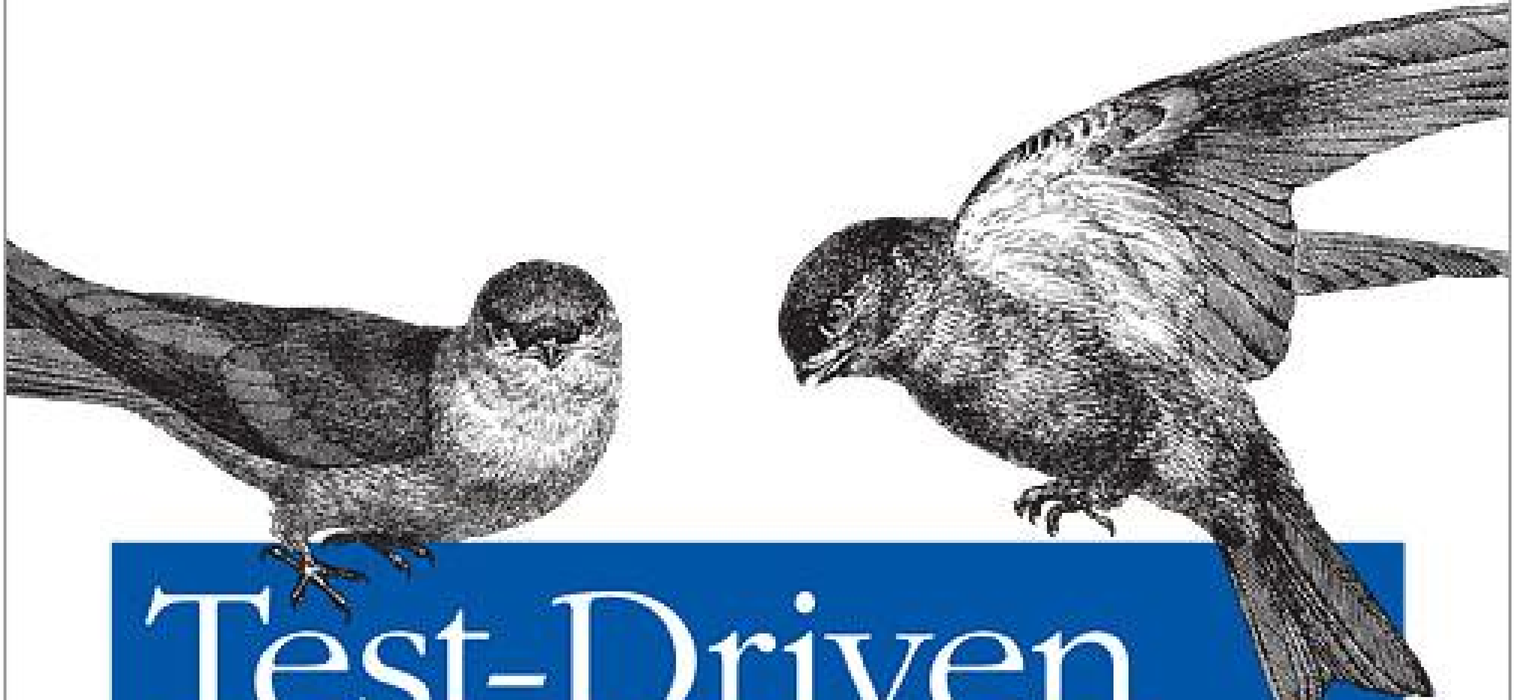
Bring Behavior-Driven Development to Infrastructure as Code



Test-Driven Infrastructure with Chef

O'REILLY®

Stephen Nelson-Smith

Table of Contents

Prepared for it@online-buddies.com, Joe Marshall
Copyright © 2011 Atalanta Systems LTD. This download file is made available for personal use only and is subject to the Terms of Service. Any other use requires prior written consent from the copyright owner. Unauthorized use, reproduction and/or distribution are strictly prohibited and violate applicable laws. All rights reserved.

CHAPTER 1

Infrastructure As Code

"When deploying and administering large infrastructures, it is still common to think in terms of individual machines rather than view an entire infrastructure as a combined whole. This standard practice creates many problems, including labor-intensive administration, high cost of ownership, and limited generally available knowledge or code usable for administering large infrastructures."

—Steve Traugott and Joel Huddleston, TerraLuna LLC

"In today's computer industry, we still typically install and maintain computers the way the automotive industry built cars in the early 1900s. An individual craftsman manually manipulates a machine into being, and manually maintains it afterwards.

The automotive industry discovered first mass production, then mass customisation using standard tooling. The systems administration industry has a long way to go, but is getting there."

—Steve Traugott and Joel Huddleston, TerraLuna LLC

These two statements came from the prophetic *infrastructures.org* website at the very start of the last decade. Nearly ten years later, a whole world of exciting developments have taken place, which have sparked a revolution, and given birth to a radical new approach to the process of designing, building and maintaining the underlying IT systems that make web operations possible. At the heart of that revolution is a mentality and a tool set that treats *Infrastructure as Code*.

This book is written from the standpoint that this approach to the designing, building, and running of Internet infrastructures is fundamentally correct. Consequently, we'll spend a little time exploring its origin, rationale, and principles before outlining the risks of the approach—risks which this book sets out to mitigate.

Chapter 1. Infrastructure As Code. Test-Driven Infrastructure with Chef, ISBN: 9781449309718
Prepared for it@online-buddies.com, Joe Marshall
Copyright © 2011 Atalanta Systems LTD. This download file is made available for personal use only and is subject to the Terms of Service. Any other use requires prior written consent from the copyright owner. Unauthorized use, reproduction and/or distribution are strictly prohibited and violate applicable laws. All rights reserved.

The Origins of Infrastructure as Code

Infrastructure as Code is an interesting phenomenon, particularly for anyone wanting to understand the evolution of ideas. It emerged over the last four or five years in response to the juxtaposition of two pieces of disruptive technology [*]—utility computing, and second-generation web frameworks.

The ready availability of effectively infinite compute power, at the touch of a button, combined with the emergence of a new generation of hugely productive web frameworks brought into existence a new world of scaling problems that had previously only been witnessed by large systems integrators. The key year was 2006, which saw the launch of Amazon Web Services' Elastic Compute Cloud (EC2), a few months after the release of version 1.0 of Ruby on Rails the previous Christmas. This convergence meant that anyone with an idea for a dynamic website—an idea which delivered functionality or simply amusement—to a rapidly growing Internet community, could go from a scribble on the back of a beermat to a household name in weeks.

Suddenly very small, developer-led companies found themselves facing issues that were previously tackled almost exclusively by large organizations with huge budgets, big teams, enterprise-class configuration management tools, and lots of time. The people responsible for these websites that had gotten huge almost overnight now had to answer questions such as how to scale read or write-heavy databases, how to add identical machines to a given layer in the architecture, and how to monitor and back up critical systems. Radically small teams needed to be able to manage infrastructures at scale, and to compete in the same space as big enterprises, but with none of the big enterprise systems.

It was out of this environment that a new breed of configuration management tools emerged.[†] Given the significance of 2006 in terms of the disruptive technologies we describe, it's no coincidence that in early 2006 Luke Kanies published an article on "Next-Generation Configuration Management"[‡] in *;login:* (the USENIX magazine), describing his Ruby-based system management tool, Puppet. Puppet provided a high level DSL with primitive programmability, but the development of Chef (a tool influenced by Puppet, and released in January 2009) brought the power of a 3GL programming language to system administration. Such tools equipped tiny teams and developers with the kind of automation and control that until then had only been available to the big players. Furthermore, being built on open source tools and released early to developer communities, these tools rapidly began to evolve according to demand, and

[*] Joseph L. Bower and Christensen, Clayton M, "Disruptive Technologies: Catching the Wave," *Harvard Business Review* January–February 1995.

[†] Although open source configuration management tools already existed, specifically CFengine, frustration with these existing tools contributed to the creation of Puppet.

[‡] *http://www.usenix.org/publications/login/2006-02/pdfs/kanies.pdf*

Licensed by Joe Marshall 2245195

Chapter 1. Infrastructure As Code. Test-Driven Infrastructure with Chef, ISBN: 9781449309718
Prepared for it@online-buddies.com, Joe Marshall
Copyright © 2011 Atalanta Systems LTD. This download file is made available for personal use only and is subject to the Terms of Service. Any other use requires prior written consent from the copyright owner. Unauthorized use, reproduction and/or distribution are strictly prohibited and violate applicable laws. All rights reserved.

arguably soon started to become even more powerful than their commercial counterparts.

Thus a new paradigm was introduced—the paradigm of *Infrastructure as Code*. The key concept is that it is possible to model our infrastructure as if it were code—to abstract, design, implement, and deploy the infrastructure upon which we run our web applications in the same way, and to work with this code using the same tools, as we would with any other modern software project. The code that models, builds, and manages the infrastructure is committed into source code management alongside the application code. The mindshift is in starting to think about our infrastructure as redeployable from a code base; a code base that we can work with using the kinds of software development methodologies that have developed over the last dozen or more years as the business of writing and delivering software has matured.

This approach brings with it a series of benefits that help the small, developer-led company to solve some of the scalability and management problems that accompany rapid and overwhelming commercial success:

Repeatability
: Because we're building systems in a high-level programming language, and committing our code, we start to become more confident that our systems are ordered and repeatable. With the same inputs, the same code should produce the same outputs. This means we can now be confident (and ensure on a regular basis) that what we believe will recreate our environment really *will* do that.

Automation
: We already have mature tools for deploying applications written in modern programming languages, and the very act of abstracting out infrastructures brings us the benefits of automation.

Agility
: The discipline of source code management and version control means we have the ability to roll forwards or backwards to a known state. In the event of a problem, we can go to the commit logs and identify what changed and who changed it. This brings down the average time to fix problems, and encourages root cause analysis.

Scalability
: Repeatability plus automation makes scalability much easier, especially when combined with the kind of rapid hardware provisioning that the cloud provides.

Reassurance
: The fact that the architecture, design, and implementation of our infrastructure is modeled in code means that we automatically have documentation. Any programmer can look at the source code and see at a glance how the systems work. This is a welcome change from the common scenario in which only a single sysadmin or architect holds the understanding of how the system hangs together. That is risky—this person is now able to hold the organization ransom, and should they leave or become ill, the company is endangered.

Chapter 1. Infrastructure As Code. Test-Driven Infrastructure with Chef, ISBN: 9781449309718
Prepared for it@online-buddies.com, Joe Marshall
Copyright © 2011 Atalanta Systems LTD. This download file is made available for personal use only and is subject to the Terms of Service. Any other use requires prior written consent from the copyright owner. Unauthorized use, reproduction and/or distribution are strictly prohibited and violate applicable laws. All rights reserved.

Disaster recovery
: In the event of a catastrophic event that wipes out the production systems, if your entire infrastructure has been broken down into modular components and described as code, recovery is as simple as provisioning new compute power, restoring from backup, and deploying the infrastructure and application code. What may have been a business-ending event in the old paradigm of custom-built, partially-automated infrastructure becomes a manageable few-hour outage, potentially delivering competitive value over those organizations suffering from the same external influences, but without the power and flexibility brought about by Infrastructure as Code.

Infrastructure as Code is a powerful concept and approach that promises to help repair the split-brain witnessed so frequently in organizations where developers and system administrators view each other as enemies, and don't work together. By giving operational responsibilities to developers, and liberating system administrators to start thinking at the higher levels of abstraction that are necessary if we're to succeed in building robust scaled architectures, we open up a new way of cooperating, a new way of working—which is fundamental to the emerging Devops movement.

The Principles of Infrastructure as Code

Having explored the origins and rationale for the project of managing Infrastructure as Code, we now turn to the core principles we should put into practice to make it happen.

Adam Jacob, co-founder of Opscode, and creator of Chef argues that, at a high level, there are two steps:

1. Break the infrastructure down into independent, reusable, network-accessible services.
2. Integrate these services in such a way as to produce the functionality your infrastructure requires.

Adam further identifies ten principles that describe what the characteristics of the reusable primitive components look like. His essay is essential reading[§], but I will summarize his principles here:

Modularity
: Our services should be small and simple—think at the level of the simplest free-standing, useful component.

Cooperation
: Our design should discourage overlap of services, and should encourage other people and services to use our service in a way which fosters continuous improvement of our design and implementation.

§ Infrastructure as Code in *Web Operations*, edited by John Allspaw and Jesse Robbins (O'Reilly)

Chapter 1. Infrastructure As Code. Test-Driven Infrastructure with Chef, ISBN: 9781449309718
Prepared for it@online-buddies.com, Joe Marshall
Copyright © 2011 Atalanta Systems LTD. This download file is made available for personal use only and is subject to the Terms of Service. Any other use requires prior written consent from the copyright owner. Unauthorized use, reproduction and/or distribution are strictly prohibited and violate applicable laws. All rights reserved.

Composability
: Our services should be like building blocks—we should be able to build complete, complex systems by integrating them.

Extensibility
: Our services should be easy to modify, enhance, and improve in response to new demands.

Flexibility
: We should build our services using tools that provide unlimited power to ensure we have the (theoretical) ability to solve even the most complicated of problems.

Repeatability
: Our services should produce the same results, in the same way, with the same inputs, every time.

Declaration
: We should specify our services in terms of *what* we want it to do, not *how* we want to do it.

Abstraction
: We should not worry about the details of the implementation, and think at the level of the component and its function.

Idempotence
: Our services should only be configured when required—action should only be taken once.

Convergence
: Our services should take responsibility for their own state being in line with policy; over time, the overall system will tend to correctness.

In practice, these principles should apply to every stage of the infrastructure development process—from provisioning (cloud-based providers with a published API are a good example), backups, and DNS; as well as the process of writing the code that abstracts and implements the services we require.

This book concentrates on the task of writing infrastructure code that meets these principles in a predictable and reliable fashion. The key enabler in this context is a powerful, declarative configuration management system that enables an engineer (I like the term *infrastructure developer*) to write executable code that both describes the shape, behavior and characteristics of an infrastructure that they are designing, and, when actually executed, will result in that infrastructure coming to life.

The Risks of Infrastructure as Code

Although the potential benefits of Infrastructure as Code are hard to overstate, it must be pointed out that this approach is not without its dangers. Production infrastructures that handle high-traffic websites are hugely complicated. Consider, for example, the

Chapter 1. Infrastructure As Code. Test-Driven Infrastructure with Chef, ISBN: 9781449309718
Prepared for it@online-buddies.com, Joe Marshall
Copyright © 2011 Atalanta Systems LTD. This download file is made available for personal use only and is subject to the Terms of Service. Any other use requires prior written consent from the copyright owner. Unauthorized use, reproduction and/or distribution are strictly prohibited and violate applicable laws. All rights reserved.

mix of technologies involved in a large Drupal installation. We might easily have multiple caching strategies, a full-text indexer, a sharded database, and a load-balanced set of webservers. That's a significant number of moving parts for the engineer to manage and understand.

It should come as no surprise that the attempt to codify complex infrastructures is a challenging task. As I visit clients embracing the approaches outlined in this chapter, I see a lot of problems emerging as they start to put these kind of ideas into practice.

Here are a few symptoms:

- Sprawling masses of Puppet or Chef code.
- Duplication, contradiction, and a lack of clear understanding of what it all does.
- Fear of change: a sense that we dare not meddle with the manifests or recipes, because we're not entirely certain how the system will behave.
- Bespoke software that started off well-engineered and thoroughly tested, but now littered with TODOs, FIXMEs, and quick hacks.
- A sense that, despite the lofty goal of capturing the expertise required to understand an infrastructure in the code itself, if one or two key people were to leave, the organization or team would be in trouble.

These issues have their roots in the failure to acknowledge and respond to a simple but powerful side effect of treating our Infrastructure as Code. If our environments are effectively software projects, then it's incumbent upon us to make sure we're applying the lessons learned by the software development world in the last ten years, as they have strived to produce high quality, maintainable, and reliable software. It's also incumbent upon us to think critically about some of the practices and principles that have been effective there, and start to introduce our own practices that embrace the same interests and objectives. Unfortunately, many of the embracers of Infrastructure as Code have had insufficient exposure to or experience with these ideas.

I have argued elsewhere[‖] that there are six areas where we need to focus our attention to ensure that our infrastructure code is developed with the same degree of thoroughness and professionalism as our application code:

Design
  Our infrastructure code should seek to be simple, iterative, and we should avoid feature creep.

Collective ownership
  All members of the team should be involved in the design and writing of infrastructure code and, wherever possible, code should be written in pairs.

‖ *http://www.agileweboperations.com/the-implications-of-infrastructure-as-code*

Chapter 1. Infrastructure As Code. Test-Driven Infrastructure with Chef, ISBN: 9781449309718
Prepared for it@online-buddies.com, Joe Marshall
Copyright © 2011 Atalanta Systems LTD. This download file is made available for personal use only and is subject to the Terms of Service. Any other use requires prior written consent from the copyright owner. Unauthorized use, reproduction and/or distribution are strictly prohibited and violate applicable laws. All rights reserved.

Code review
: The team should be set up in such a way as to both pair frequently and to see regular notifications of when changes are made.

Code standards
: Infrastructure code should follow the same community standards as the Ruby world; where standards and patterns have grown up around the configuration management framework, these should be adhered to.

Refactoring
: This should happen at the point of need, as part of the iterative and collaborative process of developing infrastructure code; however, it's difficult to do this without a safety net in the form of thorough test coverage of one's code.

Testing
: Systems should be in place to ensure that one's code produces the environment needed, and to ensure that our changes have not caused side effects that alter other aspects of the infrastructure.

The first five areas can be implemented with very little technology, and with good leadership. However the final area—that of testing infrastructure—is a difficult endeavor. As such, it is the subject of this book—a manifesto for bravely rethinking how we develop infrastructure code.

Chapter 1. Infrastructure As Code. Test-Driven Infrastructure with Chef, ISBN: 9781449309718
Prepared for it@online-buddies.com, Joe Marshall
Copyright © 2011 Atalanta Systems LTD. This download file is made available for personal use only and is subject to the Terms of Service. Any other use requires prior written consent from the copyright owner. Unauthorized use, reproduction and/or distribution are strictly prohibited and violate applicable laws. All rights reserved.

Prepared for it@online-buddies.com, Joe Marshall
Copyright © 2011 Atalanta Systems LTD. This download file is made available for personal use only and is subject to the Terms of Service. Any other use requires prior written consent from the copyright owner. Unauthorized use, reproduction and/or distribution are strictly prohibited and violate applicable laws. All rights reserved.