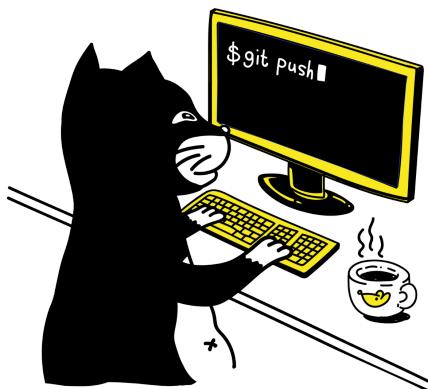


# *Deployment from Scratch*



by Josef Strzibny

1st edition

Version 1.1

# Contents

<b>1</b>	<b>Introduction</b>	<b>14</b>
1.1	Acknowledgment	15
1.2	Expectations	16
1.3	Conventions	17
1.4	Feedback	18
<b>2</b>	<b>Bird's Eye</b>	<b>19</b>
2.1	High-Level Concepts	20
<b>3</b>	<b>Operating Systems</b>	<b>23</b>
3.1	Fedora and Friends	24
3.2	Terminals and Shells	26
3.2.1	Commands	26
3.2.2	Standard Streams	30
3.2.3	File Editing	34
3.2.4	Navigation	35
3.2.5	Pipelines	37
3.3	Summary	39

<b>4 Little Bit of Network Theory</b>	<b>40</b>
4.1 Mental Model . . . . .	41
4.1.1 Signals and Waves . . . . .	42
4.1.2 Internet . . . . .	44
4.1.3 Transport . . . . .	50
4.1.4 Applications . . . . .	52
4.2 Summary . . . . .	55
<b>5 Secure Connections</b>	<b>56</b>
5.1 SSH and OpenSSH . . . . .	57
5.1.1 SSH Keys . . . . .	58
5.2 A First Server . . . . .	60
5.3 Remote Shell . . . . .	62
5.4 File Transfers . . . . .	64
5.5 SSH on Servers . . . . .	65
5.6 SSH Tunneling . . . . .	69
5.7 Summary . . . . .	71
<b>6 Hands-On Networking</b>	<b>72</b>
6.1 Network Interfaces . . . . .	73
6.2 MAC and IP Addresses . . . . .	75
6.3 Sending and Receiving Traffic . . . . .	79
6.4 Hostnames . . . . .	84
6.5 Ports and Sockets . . . . .	87
6.6 Reboots . . . . .	91
6.7 Summary . . . . .	92

<b>7</b>	<b>Server Configuration 101</b>	<b>93</b>
7.1	Software Packages . . . . .	94
7.2	System Managers . . . . .	95
7.3	Package Repositories . . . . .	98
7.3.1	Package Modules . . . . .	104
7.3.2	Repository Configuration . . . . .	106
7.3.3	Repository Data . . . . .	109
7.4	Audits . . . . .	110
7.5	Changing Configuration . . . . .	111
7.6	Automation . . . . .	117
7.6.1	Bash Options . . . . .	119
7.6.2	Working Directory . . . . .	121
7.6.3	Conditionals . . . . .	123
7.6.4	Functions . . . . .	125
7.6.5	Shebangs . . . . .	127
7.7	Summary . . . . .	128
<b>8</b>	<b>Filesystems</b>	<b>129</b>
8.1	Device Files . . . . .	130
8.2	Paths, Paths, Paths . . . . .	133
8.2.1	Web Applications . . . . .	136
8.2.2	Profiles . . . . .	138
8.2.3	FHS Layout . . . . .	139
8.3	Lost and Found . . . . .	140
8.4	Summary . . . . .	142

<b>9 User Roles</b>	<b>143</b>
9.1 Users . . . . .	144
9.1.1 Switching Users . . . . .	148
9.1.2 Running Commands as Superuser . . . . .	149
9.2 Groups . . . . .	154
9.3 Summary . . . . .	158
<b>10 Permissions</b>	<b>159</b>
10.1 Discreet Permissions . . . . .	160
10.1.1 File Mode . . . . .	164
10.1.2 Changing Permissions . . . . .	167
10.1.3 Changing Ownership . . . . .	169
10.2 Access Control List . . . . .	171
10.3 Summary . . . . .	175
<b>11 Processes</b>	<b>176</b>
11.1 Computers . . . . .	177
11.2 Daemons . . . . .	179
11.3 Process Managers . . . . .	182
11.4 Monitoring . . . . .	186
11.4.1 /proc . . . . .	186
11.4.2 ps . . . . .	189
11.4.3 top and htop . . . . .	192
11.4.4 w, pstree and uptime . . . . .	194
11.5 Logging . . . . .	195
11.6 Tracing . . . . .	198
11.7 Termination . . . . .	200
11.8 Capabilities . . . . .	202

11.9 Niceness . . . . .	204
11.10 Swap . . . . .	205
11.11 Recurring Runs . . . . .	206
11.12 Syncing Time . . . . .	210
11.13 Summary . . . . .	212
<b>12 Web Servers</b>	<b>213</b>
12.1 Kinds of Web Servers . . . . .	214
12.2 Running a Web Server . . . . .	215
12.3 NGINX Configuration . . . . .	215
12.4 Static Websites . . . . .	227
12.5 Nice URLs . . . . .	229
12.6 Configuring Redirects . . . . .	230
12.7 Separating Multi-Site Configurations . . . . .	232
12.8 Reverse Proxy . . . . .	233
12.8.1 Using TCP Sockets . . . . .	234
12.8.2 Using UNIX Sockets . . . . .	235
12.8.3 Server References . . . . .	238
12.9 Restricting Access . . . . .	239
12.10 Logging . . . . .	240
12.10.1 Log Rotation With logrotate . . . . .	241
12.11 Summary . . . . .	244
<b>13 Domain Names and Certificates</b>	<b>245</b>
13.1 Domains . . . . .	246
13.2 DNS Records . . . . .	247
13.2.1 Looking-up Records . . . . .	249
13.2.2 Changing Records . . . . .	251

13.3 HTTPS and Certificates . . . . .	252
13.4 Implementing HTTPS . . . . .	254
13.4.1 Security Options . . . . .	256
13.4.2 Redirecting HTTP . . . . .	258
13.5 Let's Encrypt Certificates . . . . .	259
13.5.1 Renewals . . . . .	261
13.6 Summary . . . . .	263
<b>14 Firewalls</b>	<b>264</b>
14.1 firewalld . . . . .	265
14.1.1 Installation . . . . .	265
14.1.2 Overseeing firewalld . . . . .	266
14.1.3 Changing firewalld Rules . . . . .	268
14.1.4 Creating Custom Zones . . . . .	270
14.1.5 Defining Services . . . . .	271
14.1.6 Rich Rules . . . . .	273
14.1.7 ICMP . . . . .	274
14.2 Scanning ports with nmap . . . . .	275
14.3 fail2ban . . . . .	278
14.4 Service Provider Firewalls . . . . .	281
14.5 Summary . . . . .	282
<b>15 Bashful Configuration Management</b>	<b>283</b>
15.1 Convention over Configuration . . . . .	284
15.2 Building a Framework . . . . .	288
15.3 Structure . . . . .	296
15.4 Conclusion . . . . .	298

<b>16 Language Runtime</b>	<b>299</b>
16.1 Interpreters and Compilers . . . . .	300
16.2 Ruby . . . . .	302
16.2.1 System Ruby . . . . .	302
16.2.2 ruby-install . . . . .	304
16.2.3 chruby . . . . .	305
16.2.4 Bundler . . . . .	307
16.3 Python . . . . .	310
16.3.1 System Python . . . . .	310
16.3.2 pyenv . . . . .	312
16.3.3 venv . . . . .	314
16.4 Summary . . . . .	317
<b>17 Application Servers</b>	<b>318</b>
17.1 Threads or Processes? . . . . .	319
17.1.1 Concurrency and Parallelism . . . . .	319
17.1.2 Threading . . . . .	321
17.2 Zero-Downtime Deployment . . . . .	323
17.3 Configuration . . . . .	324
17.4 Ruby Applications . . . . .	326
17.4.1 Puma Configuration . . . . .	328
17.5 Python Applications . . . . .	333
17.5.1 Gunicorn Configuration . . . . .	334
17.6 Summary . . . . .	338

<b>18 Building Services</b>	<b>339</b>
18.1 Defining Services . . . . .	340
18.1.1 Service Units . . . . .	343
18.1.2 User Services . . . . .	351
18.2 Application Services . . . . .	352
18.2.1 Socket Activation . . . . .	353
18.2.2 Puma Configuration . . . . .	355
18.2.3 Gunicorn Configuration . . . . .	356
18.3 Modifying Services . . . . .	357
18.4 Cgroups . . . . .	359
18.5 Summary . . . . .	363
<b>19 Databases and Key-Value Stores</b>	<b>364</b>
19.1 Running PostgreSQL . . . . .	365
19.1.1 Installation . . . . .	365
19.1.2 System Service . . . . .	366
19.1.3 Databases . . . . .	368
19.1.4 Client Authentication . . . . .	370
19.1.5 Secure Connection . . . . .	375
19.1.6 Database Roles . . . . .	377
19.1.7 Backups . . . . .	378
19.1.8 Upgrades . . . . .	380
19.2 Running Redis . . . . .	382
19.2.1 Installation . . . . .	382
19.2.2 System Service . . . . .	383
19.2.3 Databases . . . . .	384
19.2.4 Connections . . . . .	386
19.2.5 Security . . . . .	388

19.2.6	Backups	392
19.2.7	Restoring Backups	395
19.2.8	Performance	396
19.3	Summary	398
<b>20</b>	<b>SELinux</b>	<b>399</b>
20.1	Policies	400
20.2	Targeted SELinux	402
20.2.1	User and Role Enforcement	405
20.2.2	Type Enforcement	406
20.3	File Contexts	408
20.4	SELinux Booleans	409
20.5	SELinux Modes	410
20.6	Troubleshooting SELinux	411
20.6.1	auditd logs	412
20.6.2	sealert and audit2allow	413
20.7	Writing Policy Files	416
20.7.1	Policy Rules	418
20.7.2	Custom Policy Modules	421
20.8	Summary	423
<b>21</b>	<b>Storage Concerns</b>	<b>424</b>
21.1	Storage	425
21.1.1	Block Storage	426
21.1.2	NFS	430
21.2	Summary	434

<b>22 Backups and Restores</b>	<b>435</b>
22.1 Collection and Compression . . . . .	436
22.2 Encryption . . . . .	438
22.3 Summary . . . . .	441
<b>23 Secrets Management</b>	<b>442</b>
23.1 Keeping Secrets . . . . .	443
23.2 Environment Variables . . . . .	444
23.3 Rails Encrypted Credentials . . . . .	445
23.4 Summary . . . . .	447
<b>24 Application Deployment</b>	<b>448</b>
24.1 Deployment Strategies . . . . .	449
24.2 Typical Concerns . . . . .	450
24.2.1 Loading the Environment . . . . .	451
24.2.2 Installing Dependencies . . . . .	451
24.2.3 Running Migrations . . . . .	454
24.2.4 Preparing Assets . . . . .	454
24.2.5 Restarting Services . . . . .	455
24.3 Deploying with Git . . . . .	457
24.3.1 git-init . . . . .	457
24.3.2 git-push . . . . .	458
24.3.3 git-checkout . . . . .	459
24.3.4 post-receive . . . . .	459
24.4 Summary . . . . .	461

<b>25 Email Delivery</b>	<b>462</b>
25.1 What's in an email? . . . . .	463
25.2 Sending emails . . . . .	464
25.2.1 First Try . . . . .	464
25.2.2 Reverse DNS . . . . .	467
25.2.3 SPF, DKIM, and DMARC . . . . .	468
25.2.4 Relay Servers . . . . .	470
25.3 Receiving emails . . . . .	472
25.4 Summary . . . . .	475
<b>26 Linux Containers</b>	<b>476</b>
26.1 Runtimes and Images . . . . .	477
26.2 Container Blueprint . . . . .	478
26.3 Running Containers . . . . .	480
26.3.1 Podman . . . . .	483
26.3.2 Volumes . . . . .	486
26.4 Networks . . . . .	487
26.5 Restricting Containers . . . . .	491
26.6 Building Containers . . . . .	493
26.6.1 Containerfiles . . . . .	494
26.6.2 Rootless Containers . . . . .	497
26.6.3 Application Containers . . . . .	499
26.6.4 Multi-Target Builds . . . . .	501
26.7 Cleaning Up . . . . .	505
26.8 Summary . . . . .	506

<b>27 Scaling</b>	<b>507</b>
27.1 Considerations and Misconceptions . . . . .	508
27.2 Single Server . . . . .	512
27.3 Multiple Servers . . . . .	514
27.4 Load Balancing . . . . .	516
27.4.1 Implementation . . . . .	518
27.4.2 Failover . . . . .	521
27.5 Multi-Server Deployment . . . . .	521
27.6 Instance Types . . . . .	525
27.7 Summary . . . . .	528
<b>28 Fortune Telling</b>	<b>529</b>
28.1 Next Steps . . . . .	530
28.2 Closing Words . . . . .	532

# Chapter 1

## Introduction

If you are like me, you like to see the applications you write running in production. You get excited when you finally see your hard work delivering real value to people. But web applications don't run in a vacuum, not even the serverless ones.

We need servers with compatible processor architecture and connectivity. On them, we run operating systems, web servers, databases, and firewalls. We make backup copies of user data. Scale services up and down.

Maybe you are not running this show currently, but you could be.

Together, we'll get a full-stack web application up and running on a virtual server in the cloud and focus on transferable skills that you can depend on in your career for a long time to come.

Apart from a detailed look at systems configuration, we'll automate typical deployment tasks. You'll see how things fit together.

There is a lot to learn, but remember, you don't have to master it all from the get-go. Improve as you go.

Josef

## 1.1 Acknowledgment

These people helped me a lot to make this book a reality:

- **Chan Tan-Lui** is a talented graphic designer and animator who draw and animated Tiger the Cat for me.
- **Hannibal** helped me tremendously in the beginning to realize a lot about my early drafts.
- **Dave Woodall** helped me polish the book in later stages of my writing.

I am also thankful for any comment, reply, or feedback I got to make this book. If you helped me in any way or form, know that I am incredibly grateful.

## 1.2 Expectations

This book is meant to be read from the beginning to the end and the knowledge from the previous chapters will be always assumed. If you are already more experienced, feel free to cherry-pick chapters that interests you.

I expect you have already seen a UNIX-like system before. If you are on macOS or Windows, you'll need to set up the SSH tooling on your workstation and be already familiar with your system shell. Everything else will be done on a Linux system in the cloud.

I target CentOS 8 and Rocky Linux 8. If your provider doesn't have images for them, you can also use CentOS Stream 8 or Fedora. You can also use RHEL 8, but you need to have a subscription (consult official documentation on how to use the subscription manager).

## 1.3 Conventions

There are a couple of conventions in the book.

The code, snippets, and command output comes in the following boxes:

```
$ who  
strzibny  tty2          2020-09-30 16:20  (tty2)
```

Please note that I had to sometimes edit the outputs for clarity. Also, I had to wrap lines myself from time to time. As a result, they might continue on the following line.

Concepts and references come inside the following grey box:



*Linux is a UNIX-like POSIX-based operating system kernel, the part of your system providing the abstraction of hardware and means to run concurrent programs.*

And finally, commentary of Tiger the Cat comes in the yellow box:



*Don't mind me, I am extra.*

## 1.4 Feedback

Please e-mail me any feedback, errata, and questions you might have.

The e-mail is [strzibny@strzibny.name](mailto:strzibny@strzibny.name).

# Chapter 2

## Bird's Eye

The process of deploying web applications from scratch involves three kinds of activities: provisioning hardware, installing and configuring software, and deploying application code. There are multiple ways of how we can do this. Manual to automated, dedicated servers to cloud, vertical and horizontal scaling, virtual machines, or container orchestration.

There is no single right way of doing deployments, but there are certain traits of a job well done. Security is one of those. So is performance and a handful of others. But what is the quality we should strive for? A thorough understanding of what we are doing. That's where our focus will be. But now, let's start with a bird's-eye view of a general deployment process.

## 2.1 High-Level Concepts

Deployment in the context of this book is preparing the web application for use. In other words, it's the steps, processes, and activities necessary for making the application to start accepting requests. We usually talk about the deployment in the following stages that sometimes overlap: Provisioning, Configuration, and Application Deployment.

**Server provisioning** is the process of setting up infrastructure and the essential system software. Be it bare-metal dedicated servers or virtual machines in the cloud; provisioning means getting them up and running. This involves installing an operating system, configuring networking, and preparing the SSH access. With providers like Digital Ocean, a newly provisioned virtual machine is just a few clicks away.

You may have heard about **Infrastructure as Code** (IaC), which aims to automate this initial setup with code. Infrastructure as code allows us to write a machine-readable definition of what servers have to be provided. By specifying their compute size, count, provider, and region we can do for infrastructure what we'll do for the system configuration in this book.

You can imagine IaC as writing a program consuming a server provider's API. The program would call an endpoint `/servers` with a payload of what kind of servers you want. You would get back their IP addresses. An example of a popular tool in this space is Terraform, which implements popular cloud providers' APIs.



*The real power behind IaC is the ability to consolidate a lot of things together. Depending on the tool and provider in question, you could define DNS, firewalls, object storage, load balancers, and more.*

But automating everything every single time might not be the best use of our time. When the application architecture and incoming traffic don't require more than a few virtual servers, we can create new machines by hand. A lot of applications can run sufficiently on a machine or two. With managed load balancers and databases, there might be little incentive to add the complexity of yet another tool.

Soon, we'll provision virtual servers from providers like Digital Ocean, Vultr, and Hetzner from their customer admin area. It's an ideal place to start and

– to the disbelief of many – sufficient for a lot of businesses out there. Once reaching a certain level of complexity, I recommend to start looking into IaC.



*Don't feel like you have to know every term or tool mentioned. We'll get to everything just in time.*

Once the resources have been provisioned, the next step is the configuration. **Configuration management** is about maintaining a desired state. Its purpose is to configure the already provisioned servers to a specific state that can support the application at hand. Think application dependencies, NGINX configuration, the PostgreSQL `pg_hba.conf` file, or firewalld settings.

We'll learn how to configure various components to the desired state over the course of this book. To achieve that, we'll automate this process using Bash and SSH connection. Some well-known tools in this space, including Ansible, are based on the same idea of executing shell commands and scripts over an SSH connection. They add a little bit more ceremony to make a larger multi-node deployment more manageable.

Then there are tools like Chef or Puppet that, by default, expect a master node that acts as a hub for configuration data (holding the desired state). The other nodes run lightweight agents that can fetch the newly applied state and configure the node accordingly. This approach might be faster when a higher number of nodes are considered, but the former feels closer to doing things yourself with Bash.



*Some tools can support more than one stage of deployment. Sometimes they overlap in functionality. Ansible started as a configuration management tool but will happily provision your servers in the IaC fashion today.*

The last step is the **application deployment**. Its simplest form is most likely copying files over SFTP. In the case of Ruby and Python applications, it is perhaps uploading the files with `scp` and restarting the application server. But for a typical web application today, there are few more tasks to solve, from updating runtime dependencies to pre-compiling front-end assets.

As you can imagine, we might need a short script or take advantage of a specialized tool like Capistrano to handle all that. Also, deployment is not

always forward. We need to be able to roll back to a previous version of the application if things go haywire. We could send a different revision over or do what Capistrano does; keep older versions around and symlink to the desired one. On the following pages, we'll configure the virtual server to deploy static sites with a short wrapper around `scp` and full-featured web applications with `git` (a deployment popularized by Heroku):

```
$ SERVER=165.227.158.200 server/setup.sh
$ git push production v1.0.0
```

Then there are **Linux containers** that significantly change how we build the dependencies and approach the application delivery. A container can be similarly pushed to a server as code, or built from this code on the server side. However, most use containers in combination with a centralized repository that hosts all releases. Application deployment is then pulling the new version and replacing the previous running container without any build steps. We'll see how to build containers from scratch, and how they change the delivery of applications.

Automation of running containerized services is **container orchestration**. It's partly provisioning, partly deployment. This approach was originally invented for cloud providers and the consolidation of many services in corporations. It usually involves moving and scaling container workloads across many nodes. Today, many developer teams adopt tools like Kubernetes and OpenShift with the availability of the tools' *managed* offerings.

In the end of the book, we'll discuss the next steps you can take with your application deployment.