НАДЕЖНЫЙ ДЕПЛОЙ RAILS ПРИЛОЖЕНИЙ

Надежный, повторно используемый деплой приложений и настройка серверов

БЕН ДИКСОН И ЕВГЕНИЙ БУРМАКИН

Надежный деплой Rails приложений

Уверенная настройка серверов и надежный деплой приложений

Ben Dixon and Eugene Burmakin

This book is for sale at http://leanpub.com/deploying_rails_applications-ru

This version was published on 2016-03-03

Also By Ben Dixon

Reliably Deploying Rails Applications

Rails 程序部署之道

Оглавление

1.0 - Введение **1**
Цель этой книги 1
Обо мне 1
Почему эта книга существует 2
Целевая аудитория 3
Что нужно знать и уметь 3
Как читать эту книгу 3
Структура первой части - Настройка сервера 4
Структура второй части - Деплой на сервер 7
Версия 8

2.0 - Набор технологий **9**
Обзор 9
Ubuntu 14.04 LTS 9
Nginx 9
Unicorn 10
Postgresql / MongoDB / MySQL 10
Ruby (rbenv) 10
Redis 11
Memcached 11
Почему именно этот набор технологий 11
Адаптация к вашему стеку 11

3.0 - Термины Chef **12**
Введение 12
Автоматизация 12
Инструменты для автоматической установки и настройки 13

Как купить **15**

1.0 - Введение

Цель этой книги

Эта книга расскажет вам о том, как:

- Настроить VPS с нуля
- Автоматизировать настройку дополнительных серверов за минуты
- Использовать Capistrano для надежного деплоя
- Автоматизировать скучные задачи по обслуживанию

Если у вас есть приложения на Heroku, которые обходятся вам в целое состояние, эта книга обеспечит вас инструментами для миграции ваших приложений на VPS.

Если вы уже размещаете ваше приложение на VPS, но процесс деплоя кажется вам неудовлетворительным - иногда приложение не перезапускается или загружается неверная версия кода - эта книга обеспечит вас шаблонами, с помощью которых вы сможете сделать процесс более надежным.

Я потратил сотни часов, читая посты в блогах, документацию и настраивая файлы конфигурации. Это привело меня к моменту, когда деплой на VPS стал таким же простым, - на самом деле, даже проще - как деплой на Heroku. Если вы хотите научиться тому же самому, эта книга сэкономит вам огромное количество времени.

Обо мне

В настоящее время находясь в лондонском кампусе Google, я разрабатываю веб-приложения более восьми лет. В последние годы специализируюсь на разработке и деплое приложений на Ruby on Rails. Я также работаю в качестве консультанта - в основном, для стартапов - и над собственными проектами.

Я глава технического отдела в фитнесс-стартапе, который предоставляет графики занятий для многих спортивных комплексов в Великобритании, а так же работаю над международным iOS-приложением (Speedo Fit) для пловцов.

В рамках развития инфраструктуры для этого, я имел дело со множеством вещей, начиная со стандартного быстрого роста от 10 запросов в минуту до 10 запросов в секунду, и заканчивая более необычными задачами, такими, как расширение инфраструктуры в Китай и работа с отладкой скрытых проблем с индексами MongoDB.

Я выступал с докладом о деплое Rails-приложений на Railsconf 2014 и с докладом об интеграции Docker с Rails на Railsconf 2015.

Почему эта книга существует

Я пришел к Rails после нескольких лет разработки на PHP и Python. Причиной перехода на этот фреймворк стало то, что он берет большую часть повторяющихся вещей, которые мне нужно сделать, и предоставляет эту функциональность “из коробки”, позволяя мне сосредоточиться на реальной функциональности приложения.

Первая реальная проблема возникла, когда мне потребовалось задеплоить приложение на что-то отличное от Heroku. Мне потребовалось снизить стоимость поддержки сторонних проектов с фоновыми задачами и нескольких крупных проектов в продакшене, где требовалось больше гибкости.

Поиск информации о том, как деплоить Rails-приложения на VPS выдал мне множество руководств, предлагающих вводить каждую команду вручную, и я попробовал этот способ. Это было медленно и требовало повторения одних и тех же действий. Тем не менее, хотя я аккуратно задокументировал процесс, применение этого же способа не сработало в следующий раз, когда мне понадобилось настроить сервер.

Это было очень неудобно по сравнению с процессом разработки, где все, что мне требовалось в нескольких проектах, просто было вынесено в абстракцию в виде гема и использовано снова, но я был упорен и рано или поздно заставлял все работать как положено. Параллельно я общался с людьми из локальных Ruby-групп и другими компаниями, которые проходили через аналогичный процесс.

Наконец, случилось неизбежное и сервер упал окончательно, что означало необходимость полной перенастройки.

К несчастью, в это время я был в деревне во Франции в одном из нечастых отпусков, где единственный стабильный WiFi-сигнал можно было поймать сидя наполовину в гардеробе, на чердаке дачного домика. После почти 12 часов ввода команд, сидя в гардеробе, по соединению, которого едва хватало на поддержку SSH-сессии, я понял, что эту проблему уже наверняка кто-то решил.

Затем я узнал об управлении конфигурациями, поэкспериментировал с Chef, Puppet и Ansible, и в конце концов выбрал Chef. За удивительно короткий промежуток времени при помощи Chef я написал простую конфигурацию, которая позволила мне настраивать сервер для Rails всего несколькими локальными командами, практически гарантируя, что все будет работать так же, как работало на предыдущих серверах.

Эта гарантия значила, что я могу так же использовать стандартную конфигурацию Capistrano для деплоя. Неожиданно деплой Rails-приложения на VPS впервые перешел из категории дел, которые занимают по меньшей мере день в категорию дел, занимающих в худшем случае час.

Поговорив с другими людьми о том, почему они не использовали подобную систему, стало понятно, что управление конфигурацией расценивается как нечто, принадлежащее к администрированию со слишком крутой кривой обучения для большинства разработчиков,

которые просто хотят деплоить свои приложения куда им удобно. Это совсем не вязалось с моим опытом, Chef был простым кодом на Ruby и изучение основ отняло у меня едва ли больше времени, чем обычно отнимала настройка сервера вручную.

Я начал консультировать компании, желающие научиться и внедрить управление конфигурациями, и делиться своими заметками о процессе с коллегами и друзьями для их собственных проектов. Кто-то предположил, что это может стать основой для книги. И вот она перед вами!

Я постарался достичь баланса между предоставлением в примерах кода, который работает “из коробки” и способен выполнять свою задачу, и между тем, чтобы все же дать достаточно деталей и объяснить, как работает каждый компонент, чтобы каждый читатель был способен постепенно разработать шаблон, подходящий под его цели.

Целевая аудитория

Эта книга нацелена на людей, которые разрабатывают приложения на Ruby on Rails и вовлечены в управление инфраструктурой и деплой приложений или полностью этим занимаются.

Даже если деплой приложений - это нечто новое для вас, или вы уже некоторое время занимаетесь этим вручную и хотите использовать более структурированный подход, я надеюсь, что эта книга будет вам полезна.

Что нужно знать и уметь

Предполагается, что любой, кто читает эту книгу, уже имеет опыт в разработке на Ruby on Rails.

Так же предполагается знание некоторых базовых команд unix, в частности, убедитесь, что вы можете:

- Использовать SSH для соединения с удаленными серверами
- Перемещаться по файловой системе в терминале (cd, ls, dir, mv, rm, и так далее)
- Имеете базовое представление о веб архитектуре. Например, что такое сервер, как настроить DNS, и так далее

Как читать эту книгу

Эта книга строится вокруг кода примеров, которые доступны на Github. В целом, в каждой главе будет рассмотрен код примера, объяснено, как он работает и как его использовать.

Начиная с шестой главы, вместо слов “а теперь добавим этот код к ƒайлу x”, рассматриваемый код будет воспроизведен в тексте, а его расположение в коде примера будет помечено.

Ключевые команды для выполнения конкретных операций будут воспроизведены, но не на уровне

```
cd xxxxx
mv yyy zzzzz
etc etc
```

Так же важно читать эту книгу, имея доступ к коду. Хотя каждая глава может быть прочитана и вы поймете все, что в ней написано без прямой отсылки к полному коду примера, важно потратить время на эксперименты и понять, где в конечном итоге должен располагаться каждый ƒайл.

Я бы по меньшей мере рекомендовал прочитывать каждую главу, затем открывать ƒайлы, о которых идет речь, в репозитории с примерами и убеждаться, что вы понимаете их содержимое и структуру, прежде чем переходить к следующей главе. У каждого примера кода в тексте есть заголовок, в котором отражено его расположение.

Цель в том, чтобы к концу книги вы имели шаблон, настроенный под ваши собственные нужды и позволяющий деплоить Rails-приложение на сервер в считанные минуты.

Структура первой части - Настройка сервера

Первая часть сосредоточена на автоматизированной установке и использовании Chef в качестве инструмента для управления конƒигурацией. Даже если вы используете другой инструмент, такой как Ansible, главы 7 - 15 по-прежнему должны быть полезными.

Глава 2.0 - Набор технологий

Обзор компонентов, составляющих нашу конƒигурацию для сервера, их предназначение и беглый взгляд на причины, по которым мы выбрали каждый из них.

Глава 3.0 - Определения Chef

Кратко о том, почему использование инструмента автоматизации для настройки серверов так важно и о некоторых терминах, с которыми мы столкнемся, используя избранный нами инструмент автоматизации - Chef.

Глава 4.0 - Быстрый Старт

Глава для тех, кто читает инструкции после того, как все сделано. Эта глава предоставит пошаговые инструкции для настройки полностью работоспособного, воспроизводимого Rails-сервера, при помощи примеров кода из книги и нескольких простых команд

Глава 5.0 - Анатомия Проекта Chef Solo

Более глубокое погружение в Chef и Chef Solo, связанные с ними термины и то, как создать проект Chef с нуля. Думайте об этом как о `rails new`, только для настройки сервера.

Глава 5.1 - Создание Chef Cookbook

Детальное руководство по тому, как создавать собственные кукбуки для Chef. Кукбуки подобны гемам, но используются для установки и настройки компонентов на сервере. Мы начнем с команд, которые будем вводить вручную, чтобы установить программу на сервер, а затем конвертируем полученное в простейший кукбук для автоматизации процесса. Постепенное усложнение позволит нам воспользоваться некоторыми из самых мощных возможностей Chef.

Глава 5.2 - Определение Нод и Ролей

Первый взгляд на концепт “определения ноды”, json-файл, который определяет все аспекты создания сервера. Затем мы посмотрим на то, как Chef позволяет нам создавать повторно используемые элементы функциональности, которые позднее могут быть применены ко множеству серверов или повторно использованы в будущих проектах при помощи “ролей”.

Глава 6.0 - Шаблон для Rails-серверов

Более детальное введение в конфигурацию, которую мы использовали в Главе 4 в качестве примера для быстрого запуска полностью работоспособного сервера в считанные минуты

Глава 6.1 - Управление Кукбуками при помощи Berkshelf

Для Rails мы используем bundler, чтобы быть уверенными в корректности установленных зависимостей, для Chef у нас есть Berkshelf, который выполняет точно такую же функцию. С Berkshelf мы можем быть уверенными в том, что имеем все необходимые кукбуки для настройки сервера, с легкостью дополнять написанные сообществом кукбуки собственной функциональностью.

Глава 7.0 - Базовая настройка сервера

Базовые компоненты, необходимые на любом сервере, как - и когда - автоматически обновлять пакеты, как настроить автоматическое обновление системного времени и почему так много людей попадаются на локали.

Глава 8.0 - Безопасность

Начнем с рассмотрения некоторых типичных проблем в безопасности в процессе деплоя на свежий VPS. Затем более детально взглянем на блокировку доступа по SSH, управление правилами ƒайрвола и автоматическую настройку пользователей и публичных ключей

Глава 9.0 - Настройка сервера для Rails

Убедимся, что у нас есть некоторые базовые программы и инструменты, необходимые для установки стандартных гемов, а затем детально рассмотрим установку Ruby и убедимся, что имеем корректную версию для нашего Rails-приложения.

Глава 10.0 - Monit

Когда что-то ломается, в идеальном мире, оно чинится само. Эта глава расскажет о том, как использовать Monit для автоматического мониторинга работоспособности сервисов и перезапускать их, когда они падают. Если же и перезапуск не удался, Monit должен будет уведомить нас через email, чтобы мы могли вмешаться и все исправить.

Глава 11.0 - Nginx

Nginx будет первым, кто обрабатывает поступающие запросы. Как настраивать виртуальные хосты, обслуживать статические ресурсы и зачем нам вообще нужен Nginx.

Глава 12.0 - PostgreSQL

Как установить и настроить PostgreSQL, как управлять разными видами аутентификации, убедиться, что у нас установлена необходимая версия и как управлять импортом и экспортом данных

Глава 13.0 - MySQL

Как установить и настроить MySQL, как управлять разными видами аутентификации и убедиться, что у нас установлены необходимые для гема MySQL компоненты. Наконец, краткое введение в импорт и экспорт данных.

Глава 14.0 - MongoDB

Кратко о том, когда MongoDB является подходящим хранилищем данных, затем убедимся, что у нас установлена корректная версия и поговорим о работе с кукбуками сообщества для установки новых версий. Так же кратко рассмотрим импорт и экспорт баз данных.

Глава 15.0 - Redis и Memcached

Короткая глава, посвященная двум самым простым элементам нашего стека. Как установить Redis и Memcached, определить, должны ли они быть привязаны к локальному хосту и настроить максимальный размер, которого они могут достичь.

Структура второй части - Деплой на сервер

Глава 16.0 - Быстрый старт с Capistrano

Как и в глава 4, эта глава для тех, кто предпочитает сначала делать, а потом уже читать инструкции. Эта глава рассматривает минимальные шаги, необходимые для деплоя существующего Rails-приложения на наш новый сервер.

Глава 16.1 - Деплой при помощи Capistrano

Намного более детальное введение в то, как деплоить на наш сервер, используя Capistrano 3, не используя гемы помимо тех, что входят в состав ядра Capistrano. Включает в себя краткий обзор разницы между 2 и 3 версиями инструмента и детальный раздел по управлению конфигурационными файлами, отличающимися от сервера к серверу.

Глава 16.2 - Написание собственных заданий для Capistrano

Наша цель в том, чтобы автоматизировать все, что касается взаимодействия с нашим сервером. В конце концов мы должны будем получить рецепт, который никто еще не автоматизировал таким способом, чтобы мы могли его повторно использовать. Эта глава рассматривает написание собственных заданий для Capistrano. Когда мы научимся их писать, мы сможем автоматизировать практически любое взаимодействие с нашим сервером, какое только можно представить.

Глава 17.0 - Конфигурация Unicorn и Zero Downtime Deployment

Как настроить деплой таким образом, чтобы, когда мы деплоим, происходило незаметное для пользователя переключения с одной версии кода на другую, без задержки, вызванной запуском новой версии. Включает раздел, где детально рассматриваются причины потенциальных проблем при такой настройке и предлагаются решения на случай, если такая настройка недостаточно хорошо работает.

Глава 18.0 - Виртуальные хосты и SSL

Более глубокий взгляд на виртуальные хосты Nginx, которые контролируют распределение запросов, полученных Nginx. Так же рассматривается настройка SSL и управление ротацией сертификатов.

Глава 19.0 - Sidekiq

Sidekiq является одной из самых популярных и эффективных библиотек для работы с фоновыми задачами в экосистеме Rails. Здесь мы рассмотрим, как интегрировать запуск и остановку воркеров Sidekiq в качестве части процесса деплоя и как настроить мониторинг таким образом, чтобы они автоматически перезапускались в случае падения.

Глава 20.0 - Автоматизация бэкапов

Печально признавать, но рано или поздно сервер упадет. В этой главе мы настроим автоматический бэкап базы данных на Amazon S3 (или на любое количество других сервисов), чтобы быть уверенными, что в случае полного отказа сервера наши данные могут быть восстановлены.

Версия

Вы читаете версию 3.1.1 этой книги. Первое и второе (x.x) числа отражают номер версии в примерах кода. Последнее число ссылается на мелкие исправления, такие, как опечатки. Купив эту книгу однажды, вы сможете скачивать последние её версии с Leanpub в любое время.

2.0 - Набор технологий

Обзор

Технологии, используемые для примера в этой книге, являются лишь одной из множества возможных конфигураций. Я выбрал комбинацию компонентов, которые, как я обнаружил, наиболее распространены и используемы.

Если часть вашего стека не описана здесь, не отчаивайтесь: все общие принципы будут всегда одинаковыми.

Если вы знаете или можете найти инструкцию по тому, как установить необходимый компонент, то сможете применить принципы автоматизации с Chef для его установки.

Ubuntu 14.04 LTS

Ubuntu стала одной из наиболее распространенных операционных систем, применяемых в серверном окружении за последние четыре или пять лет. Главная причина, по которой я рекомендую её - это высокий уровень поддержки со стороны сообщества.

Благодаря тому, что этой системой пользуется огромное количество людей с самыми разными навыками, вы практически никогда не столкнетесь с проблемой, которую не решал никто до вас. В 99% случаев быстрый поиск по Stack Overflow направит вас в нужном направлении.

Для оставшегося 1% есть чрезвычайно дружелюбное и открытое сообщество.

Nginx

Nginx - это веб-сервер и обратный прокси, известный своей высокой производительностью и малым потреблением памяти.

Ключевое преимущество nginx заключено в его событийной архитектуре, использование памяти чрезвычайно предсказуемо, даже под высокими нагрузками. Это свойство делает его идеальным для проектов, которые могут начинаться с маленького VPS для тестирования и постепенно вырасти до значительно больших размеров.

Unicorn

Unicorn - это веб (HTTP) сервер для Rack-приложений (одним из которых является Rails).

Если вы хотите понять, как работает Unicorn, я настоятельно рекомендую прочитать на Github статью, посвященную причинам, по которым они отказались от Mongrel в пользу Unicorn:

https://github.com/blog/517-unicorn

Кратко, архитектура такова: когда запросы приходят на сервер, они обрабатываются Nginx, который далее передает их Unicorn, а он, в свою очередь, передает их запущенному Rails-приложению и возвращает ответ.

Чтобы узнать больше о преимуществах Unicorn относительно Passenger и почему их подходы различаются, прочтите этот пост Engineyard; https://blog.engineyard.com/2012/passenger-vs-unicorn

Postgresql / MongoDB / MySQL

PostgreSQL и MySQL - традиционные реляционные базы данных. MySQL, вероятно, более известна благодаря своему послужному списку, поскольку является довольно частым выбором в крупных организациях.

PostgreSQL некоторое время назад стала практически стандартной базой данных для Rails-приложений, использующих реляционные БД, отчасти, благодаря тому, что именно она поддерживается Heroku по умолчанию. Я предпочитаю использовать именно эту базу данных в большинстве случаев, потому что с добавлением нативной поддержки JSON она стала лучшим инструментом, позволяющим сочетать преимущества традиционных реляционных баз данных и NoSQL-решений, таких как Mongo.

MongoDB - нереляционная база данных, чрезвычайно популярная в случаях, когда необходимо хранить большие объемы неструктурированных данных.

Примеры кода включают в себя установку и настройку всех трех перечисленных БД. Мое личное предпочтение для большинства приложений - PostgreSQL, в работе с ней я имею больше всего опыта, поэтому эта БД будет использована в большинстве примеров.

Ruby (rbenv)

Эта книга рассматривает и была протестирована на Ruby 1.9.x, Ruby 2.0.x, Ruby 2.1.x и Ruby 2.2.x. В то время, как многие другие способы могут работать с более экзотичными видами, такими, как JRuby, мой опыт взаимодействия с ними минимален, поэтому я не буду рассматривать их напрямую.

Мой выбор для управления и установки версий Ruby на продакшен-сервера - rbenv. В основном это так, потому что я считаю работу rbenv простой для понимания (и, как следствие, для решения проблем), в то время как работу rvm я нахожу гораздо более сложной.

Из интереса я пытался собирать сервера на обеих системах управления версиями Ruby, и при работе с rbenv проблем было меньше, чем в случае с rvm. Это совершенно не должно восприниматься как критика rvm. Больше деталей о том, как именно работает rbenv, вы найдете в главе 9.

Redis

Redis - это чрезвычайно быстрое хранилище данных вида “ключ-значение”. Он прекрасно подходит для таких вещей, как кэширование и ограничение количества запросов к api.

Есть несколько неочевидных моментов, таких, как его поведение при достижении максимального лимита памяти, мы поговорим об этом в разделе, посвященном настройке redis.

Memcached

Похож на Redis, но хранится полностью в памяти (перезагрузи систему и потеряешь все, что было в Memcached). Отлично подходит для кэширования и, как и Redis, удивительно прост в установке и поддержке.

Почему именно этот набор технологий

Эти компоненты покрывают требования большинства приложений, с которыми я сталкивался в течение последних нескольких лет. В целом, я довольно нейтрален в вопросе “какой стек лучше всех”. Существует множество других отличных сочетаний, и идеи, описанные в этой книге, применимы к большей их части.

Это, с правильно подобранной базой данных, мой набор технологий для нового проекта, и я надеюсь, что он послужит хорошим стартом так же и для ваших проектов.

Адаптация к вашему стеку

Если ваш стек не описан выше, не переживайте. Цель этой книги в том, чтобы показать, как легко использовать Chef для автоматизации настройки любого Rails-стека, описанные компоненты - лишь пример. К концу книги вы будете способны написать свои собственные рецепты для Chef, которые позволят установить и настроить практически что угодно.

Chef используется только для настройки сервера, не для деплоя. Вторая часть книги - деплой при помощи Capistrano - будет полезна вне зависимости от того, какой инструмент для управления конфигурациями вы используете

3.0 - Термины Chef

Введение

“Простейший” способ установить и настроить новый сервер - создать новый VPS где-нибудь вроде Linode или Digital Ocean, войти через SSH и начать устанавливать необходимые пакеты, иногда заходя в vim, чтобы поправить настройки и добавить несколько новых источников пакетов, где это необходимо.

Когда нужно что-то новое, мы снова входим по ssh, устанавливаем или обновляем пакеты, наращивая слои сервера. Где-то наверняка есть текстовый файл или wiki-страница со “всеми командами, которые нам нужны” на случай, если нам понадобится снова проделать эти операции.

Этот подход имеет несколько важных проблем:

1) Очень сложно отслеживать, что было сделано. Даже в лучшем случае текстовый файл не будет содержать все необходимые команды. Пока не возникнет необходимость в новом сервере (зачастую, при неблагоприятных условиях), никто не узнает, что описанные команды могут не подходить для его настройки.

2) Это медленно (а значит, дорого). Даже если процесс превосходно задокументирован, кому-то все равно нужно будет сидеть и вводить команды. Это нерациональная, скучная работа, но зачастую, её придется выполнять инженеру, чье время следовало бы потратить на работу над самим продуктом.

3) Это не масштабируется. Иметь инженера, который может вводить команды и обладает достаточной дисциплиной, неплохо, когда речь идет лишь об одном сервере. Добавьте еще пять или шесть серверов и тут же начнутся проблемы.

Автоматизация

Цель первой части и этой книги в целом в том, чтобы взять ручные процессы и автоматизировать их. Главное правило состоит в том, что любой процесс, который я должен буду повторять более одного или двух раз в год на протяжении жизненного цикла деплоя и менеджмента, я испытываю и автоматизирую.

Автоматизация полагается на подход больше, чем на какой-либо конкретный инструмент. Если мы делаем что-либо, что включает запуск более одной команды или входа по ssh на удаленный сервер более одного раза в месяц, вероятнее всего, стоит остановиться и подумать: “как это можно автоматизировать?”

Преимущества зачастую видны уже через короткий промежуток времени. Автоматизация деплоя на сервер не только облегчает исправление ситуации в случае возникновения проблем, она значительно упрощает создание тестового и стейджинг окружений, делает более простым и эффективным тестирование новых деплоев и, таким образом, сокращает время простоя.

Автоматизация копирования баз данных с продакшена в среды тестирования и стейджинга позволяет разработчикам использовать более или менее актуальные данные в их тестах. Это делает тесты в девелопмент-среде более осмысленными, а значит, увеличивает продуктивность и снижает расходы.

Наконец, автоматизировать, как правило, проще, чем кажется. Однажды освоив Chef для автоматизации задач по настройке серверов и Capistrano для автоматизации задач по деплою и последующим взаимодействиям с продакшен и стейджинг-серверами, вы поймете, что автоматизация дополнительных задач редко отнимает больше времени, чем потребовалось бы на однократное их вы выполнение.

Инструменты для автоматической установки и настройки

Chef и Chef Solo

Chef - платформа для автоматизации, созданная компанией Opscode и использующая ruby DSL (Domain Specific Language), чтобы предоставлять команды, необходимые для установки и настройки сервера в подходящем для повторного использования формате.

При помощи Chef вы можете определить шаги, необходимые для настройки сервера, призванного выполнять “роль”, например, сервер для приложения на Rails, или сервер баз данных, и затем применить комбинацию этих ролей к конкретной удаленной машине.

Chef часто работает централизованно. Центральный сервер Chef “знает” роли, которые должны быть применены к большому количеству других серверов. Если вы обновляете роль, изменения применяются ко всем этим серверам автоматически.

Несмотря на то, что это великолепный и очень мощный способ конфигурирования, если у вас есть десятки или сотни серверов, такой подход видится чрезмерно усложненным, если нам нужно управлять 1-10 серверами.

К счастью, chef также может работать и в “соло”-конфигурации (chef-solo). В этом случае мы используем наше локальное окружение чтобы определить конфигурации и роли серверов, и затем вручную применяем эти конфигурации к серверам, как и когда нам необходимо. Это функционально идентично тому, как работают клиентские инструменты, такие, как Ansible.

Это идеально для небольших проектов, все элементы которых работают на одном сервере, хотя я также с большим успехом использовал этот подход для настройки до десяти связанных серверов.

Если ваш проект вырастет за эти рамки, практически все задачи, автоматизированные при помощи chef solo, могут быть повторно использованы на chef server.

Knife и Knife Solo

Knife - инструмент командной строки (cli), предоставляющий интерфейс для взаимодействия между локальным (размещенным на нашей машине) chef-репозиторием и удаленным сервером.

Традиционно этот удаленный сервер был бы главным “chef сервером”, но дополнительный инструмент “knife solo” позволяет нам использовать chef в соло-режиме и взаимодействовать напрямую с сервером, который мы хотим настроить.

Больше информации о knife solo доступно на его странице на Githib:

https://github.com/matschaffer/knife-solo

Полная инструкция по использованию будет описана в следующей главе.

Berkshelf

В терминологии chef команды для установки индивидуального компонента на систему называются “рецептами” (“recipe”). Например, вы можете иметь рецепт для настройки ruby и другой рецепт для установки основных зависимостей rails. Несколько рецептов, относящихся к конкретному функциональному элементу (например, MySQL сервер и MySQL клиент) зачастую можно объединить в Кукбук (Cookbook).

Berkshelf выполняет для рецептов роль, аналогичную той, что Bundler выполняет для гемов. Вы определяете рецепты, от которых зависит ваша конфигурация, например, рецепты “rbenv” и “rails_gem_dependencies”. Так же, как команда “bundle install” устанавливает для вас необходимые гемы (включая конкретные версии), Berkshelf получает для вас рецепты chef (включая конкретные версии), от которых зависит конфигурация вашего сервера.

Как купить

Эта книга опубликована через LeanPub. Вы можете купить её здесь: https://leanpub.com/deploying_rails_applications-ru/.

Купив её однажды, вы будете автоматически получать бесплатные обновления по мере их публикации. Книга регулярно обновляется, чтобы оставаться актуальной с выходом новых версий используемых инструментов.