a pleasure to work with. I’m likewise deeply indebted to my patient and helpful production and copy editors, Kristen and Rachel, without whom my jumbled pile of letters wouldn’t

make any sense at all.

All of us who use Puppet today owe significant gratitude to Luke Kaines, who conceived of Puppet and continues to direct its growth in Puppet Labs. His vision and foresight made all of this possible.

Introduction

This book will teach you how to install and use Puppet for managing computer systems. It will introduce you to how Puppet works, and how it provides value. To better understand Puppet and learn the basics, you'll set up a testing environment, which will evolve as your Puppet knowledge grows. You'll learn how to declare and evaluate configuration policy for hundreds of nodes.

This book covers modern best practices for Puppet. You'll find tips throughout the book labeled Best Practice.

You'll learn how to update Puppet 2 or Puppet 3 puppet code for the increased features and improved parser of Puppet 4. You'll learn how to deploy the new Puppet Server. You'll have a clear strategy for upgrading older servers to Puppet 4 standards. You'll learn how to run Puppet services over IPv6 protocol.

Most important of all, this book will cover how to scale your Puppet installation to handle thousands of nodes. You'll learn multiple strategies for handling diverse and heterogenous environments, and reasons why each of these approaches may or may not be appropriate for your needs.

What Is Puppet?

Puppet manages configuration data, including users, packages, processes, and services—in other words, any resource of the node you can define. Puppet can manage complex and distributed components to ensure service consistency and availability. In short, Puppet brings computer systems into compliance with a configuration policy.

Puppet can ensure configuration consistency across servers, clients, your router, and that computer on your hip. Puppet utilizes a flexible hierarchy of data sources combined with node-specific data to tune the policy appropriately for each node. Puppet can help you accomplish a variety of tasks. For example, you can use it to do any of the following:

- Deploy new systems with consistent configuration and data installed.
- Upgrade security and software packages across the enterprise.
- Roll out new features and capabilities to existing systems painlessly.
- Adjust system configurations to make new data sources available.
- Decrease the cost and effort involved in minor changes on hundreds of systems.
- Simplify the effort and personnel involved in software deployments.
- Automate buildup and teardown of replica systems to test proposed changes.
- Repurpose existing computer resources for a new use within minutes.
- Gather a rich data set of information about the infrastructure and computing resources.
- Provide a clear and reviewable change control mechanism.

Twenty years ago, people were impressed that I was responsible for 100 servers. At a job site last year I was responsible for over 17,000 servers. At my current job I'd have to go check somewhere to find

out, as we scale up and down dynamically based on load. These days, my Puppet code spins up more servers while I'm passed out asleep than I did in the first 10 years of my career. You can achieve this only by fully embracing what Puppet provides.

I was recently reminded of something I quipped to a CEO almost six years ago:

> *You can use Puppet to do more faster if you have ten nodes.*
> *You must use Puppet if you have ten hundred nodes.*

Puppet enables you to make a lot of changes both quickly and consistently. You don't have to write out every step, you only have to define how it should be. You are not required to write out the proces for evaluating and adjusting each platform. Instead, you utilize the Puppet configuration language to declare the final state of the computing resources. Thus, we describe Puppet as *declarative*.

Why Declarative

When analyzing hand-built automation systems, you'll invariably find commands such as the following:

```
$ echo "param: newvalue" >> configuration-file
```

This command appends a new parameter and value to a configuration file. This works properly the first time you run it. However, if the same operation is run again, the file has the value twice. This isn't a desirable effect in configuration management. To avoid this, you'd have to write code that checks the file for the configuration parameter and its current value, and then makes any necessary changes.

Language that describes the actions to perform is called *imperative*. It defines what to do, and how to do it. It must define every change that should be followed to achieve the desired configuration. It mu also deal with any differences in each platform or operating system.

When managing computer systems, you want the operations applied to be *idempotent*, where the operation achieves the same results every time it executes. This allows you to apply and reapply (or converge) the configuration policy and always achieve the desired state.

To achieve a configuration state no matter the existing conditions, the specification must avoid describing the actions required to reach the desired state. Instead, the specification should describe th desired state itself, and leave the evaluation and resolution up to the interpreter.

Language that declares the final state is called *declarative*. Declarative language is much easier to read, and less prone to breakage due to environment differences. Puppet was designed to achieve consistent and repeatable results. Every time Puppet evaluates the state of the node, it will bring the node to a state consistent with the specification.

How Puppet Works

Any node you control contains an application named `puppet agent`. The agent evaluates and applies

Puppet manifests, or files containing Puppet configuration language that declares the desired state of the node. The agent evaluates the state of each component described in a manifest, and determines whether or not any change is necessary. If the component needs to be changed, the agent makes the requested changes and logs the event.

If Puppet is configured to utilize a centralized Puppet server, Puppet will send the node's data to the server, and receive back a *catalog* containing only the node's specific policy to enforce.

Now you might be thinking to yourself, "What if I only want the command executed on a subset of nodes?" Puppet provides many different ways to classify and categorize nodes to limit which resources should be applied to which nodes. You can use node facts such as hostname, operating system, node type, Puppet version, and many others. Best of all, new criteria custom to your environment can be easily created.

The Puppet agent evaluates the state of only one node. In this model, you can have agents on tens, hundreds, or thousands of nodes evaluating their catalogs and applying changes on their nodes at exactly the same time. The localized state machine ensures a scalable and fast parallel execution environment.

Why Use Puppet

As we have discussed, Puppet provides a well-designed infrastructure for managing the state of many nodes simultaneously. Here are a few reasons to use it:

- Puppet utilizes a node's local data to customize the policy for each specific node: hundreds of values specific to the node—including hostname, operating system, memory, networking configuration, and many node-specific details—are used to tune the policy appropriately.
- Puppet agents can handle OS-specific differences, allowing you to write a single manifest that wil be applied by OS-specific *providers* on each node.
- Puppet agents can be invoked with specific tags, allowing a filtered run that only performs operations that match those tags during a given invocation.
- Puppet uses a decentralized approach where each node evaluates and converges its own Puppet catalog separately. No node is waiting for another node to complete.
- Puppet agents report back success, failure, and convergence results for each resource, and the enti run.
- Orchestration systems such as the Marionette Collective (MCollective) can invoke and control the Puppet agent for instantaneous large-scale changes.

In Part I, you will learn how to write simple declarative language that will make changes only when necessary.

In Part II, you will create a module that uses Puppet to install and configure the Puppet agent. This kind of recursion is not only possible but common.

In Part III, you will learn how to use Puppet masters and Puppet Server to offload and centralize catalog building, report processing, and backup of changed files.

In Part IV, you will use MCollective to orchestrate immediate changes with widespread Puppet agent

Puppet provides a flexible framework for policy enforcement that can be customized for any environment. After reading this book and using Puppet for a while, you'll be able to tune your environment to exactly your needs. Puppet's declarative language not only allows but encourages creativity.

Is Puppet DevOps?

While Puppet is a tool used by many DevOps teams, using Puppet will not by itself give you all the benefits of adopting DevOps practices within your team.

In practice, Puppet is used by many classic operations teams who handle all change through tradition planning and approval processes. It provides them with many benefits, most especially a readable, somewhat self-documenting description of what is deployed on any system. This provides tremendou value to operations where change management control and auditing are primary factors.

On the other hand, the ability to manage rapid change across many systems that Puppet provides has also cracked open the door to DevOps for many operations teams. Classic operations teams were ofte inflexible because they were responsible for the previously difficult task of tracking and managing change. Puppet makes it possible to not only track and manage change, but to implement locally customized change quickly and seamlessly across thousands of nodes. This makes it possible for operations teams to embrace practices that are flexible for changing business needs.

You don't have to be working with developers to utilize DevOps practices. This is a common misconception of DevOps. The developer in DevOps is not a different person or team, it is you! Ther are many teams which utilize DevOps practices that don't support developers; rather, they manage systems that support a completely different industry. You are participating in DevOps when you utilize Agile development processes to develop code that implements operations and infrastructure designs.

WHAT IS DEVOPS?

I feel that it is essential to refer to the Agile Manifesto, because it is the base on which DevOps practices emerged. It may help you to mentally replace *software* with *services* when reading the manifesto:

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right,
we value the items on the left more.

The DevOps term was first used to refer to the application of Agile software development practices to systems administration. The original DevOps discussion group was named "Agile System Administration." This led to a conference in Belgium named "DevOpsDays," which was the first appearance of the name. After the conference, the hashtag *#devops* was used to continue the discussion on Twitter.

While DevOps teams customize their Agile practices for needs specific to operations processes, embracing DevOps practices provides many advantages to Agile development teams. Agile Scrum-based *continuous integration* (*CI*) can also include *continuous delivery* (*CD*) when the operations teams utilize DevOps practices.

Perhaps the biggest source of confusion comes when people try to compare using Puppet (a tool) to implement DevOps practices, and the idea of "valuing individuals and interactions over processes an tools." It is easiest to explain this by first outlining the reasons that operations teams were historicall perceived as inflexible. The tools available for managing software within the enterprise used to be shockingly limited. Many times a very small change, or a customization for one group, would require throwing away the software management tool and embracing another. That's a tremendous amount o work for an operations team.

When using Puppet, if an individual can make a case for the value of a change (interaction), then rolling out a local customization usually involves only a small refactoring of the code. Applying changes becomes easy, and thus avoids a conflict between valuable requests and the limitations of th tools available. Discussion of the merits of the change has higher value than processes used to protec operations teams from unmanageable change.

No tool or set of tools, product, or job title will give an operations team all the benefits of utilizing DevOps practices. I have seen far too many teams with all of the keywords, all of the titles, and none of the philosophy. It's not a product, it's an evolving practice. You don't get the benefits without changing how you think.

I highly recommend making the effort to fully understand both Agile processes and DevOps practice and methodologies. Don't skimp. Someone on your team who is good at promoting and guiding other should be fully trained on Agile processes. Get the people responsible for creating change out to conferences where they can learn from others' experiences, such as DevOps Days, PuppetConf, and O'Reilly's Velocity conference.

That's not an obligatory push of the publisher's conference. Many people consider John Allspaw's "10+ Deploys Per Day: Dev and Ops Cooperation," which was presented at Velocity 2009, to be a founding cornerstone of the DevOps movement. Velocity was the first large conference to add DevO

items to its agenda, and DevOps practices are a major focus of the conference today.

Puppet is a great tool, and you'll find that it's even more valuable when used with DevOps practices. While this is not a book about DevOps, it will present many useful tools and strategies for practicing DevOps in your organization.

Time to Get Started

As we proceed, this book will show you how Puppet can help you do more, and do it faster and more consistently than ever before. You'll learn how to extend Puppet to meet your specific needs:

- You'll install Puppet and get it working seamlessly to control files, packages, services, and the Puppet daemon.
- You'll discover an active community of Puppet developers who develop modules and other Puppe plugins on the Puppet Forge and GitHub.
- You'll build your own custom fact. You'll use this fact within your Puppet manifest to handle something unique to your environment.
- You'll build your own custom Puppet module. You'll learn how to test the module safely prior to deploying in your production environment.
- You'll learn how to package your module and upload it to a Puppet Forge.
- You'll learn how to configure Puppet Server, allowing you to centralize Puppet services within a campus or around the globe.
- You'll tour the ecosystem of components that utilize, extend, and enhance Puppet within your environment.

By the time you finish this book, you will understand not just how powerful Puppet is, but also exactl how it works. You'll have the knowledge and understanding to debug problems within any part of the infrastructure. You'll know what to tune as your deployment grows. You'll have a resource to use for further testing as your knowledge and experience expands.

It's time to get declarative.

Part I. Controlling with Puppet Apply

In this part, you'll learn about the Puppet configuration language and how to think in a declarative manner. You'll set up a testing environment that you can use to learn Puppet while reading this book You'll be able to continue to use this setup to develop and test your Puppet code long after you have finished this book.

To start, you will install Puppet and create your first Puppet manifests. You'll learn how to utilize resources, how to associate them, and how to order and limit the changes upon them.

When you finish this part of the book, you'll have a solid understanding of the Puppet configuration language. As you make your way through the chapters in this part, you will write and test your own Puppet manifests. This part establishes a foundation of modern best practices from which you can explore Puppet's features and capabilities.

Chapter 1. Thinking Declarative

If you have experience writing shell, Ruby, Python, or Perl scripts that make changes to a system, you've very likely been performing *imperative programming*. Imperative programming issues commands that change a target's state, much as the imperative grammatical mood in natural languag expresses commands for people to act on.

You may be using *procedural programming* standards, where state changes are handled within procedures or subroutines to avoid duplication. This is a step toward declarative programming, but th main program still tends to define each operation, each procedure to be executed, and the order in which to execute them in an imperative manner.

While it can be useful to have a background in procedural programming, a common mistake is to attempt to use Puppet to make changes in an imperative fashion. The very best thing you can do is forget everything you know about imperative or procedural programming.

If you are new to programming, don't feel intimidated. People without a background in imperative or procedural programming can often learn good Puppet practices faster.

Writing good Puppet manifests requires declarative programming. When it comes to maintaining configuration on systems, you'll find declarative programming to be easier to create, easier to read, and easier to maintain. Let's show you why.

Handling Change

The reason that you need to cast aside imperative programming is to handle change better.

When you write code that performs a sequence of operations, that sequence will make the desired change the first time it is run. If you run the same code the second time in a row, the same operations will either fail or create a different state than desired. Here's an example:

```
$ sudo useradd -u 1001 -g 1001 -c "Joe User" -m joe
$ sudo useradd -u 1001 -g 1000 -c "Joe User" -m joe
useradd: user 'joe' already exists
```

So then you need to change the code to handle that situation:

```
# bash excerpt
getent passwd $USERNAME > /dev/null 2> /dev/null
if [ $? -ne 0 ]; then
    useradd -u $UID -g $GID -c "$COMMENT" -s $SHELL -m $USERNAME
else
    usermod -u $UID -g $GID -c "$COMMENT" -s $SHELL -m $USERNAME
fi
```

OK, that's six lines of code and all we've done is ensure that the username isn't already in use. What

if we need to check to ensure the UID is unique, the GID is valid, and that the password expiration is set? You can see that this will be a very long script even before we adjust it to ensure it works properly on multiple operating systems.

This is why we say that imperative programming doesn't handle change very well. It takes a lot of code to cover every situation you need to test.

Using Idempotence

When managing computer systems, you want the operations applied to be *idempotent*, where the operation achieves the same results every time it executes. Idempotence allows you to apply and reapply (or converge) a configuration manifest and always achieve the desired state.

In order for imperative code to be idempotent, it needs to have instructions for how to compare, evaluate, and apply not just every resource, but also each attribute of the resource. As you saw in the previous section, even the simplest of operations will quickly become ponderous and difficult to maintain.

WHAT IS AN IDEMPOTENT OPERATION?

In mathematics and computer science, idempotent operations are those that can be applied multiple times without changing the result beyond the initial application. The word literally means "[the quality of] having the same power," from the Latin roots *idem* + *potent* "same" + "power."[1] Here are some examples of idempotent and non-idempotent math and code:

`any number^1`	**Idempotent**	A number to the power of 1 is the same
`value = value * 2`	**Non-idempotent**	Will double every time
`value = value * 2 / 2`	**Idempotent**	Remains the same value
`echo "Good!" >> /some/file`	**Non-idempotent**	File will keep growing
`echo "Good!" > /some/file`	**Idempotent**	File will always have the same content

The simplistic final example avoids having to compare the state of the item by simply overwriting it every time. This only works in a limited set of situations. Most changes require evaluation to determine what changes are necessary.

Declaring Final State

As we mentioned in Introduction, for a configuration state to be achieved no matter the conditions, th configuration language must avoid describing the actions required to reach the desired state. Instead, the configuration language should describe the desired state itself, and leave the actions up to the interpreter. Language that declares the final state is called *declarative*.

Rather than writing extensive imperative code to handle every situation, it is much simpler to declare what you want the final state to be. In other words, instead of including dozens of lines of comparison the code reflects only the desired final state of the resource (a user account, in this example). Here w will introduce you to your first bit of Puppet configuration language, a resource declaration for the same user we created earlier:

```
user { 'joe':
  ensure     => present,
  uid        => '1001',
  gid        => '1000',
  comment    => 'Joe User',
  managehome => true,
}
```

As you can see, the code is not much more than a simple text explanation of the desired state. A user named *Joe User* should be *present*, a home directory for the user should be created, and so on. It is very clear, very easy to read. Exactly how the user should be created is not within the code, nor are instructions for handling different operating systems.

Declarative language is much easier to read, and less prone to breakage due to environment differences. Puppet was designed to achieve consistent and repeatable results. You describe what the final state of the resource should be, and Puppet will evaluate the resource and apply any necessary changes to reach that state.

Reviewing Declarative Programming

Conventional programming languages create change by listing exact operations that should be performed. Code that defines each state change and the order of changes is known as *imperative programming*.

Good Puppet manifests are written with declarative programming. Instead of defining exactly how to make changes, in which you must write code to test and compare the system state before making that change, you instead declare how it should be. It is up to the Puppet agent to evaluate the current state and apply the necessary changes.

As this chapter has demonstrated, declarative programming is easier to create, easier to read, and easier to maintain.

---

[1] First seen in George Boole's book *The Mathematical Analysis of Logic*, originally published in 1847.

Chapter 2. Creating a Learning Environmen

In this chapter, we will create a virtualized environment suitable for learning and testing Puppet. We will utilize Vagrant and VirtualBox to set up this environment on your desktop or laptop. You can keep using this environment long after you have finished this book, or rebuild it quickly any time you'd like to test something new.

If you are an experienced developer or operations engineer, you are welcome to use a testing environment of your own choice. Anything that can host multiple Linux nodes will work. Puppet's needs are minimal. Any of the following would be suitable for use as a Puppet test lab:

- A spare system that you can install Linux on
- An Amazon Web Services (AWS) Free Tier instance
- An OpenStack DevStack development instance
- A VMware Free vSphere ESXi solo instance
- A Vagrant development environment on your personal computer

You can build your own test lab using one of the preceding solutions, or you can use an existing test lab you maintain. In all cases, I recommend using an OS compatible with RedHat Enterprise Linux 7 for learning purposes. The CentOS platform is freely available, and fully supported by both Red Hat and Puppet Labs. This will allow you to breeze through the learning exercises without distractions.

We recommend and are going to use Vagrant for the remainder of this book, for the following reason

- It is easier for you to set up and get started quickly.
- You can more easily carry it with you, and restart it at any time.
- The Vagrant setup includes the Puppet manifests shown in this book.
- You can always build one of the other environments later for a comparison point.

If you plan to use your own testing environment, skip ahead to "Initializing Non-Vagrant Systems".

Installing Vagrant

If you are going to follow the recommendation to use Vagrant, let's get started installing it. You'll need to download two packages.

Go to VirtualBox Downloads and download the appropriate platform package for your system.

Next, go to Vagrant Downloads and download the appropriate platform package for your system.

Install these packages according to the instructions for your operating system. I've included detailed instructions for Windows and Mac users in the following subsections. If you're running Vagrant on Linux or another platform, I'm going to assume you are expert enough to handle this installation yourself.

Installing Vagrant on Mac

First, you should run the VirtualBox installer. Open the VirtualBox DMG image file you downloaded and click on the *Virtualbox.pkg* installer, as shown in Figure 2-1.

Accept the license and the installer will complete the installation.

Next, you should run the Vagrant installer. Open the Vagrant DMG image file you downloaded and click on the *Vagrant.pkg* installer, as shown in Figure 2-2.

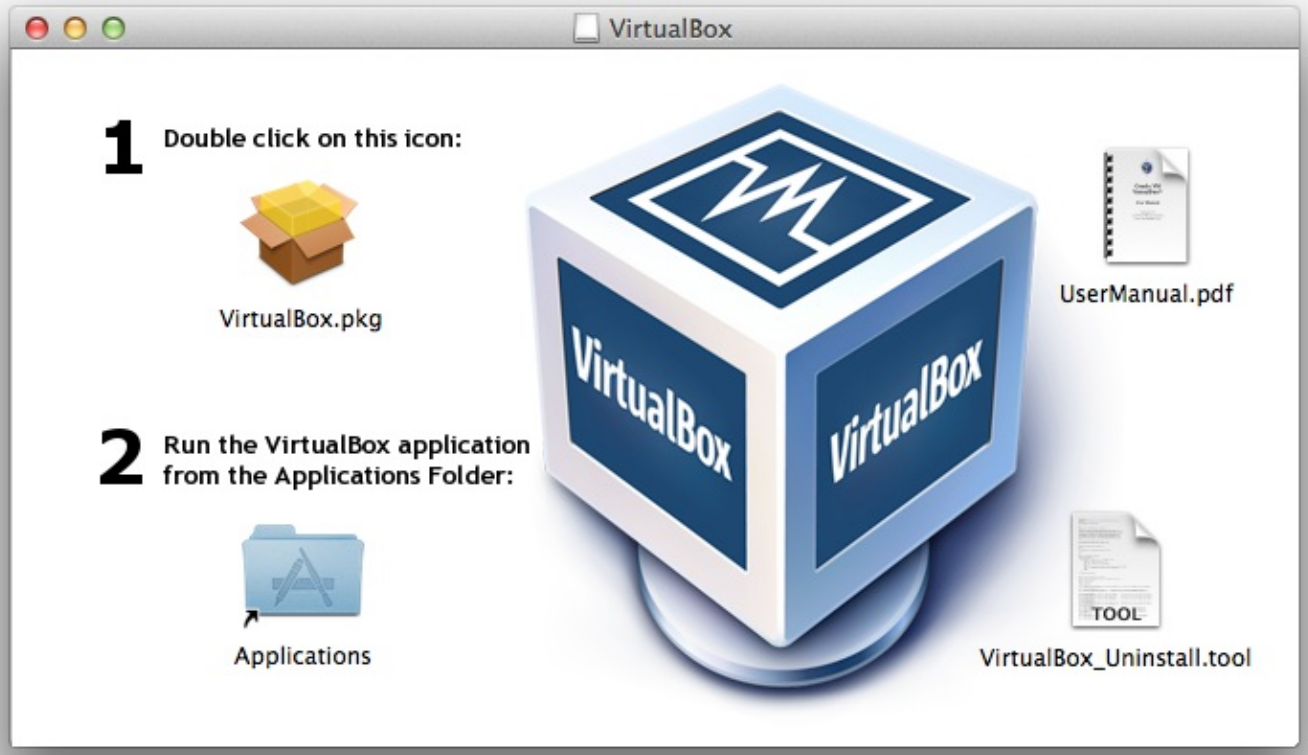

Figure 2-1. VirtualBox package installer for Mac