Community Experience Distilled

Harness the power of DevOps to boost your skill set and make your IT organization perform better

Practical DevOps

Joakim Verona

DevOps is a practical field that focuses on delivering business value as efficiently as possible. DevOps encompasses all the flows from code through testing environments to production environments. It stresses the cooperation between different roles, and how they can work together more closely, as the roots of the word imply—Development and Operations.

After a quick refresher to DevOps and Continuous Delivery, we quickly move on to looking at how DevOps affects architecture. You'll create a sample enterprise Java application that you'll continue to work with through the remaining chapters. Following this, we explore various code storage and build server options. You will then learn how to perform code testing with a few tools and deploy your test successfully. Next, you will learn how to monitor code for any anomalies and make sure it's running properly. Finally, you will discover how to handle logs and keep track of the issues that affect processes.

Who this book is written for

This book is aimed at developers and system administrators who wish to take on larger responsibilities and understand how the infrastructure that builds today's enterprises works. This book is also great for operations personnel who would like to better support developers. You do not need to have any previous knowledge of DevOps.

What you will learn from this book

- Appreciate the merits of DevOps and Continuous Delivery and see how DevOps supports the agile process
- Understand how all the systems fit together to form a larger whole
- Set up and familiarize yourself with all the tools you need to be efficient with DevOps
- Design an application that is suitable for continuous deployment systems with Devops in mind
- Store and manage your code effectively using different options such as Git, Gerrit, and GitLab
- Configure a job to build a sample CRUD application
- Test the code using automated regression testing with Jenkins Selenum
- Deploy your code using tools such as Puppet, Ansible, PalletOps, Chef, and Vagrant
- Monitor the health of your code with Nagios, Munin, and Graphite
- Explore the workings of Trac—a tool used for issue tracking

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 10 'The Internet of Things and DevOps'
- A synopsis of the book’s content
- More information on Practical DevOps
About the Author

Joakim Verona is a consultant with a specialty in Continuous Delivery and DevOps. He has worked with all aspects of systems development since 1994. He has actively contributed as the lead implementer of complex multilayered systems such as web systems, multimedia systems, and mixed software/hardware systems. His wide-ranging technical interests led him to the emerging field of DevOps in 2004, where he has stayed ever since.

Joakim completed his masters in computer science at Linköping Institute of Technology. He has also worked as a consultant in a wide range of assignments in various industries, such as banking and finance, telecom, industrial engineering, press and publishing, and game development. He is also interested in the Agile field and is a certified Scrum master, Scrum product owner, and Java professional.
The field of DevOps has become popular and commonplace in recent years. It has become so pervasive that it is easy to forget that before 2008, when Patrick Debois organized the first DevOpsDays conference, hardly anyone had even heard the word.

What does DevOps, which is a portmanteau of the words "developers" and "operations", mean though, and why does it generate such tremendous excitement? The mission of this book is to answer this seemingly simple question.

The short answer is that DevOps aims to bring different communities, such as the developer and operations communities, together to become a more efficient whole.

This is also reflected in the book. It explores many tools that are useful in DevOps work, and tools that bring people closer together are always preferred to those that create artificial borders between people. The processes we use for software development are also tools, so it is also natural to include aspects of the various Agile schools of thought as they relate to DevOps.

The book also aims, as the title suggests, to be practical.

Let's begin our journey in the land of DevOps!

What this book covers

Chapter 1, Introduction to DevOps and Continuous Delivery, deals with the background of DevOps and sets the scene for how DevOps fits in the wider world of Agile systems development.

Chapter 2, A View from Orbit, will help you understand how all the systems we use in DevOps fit together, forming a larger whole.
Chapter 3, How DevOps Affects Architecture, describes aspects of software architecture and what they mean to us while working with our DevOps glasses on.

Chapter 4, Everything is Code, explains how everything is code and you need somewhere to store it. The organization's source code management system is that place.

Chapter 5, Building the Code, explains how you need systems to build your code. They are described in this chapter.

Chapter 6, Testing the Code, shows you that if you are going to release your code early and often, you must be confident of its quality. Therefore, you need automated regression testing.

Chapter 7, Deploying the Code, shows how, when the code has been built and tested, you need to deploy it to your servers so that your customers can use the newly developed features.

Chapter 8, Monitoring the Code, covers how the code is safely deployed to your servers with the deployment solution of your choice; you need to watch over it to make sure it’s running properly.

Chapter 9, Issue Tracking, looks at systems used to handle development workflows within an organization, such as issue tracking software. Such systems are an important aid when implementing Agile processes.

Chapter 10, The Internet of Things and DevOps, describes how DevOps can assist us in the emerging field of the Internet of Things.
In the previous chapter, we explored some of the many different tool options, such as issue trackers, available to us to help us manage workflows.

This chapter will be forward looking: how can DevOps assist us in the emerging field of the Internet of Things?

The Internet of Things, or IoT for short, presents challenges for DevOps. We will explore what these challenges are.

Since IoT is not a clearly defined term, we will begin with some background.

**Introducing the IoT and DevOps**

The phrase *Internet of Things* was coined in the late 1990s, allegedly by the British entrepreneur Kevin Ashton, while he was working with RFID technology. Kevin became interested in using RFID to manage supply chains while working at Proctor and Gamble.

**RFID**, or *Radio Frequency ID*, is the technology behind the little tags you wear on your key chain and use to open doors, for instance. RFID tags are an example of interesting things that can, indirectly in this case, be connected to the Internet. RFID tags are not limited to opening doors, of course, and the form factor does not have to be limited to a tag on a key chain.
An RFID tag contains a small chip, about 2 millimeters squared, and a coil. When placed near a reader, the coil is charged with electricity by induction, and the chip is given power long enough for it to transmit a unique identifier to the reader's hardware. The reader, in turn, sends the tag's identification string to a server that decides, for example, whether it is going to open the lock associated with the reader or not. The server is likely connected to the Internet so that the identifiers associated with the RFID tags can be added or deleted remotely depending on changing circumstances regarding who has access to the locked door. Several different systems work in symbiosis to achieve the desired outcome.

Other interesting IoT technologies include passive QR codes that can be scanned by a camera and provide information to a system. The newer Bluetooth low energy technology provides intelligent active sensors. Such sensors can operate up to a year on a lithium cell.

Below, two RFID tags for operating locks are depicted:
The term "Internet of Things" is pretty vague. When is it a "thing" and when is it a computer? Or is it a little bit of both?

Let's take a small computer, such as the Raspberry Pi; it is a System on a Chip (SoC), mounted on a board the size of a credit card. It's small, but is still a full computer and powerful enough to run Java application servers, web servers, and Puppet agents.

Compared to traditional computers, IoT devices are constrained in various ways. Often, they are embedded devices that are placed in difficult-to-access, constrained locations. This, then, would seem to be the defining characteristic of the IoT from a DevOps point of view: the devices might be constrained in different regards. We cannot use every technique that we use on servers or desktop machines. The constraints could be limited memory, processing power, or access, to name a few.

Here are some example types of IoT devices, which you have probably either encountered already or soon will:

- **Smartwatches**: Today, a smartwatch can have a Bluetooth and Wi-Fi connection and automatically detect available upgrades. It can download new firmware and upgrade itself on user interaction. There are many smartwatches available today, from the robust Pebble to Android Wear and Apple Watch. There already are smartwatches with mobile network connections as well, and eventually, they will become commonplace.

- **Keyboards**: A keyboard can have upgradeable firmware. A keyboard is actually a good example of an IoT device, in a sense. It provides many sensors. It can run software that provides sensor readouts to more powerful machines that in the end are Internet-connected. There are fully programmable keyboards with open firmware, such as the Ergodox, which interfaces an Arduino-style board to a keyboard matrix (http://ergodox.org/).

- **Home automation systems**: These are Internet-connected and can be controlled with smartphones or desktop computers. The Telldus TellStick is an example of such a device that lets you control remote power relays via an Internet-connected device. There are many other similar systems.

- **Wireless surveillance cameras**: These can be monitored with a smartphone interface. There are many providers for different market segments, such as Axis Communications.
The Internet of Things and DevOps

- **Biometric sensors**: Such sensors include fitness sensors, for example, pulse meters, body scales, and accelerometers placed on the body. The Withings body scale measures biometrics and uploads it to a server and allows you to read statistics through a website. There are accelerometers in smartwatches and phones as well that keep track of your movements.

- **Bluetooth-based key finders**: You can activate these from your smartphone if you lose your keys.

- **Car computers**: These do everything from handling media and navigation to management of physical control systems in the car, such as locks, windows, and doors.

- **Audio and video systems**: These include entertainment systems, such as networked audio players, and video streaming hardware, such as Google Chromecast and Apple TV.

- **Smartphones**: The ubiquitous smartphones are really small computers with 3G or 4G modems and Wi-Fi that connects them to the wider Internet.

- **Wi-Fi capable computers embedded in memory cards**: These make it possible to convert an existing DSLR camera into a Wi-Fi capable camera that can automatically upload images to a server.

- **Network routers in homes or offices**: These are in fact, small servers that often can be upgraded remotely from the ISP side.

- **Networked printers**: These are pretty intelligent these days and can interact with cloud services for easier printing from many different devices.

The list can, of course, go on, and all of these devices are readily available and used in many households today.

It is interesting to note that many of the devices mentioned are very similar from a technical viewpoint, even if they are used for completely different purposes. Most smartphones use variants of the ARM CPU architecture, which can be licensed across manufacturers together with a few variants of peripheral chips. MIPS processors are popular among router hardware vendors and Atmel RISC processors among embedded hardware implementers, for example.

The basic ARM chips are available to developers and hobbyists alike for easy prototyping. The Raspberry Pi is ARM-based and can be used as a prototyping device for professional use. Arduino similarly makes Atmel-architecture devices available for easy prototyping of hardware.
This makes a perfect storm of IoT innovation possible:

- The devices are cheap and easy to acquire even in small runs.
- The devices are simple to develop for and get prototyping platforms.
- The development environments are similar, which makes them easier to learn. A lot of the time, the GNU Compiler Collection, or GCC, is used.
- There is a lot of support available in forums, mailing lists, documentation, and so on.

For powerful devices such as the Raspberry Pi, we can use the same methods and practices that we use for servers. Pi devices can be servers, just less powerful than a traditional server. For IoT devices, agentless deployment systems are a better fit than systems that require an agent.

Tinier devices, such as the Atmel embedded CPUs used in Arduinos, are more constrained. Typically, you compile new firmware and deploy them to the device during reboot, when special bootstrap loader code runs. The device then connects to the host via USB.

During development, one can automate the upload of firmware by connecting a separate device that resets the original device and puts it into loader mode. This might work during development, but it's not cost-effective in a real deployment scenario since it affects costs. These are the types of problems that might affect DevOps when working with the IoT. In the development environment, we might be able to use, more or less, the methods that we are used to from developing server applications, perhaps with some extra hardware. From a quality assurance perspective, though, there is a risk involved in deploying on hardware that is different to that used in testing.

**The future of the IoT according to the market**

According to the research group Gartner, there will be approximately 6.4 billion things connected to the Internet at the end of 2016: a 30 percent increase since 2015. Further away, at the end of the year 2020, there will be an estimated 21 billion Internet-connected devices. The consumer market will be responsible for the greatest number of devices, and enterprises for the greatest spending.
The next generation of wireless networks, which are tentatively called 5G mobile networks, is expected to reach the market in 2020, which isn't too far away at the time of writing. The upcoming mobile networks will have capabilities suitable for IoT devices and even be way ahead of today's 4G networks.

Some of the projected abilities of the new mobile network standards include:

- The ability to connect many more devices than what is possible with today's networks. This will enable the deployment of massive sensor networks with hundreds of thousands of sensors.
- Data rates of several tens of megabits per second for tens of thousand of users. In an office environment, every node will be offered a capacity of one gigabit per second.
- Very low latencies, enabling real-time interactive applications.

Regardless of how the 5G initiatives turn out in practice, it is safe to assume that mobile networks will evolve quickly, and many more devices will be connected directly to the Internet.

Let's look at what some industry giants have to say:

- We begin with Ericsson, a leading Swedish provider of mobile network hardware. The following is a quote from Ericsson's IoT website:

  "Ericsson and the Internet of Things

  More than 40 percent of global mobile traffic runs on Ericsson networks. We are one of the industry's most innovative companies with over 35,000 granted patents, and we are driving the evolution of the IoT by lowering thresholds for businesses to create new solutions powered by modern communications technology; by breaking barriers between industries; and by connecting companies, people and society. Going forward, we see two major opportunity areas:

  **Transforming industries:** The IoT is becoming a key factor in one sector after another, enabling new types of services and applications, altering business models and creating new marketplaces.

  **Evolving operator roles:** In the new IoT paradigm, we see three different viable roles that operators can take. An operator's choice on which roles to pursue will depend on factors like history, ambition, market preconditions and their future outlook."
Ericsson also contributed to a nice visual dictionary of IoT concepts, which they refer to as the "comic book":
http://www.alexandra.dk/uk/services/Publications/Documents/IoT_Comic_Book.pdf

- The networking giant Cisco estimates that the Internet of Things will consist of 50 billion devices connected to the Internet by 2020. Cisco prefer to use the term "Internet of Everything" rather than Internet of Things. The company envisions many interesting applications not previously mentioned here. Some of these are:
  - Smart trash bins, where embedded sensors allow the remote sensing of how full a bin is. The logistics of waste management can then be improved.
  - Parking meters that change rates based on the demand.
  - Clothes that notify the user if he or she gets sick.

All of these systems can be connected and managed together.

- IBM estimate, somewhat more conservatively perhaps, 26 billion devices connected to the Internet by 2020. They provide some examples of applications that are already deployed or in the works:
  - Improved logistics at car manufacturers' using wide sensor net deployments
  - Smarter logistics in health care situations in hospitals
  - Again, smarter logistics and more accurate predictions regarding train travel

Different market players have slightly different predictions for the coming years. At any rate, it seems clear that the Internet of Things will grow nearly exponentially in the coming years. Many new and exciting applications and opportunities will present themselves. It's clear that many fields will be affected by this growth, including, naturally, the field of DevOps.

**Machine-to-machine communication**

Many of the IoT devices will primarily connect to other machines. As a scale comparison, let's assume every human has a smartphone. That's about 5 billion devices. The IoT will eventually contain at least 10 times more devices – 50 billion devices. When this will happen differs a bit among the predictions, but we will get there in the end.
One of the driving forces of this growth is machine-to-machine communication.

Factories are already moving to a greater degree of automation, and this tendency will only grow as more and more options become available. Logistics inside factories can increase greatly with wide sensor net deployments and big data analytics for continuously improving processes.

Modern cars use processors for nearly every possible function, from lights and dashboard functions to window lifts. The next step will be connecting cars to the Internet and, eventually, self-driving cars that will communicate all sensor data to centralized coordinating servers.

Many forms of travel can benefit from the IoT.

**IoT deployment affects software architecture**

An IoT network consists of many devices, which might not be at the same firmware revision. Upgrades might be spread out in time because the hardware might not be physically available and so on. This makes compatibility at the interface level important. Since small networked sensors might be memory and processor constrained, versioned binary protocols or simple REST protocols may be preferred. Versioned protocols are also useful in order to allow things with different hardware revisions to communicate at different versioned end points.

Massive sensor deployments can benefit from less talkative protocols and layered message queuing architectures to handle events asynchronously.

**IoT deployment security**

Security is a difficult subject, and having lots of devices that are Internet-connected rather than on a private network does not make the situation easier. Many consumer hardware devices, such as routers, have interfaces that are intended to be used for upgrades but are also easy to exploit for crackers. A legitimate service facility thus becomes a backdoor. Increasing the available surface increases the number of potential attack vectors.
Perhaps you recognize some of these anti-patterns from development:

- A developer leaves a way in the code to enable him or her to later submit code that will be evaluated in the server application context. The idea is that you as a developer don't really know what kind of hot fixes will be necessary and whether an operator will be available when the fix needs to be deployed. So why not leave a "backdoor" in the code so that we can deploy code directly if needed? There are many problems here, of course. The developers don't feel that the usual agreed-upon deployment process is efficient enough, and as an end result, crackers could probably easily figure out how to use the backdoor as well. This anti-pattern is more common than one might think, and the only real remedy is code review.

Leaving open doors for crackers is never good, and you can imagine the calamity if it was possible to exploit a backdoor in a self-driving car or a heat plant, for instance.

- Unintended exploits, such as SQL injection, mostly occur because developers might not be aware of the problem.

The remedy is having knowledge about the issue and coding in such a way as to avoid the problem.

Okay, but what about DevOps and the IoT again?

Let's take a step back. So far, we have discussed the basics of the Internet of Things, which is basically our ordinary Internet but with many more nodes than what we might normally consider possible. We have also seen that in the next couple of years, the number of devices with Internet capability in some form or another will keep on growing exponentially. Much of this growth will be in the machine-to-machine parts of the Internet.

But is DevOps, with its focus on fast deliveries, really the right fit for large networks of critical embedded devices?

The classic counterexamples would be DevOps in a nuclear facility or in medical equipment such as pacemakers. But just making faster releases isn't the core idea of DevOps. It's to make faster, more correct releases by bringing people working with different disciplines closer together.
This means bringing production-like testing environments closer to the developers and the people working with them closer together as well.

Described like this, it really does seem like DevOps can be of use for traditionally conservative industries.

The challenges shouldn't be underestimated though:

- The life cycles of embedded hardware devices can be longer than those of traditional client-server computers. Consumers can't be expected to upgrade during every product cycle. Likewise, industrial equipment might be deployed in places that make them expensive to change.
- There are more modes of failure for IoT devices than desktop computers. This makes testing harder.
- In industrial and corporate sectors, traceability and auditability are important. This is the same as for deployments on servers, for instance, but there are many more IoT endpoints than there are servers.
- In traditional DevOps, we can work with small changesets and deploy them to a subset of our users. If the change somehow doesn't work, we can make a fix and redeploy. If a web page renders badly for a known subset of our users and can be fixed quickly, there is only a small potential risk involved. On the other hand, if even a single IoT device controlling something such as a door or an industrial robot fails, the consequences can be devastating.

There are great challenges for DevOps in the IoT field, but the alternatives aren't necessarily better. DevOps is also a toolbox, and you always need to think about whether the tool you pick out of the box really is the right one for the job at hand.

We can still use many of the tools in the DevOps toolbox; we just need to make sure we are doing the right thing and not just implementing ideas without understanding them.

Here are some suggestions:

- Failures and fast turnarounds are okay as long as you are in your testing lab
- Make sure your testing lab is production-like
- Don't just have the latest versions in your lab; accommodate older versions as well
A hands-on lab with an IoT device for DevOps

So far, we have mostly discussed in the abstract sense about DevOps and the IoT and the future of the IoT.

To get a feel for what we can do in hands-on terms, let’s make a simple IoT device that connects to a Jenkins server and presents a build status display. This way, we get to try out an IoT device and combine it with our DevOps focus!

The status display will just present a blinking LED in the event that a build fails. The project is simple but can be expanded upon by the creative reader. The IoT device selected for this exercise is quite versatile and can do much more than make an LED blink!

The project will help to illustrate some of the possibilities as well as challenges of the Internet of Things.

The NodeMCU Amica is a small programmable device based on the ESP8266 chip from Espressif. Apart from the base ESP8266 chip, the Amica board has added features that make development easier.

Here are some of the specifications of the design:

- There is a 32-bit RISC CPU, the Tensilica Xtensa LX106, running at 80 MHZ.
- It has a Wi-Fi chip that will allow it to connect to our network and our Jenkins server.
- The NodeMCU Amica board has a USB socket to program the firmware and connect a power adapter. The ESP8266 chip needs a USB-to-serial adapter to be connected to the USB interface, and this is provided on the NodeMCU board.
- The board has several input/output ports that can be connected to some kind of hardware to visualize the build status. Initially, we will keep it simple and just use the onboard LED that is connected to one of the ports on the device.
- The NodeMCU contains default firmware that allows it to be programmed in the Lua language. Lua is a high-level language that allows for rapid prototyping. Incidentally, it is popular in game programming, which might offer a hint about Lua’s efficiency.
The device is fairly cheap, considering the many features it offers:

There are many options to source the NodeMCU Amica, both from electronics hobbyist stores and Internet resellers.

While the NodeMCU is not difficult to source, the project is fairly simple from a hardware point of view and might also be undertaken with an Arduino or a Raspberry Pi in practice if those devices turn out to be simpler to gain access to.

Here are some hints for getting started with the NodeMCU:

- The NodeMCU contains firmware that provides an interactive Lua interpreter that can be accessed over a serial port. You can develop code directly over the serial line. Install serial communication software on your development machine. There are many options, such as Minicom for Linux and Putty for Windows.
- Use the serial settings 9600 baud, eight bits, no parity, and one stop bit. This is usually abbreviated to 9600 8N1.
- Now that we have a serial terminal connection, connect the NodeMCU to a USB port, switch to the terminal, and verify that you get a prompt in the terminal window.
If you are using Minicom, the window will look like this:

![Minicom window](image)

Before starting with the code, depending on how your particular NodeMCU was set up at the factory, it might be required to burn a firmware image to the device. If you get a prompt in the previous step, you don't need to burn the firmware image. You might then want to do it later if you require more features in your image.

To burn a new firmware image, if needed, first download it from the firmware source repository releases link. The releases are provided at [https://github.com/nodemcu/nodemcu-firmware/releases](https://github.com/nodemcu/nodemcu-firmware/releases).

Here is an example `wget` command to download the firmware. The released firmware versions are available in integer and float flavor depending on your needs when it comes to mathematical functions. The integer-based firmware versions are normally sufficient for embedded applications:

```
wget https://github.com/nodemcu/nodemcu-firmware/releases/download/0.9.6-dev_20150704/nodemcu_integer_0.9.6-dev_20150704.bin
```
You can also build the firmware image directly from the sources on GitHub locally on your development machine, or you can use an online build service that builds a firmware for you according to your own specifications.

The online build service is at http://nodemcu-build.com/. It is worth checking out. If nothing else, the build statistics graphs are quite intriguing.

Now that you have acquired a suitable firmware file, you need to install a firmware burning utility so that the firmware image file can be uploaded to the NodeMCU:

git clone https://github.com/themadinventor/esptool.git

Follow the installation instructions in the repository's README file.

If you'd rather not do the system-wide installation suggested in the README, you can install the pyserial dependency from your distribution and run the utility from the git-clone directory.

Here's an example command to install the pyserial dependency:

sudo dnf install pyserial

The actual firmware upload command takes a while to complete, but a progress bar is displayed so that you know what is going on.

The following command line is an example of how to upload the 0.9.6 firmware that was current at the time of writing:

sudo python ./esptool.py --port /dev/ttyUSB0 write_flash 0x00000 nodemcu_integer_0.9.6-dev_20150704.bin

If you get gibberish on the serial console while connecting the NodeMCU, you might need to provide additional arguments to the firmware burn command:

sudo esptool.py --port=/dev/ttyUSB0 write_flash 0x0 nodemcu_integer_0.9.6-dev_20150704.bin -fs 32m -fm dio -ff 40m

The esptool command also has some other functionalities that can be used to validate the setup:

sudo ./esptool.py read_mac
Connecting...
MAC: 18:fe:34:00:d7:21
sudo ./esptool.py flash_id
Connecting...
Manufacturer: e0
Device: 4016

After uploading the firmware, reset the NodeMCU.

At this point, you should have a serial terminal with the NodeMCU greeting prompt. You achieve this state either using the factory-provided NodeMCU firmware or uploading a new firmware version to the device.

Now, let's try some "hello world" style exercises to begin with.

Initially, we will just blink the LED that is connected to the GPIO pin 0 of the NodeMCU Amica board. If you have another type of board, you need to figure out whether it has an LED and to which input/output pin it is connected in case it does. You can, of course, also wire an LED yourself.

Note that some variants of the board have the LED wired to the GPIO pin 3 rather than pin 0 as is assumed here.

You can either upload the program as a file to your NodeMCU if your terminal software allows it, or you can type in the code directly into the terminal.

The documentation for the NodeMCU library is available at http://www.nodemcu.com/docs/ and provides many examples of usage of the functions.

You can first try to light the LED:

```python
gpio.write(0, gpio.LOW)  -- turn led on
```

Then, turn off the LED using the following:

```python
gpio.write(0, gpio.HIGH) -- turn led off
```

Now, you can loop the statements with some delays interspersed:

```python
while 1 do                     -- loop forever
gpio.write(0, gpio.HIGH) -- turn led off
tmr.delay(1000000)       -- wait one second
gpio.write(0, gpio.LOW)  -- turn led on
tmr.delay(1000000)       -- wait one second
end
```
At this point, you should be able to verify a basic working setup. Typing code directly into a terminal is somewhat primitive though.

There are many different development environments for the NodeMCU that improve the development experience.

I have a penchant for the Emacs editor and have used the NodeMCU Emacs mode. This mode, NodeMCU-mode, can be downloaded from GitHub. Emacs has an inbuilt facility to make serial connections. The reader should, of course, use the environment he or she feels most comfortable with.

We need some additional hints before being able to complete the lab.

To connect to a wireless network, use the following:

```lua
wifi.setmode(wifi.STATION)
wifi.sta.config("SSID","password")
```

**SSID** and **password** need to be replaced with the appropriate strings for your network.

If the NodeMCU connects properly to your wireless network, this command will print the IP address it acquired from the network's dhcpd server:

```lua
print(wifi.sta.getip())
```

This snippet will connect to the HTTP server at www.nodemcu.com and print a return code:

```lua
conn=net.createConnection(net.TCP, false)
conn:on("receive", function(conn, pl) print(pl) end)
conn:connect(80,"121.41.33.127")
conn:send("GET / HTTP/1.1
Host: www.nodemcu.com
Connection: keep-alive
Accept: */*
"
.."Connection: keep-alive"
"Accept: */*
"
You might also need a timer function. This example prints **hello world** every 1000 ms:

```lua
tmr.alarm(1,1000, 1, function()
   print("hello world")
end )
```
Here, Lua's functional paradigm shows through since we are declaring an anonymous function and send it as an argument to the timer function. The anonymous function will be called every 1000 milliseconds, which is every second.

To stop the timer, you can type:

    tmr.stop(1)

Now, you should have all the bits and pieces to complete the labs on your own. If you get stuck, you can refer to the code in the book's source code bundle. Happy hacking!

**Summary**

In this final chapter, we learned about the emerging field of the Internet of Things and how it affects DevOps. Apart from an overview of the IoT, we also made a hardware device that connects to a build server and presents a build status.

The idea of going from the abstract to the concrete with practical examples and then back again to the abstract has been a running theme in this book.

In *Chapter 1, Introduction to DevOps and Continuous Delivery*, we learned about the background of DevOps and its origin in the world of Agile development.

In *Chapter 2, A View from Orbit*, we studied different aspects of a Continuous Delivery pipeline.

*Chapter 3, How DevOps Affects Architecture*, delved into the field of software architecture and how the ideas of DevOps might affect it.

In *Chapter 4, Everything is Code*, we explored how a development organization can choose to handle its vital asset—source code.

*Chapter 5, Building the Code*, introduced the concept of build systems, such as Make and Jenkins. We explored their role in a Continuous Delivery pipeline.

After we have built the code, we need to test it. This is essential for executing effective, trouble-free releases. We had a look at some of the testing options available in *Chapter 6, Testing the Code*. 
In Chapter 7, Deploying the Code, we explored the many options available to finally deploy our built and tested code to servers.

When we have our code running, we need to keep it running. Chapter 8, Monitoring the Code, examined the ways in which we can ensure our code is running happily.

Chapter 9, Issue Tracking, dealt with some of the many different issue trackers available that can help us deal with the complexities of keeping track of development flows.

This is the last chapter in this book, and it has been a long journey!

I hope you enjoyed the trip as much as I have, and I wish you success in your further explorations in the vast field of DevOps!
Where to buy this book

You can buy Practical DevOps from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.