A Practical Guide to Coding for Platform-as-a-Service

Programming for

PaaS

O'REILLY®

Lucas Carlson

Programming for PaaS

Lucas Carlson

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

For Yoscelina, My Love,

Thank you for every nano-amount of support. I honestly do not think this book would be here if it were not for you and your encouragement. I love you for everything.

—Lucas Carlson

Special Upgrade Offer

If you purchased this ebook directly from oreilly.com, you have the following benefits:

- DRM-free ebooks—use your ebooks across devices without restrictions or limitations
- Multiple formats—use on your laptop, tablet, or phone
- Lifetime access, with free updates
- Dropbox syncing—your files, anywhere

If you purchased this ebook from another retailer, you can upgrade your ebook to take advantage of a
these benefits for just $4.99. Click here to access your ebook upgrade.

Please note that upgrade offers are not available from sample content.

Preface

Programming Is Hard

Programming is a hard job. Deceivingly so. At first you write code, and it works, and you are happy. Then it has bugs and you spend hours, days, even weeks trying to find, fix, and resolve bugs and edge cases. Then when you have everything perfectly programmed and just when you thought the job couldn't get harder, you have to go deploy your code. *vim apache.conf. vim my.cnf. vim /etc/hosts. iptables*. Just when you thought you were a programmer, all of a sudden you get waste deep in systen administration and wonder how you got there.

If there is one thing that programmers are good at, it is being productively lazy. When a programmer does the same thing over and over again, eventually he thinks: can't my computer do this for me? Around 2005, enough programmers in the world had edited *apache.conf* files that something dramatically changed. A few brilliant programmers decided they didn't want to do it any more.

Out of this came two paradigms that have forever changed the landscape of deploying applications in the world: DevOps and PaaS. DevOps's response to *apache.conf* editing says: I can write code templates (called recipes or cookbooks) that do system administration for me. PaaS's response to *apache.conf* editing says: I can write a program that manages system administration for me. Many great books have been written about DevOps—like *Puppet Types and Providers* by Dan Bode and Na Liu or *Test-Driven Infrastructure with Chef* by Stephen Nelson-Smith—but few books have been written about PaaS.

PaaS is great because you get the benefits of dedicated hosting (like each app running in its own process and load balanced to scale) with the ease of shared hosting (you don't do any configuration management, the PaaS does it for you). But those benefits come at a cost. You have to write code tha works with the PaaS.

Writing Code That Works on PaaS

This topic has not been written about a lot: *which programming patterns work well on PaaS and whic anti-patterns no longer work in a PaaS environment*? This is the entire theme of this book. Although the popularity of PaaS has grown exponentially with millions of developers worldwide having alread adopted it and millions more starting to learn about it right now, not a lot of work has been written to help guide developers on how to successfully incorporate PaaS best practices into their coding processes.

Specifically, one of the biggest challenges facing many developers working in businesses today is ho to move legacy code and older applications into a PaaS paradigm. There have been few resources to help guide people through this challenge and hopefully this book will be a first step in the right direction to providing the dialogue.

Audience

This book is aimed at programmers, developers, engineers, and architects who want to know more about Platform-as-a-Service.

You do not need to be a programmer to find value in this book. In fact, if you are trying to convince your boss to let you use PaaS inside your company, you may want to give your boss a copy of this book. Alternatively, you can find resources for talking to your boss about PaaS, both the pros and cons, in Chapter 8. This will show you have thought through the problem from both sides of the table and have an informed opinion, not just a passing fashion.

In some of the technical chapters, I even provide code samples. Since PaaS works with many programming languages, I provided simple programming samples in various programming languages including PHP, Ruby, Node.js, Java, and Objective-C. We do not go deep in any one language, but rather stay high level on various ones in hopes that you are familiar with one or two and can read through the others.

The Structure of This Book

If you are an architect or technical manager, or are simply new to PaaS, the first three chapters are very important to understand the context for which PaaS has entered the technical landscape. These chapters explain what the cloud is (Chapter 1), what PaaS is (Chapter 2), and different kinds of PaaS technologies and their relative strengths and weaknesses (Chapter 3).

If you already know about the history of PaaS or have used a PaaS, you can skim through the first three chapters and dig in for the next three chapters around Chapters 4, 5, or 6. These chapters are the heart of this book, providing real life tools, techniques, and programming patterns that will help you stay away from the biggest pitfalls of PaaS and not waste any time on programming anti-patterns.

Chapter 7 is an important chapter for everyone to understand. Services like database and caching services or email services contain some of the biggest gotchas in the PaaS world. If you are not careful, this is the place you can fail most quickly when adopting PaaS.

The next two chapters go back to a higher level of understanding. In Chapter 8, there is discussion around the appropriateness of adopting PaaS at all, including the strengths and weaknesses of PaaS in general. Understanding whether PaaS is a good fit for the problem you are tackling is critical. In Chapter 9, the discussion centers around where PaaS is going, some industry trends, and thoughts around Open Source movements in the PaaS world.

The last chapter is a great place to reference any technologies available around PaaS. Chapter 10 should have a bookmark sticking out of it, because you will be flipping to it to find ideas for service providers and technologies of all types (PaaS, IaaS, SaaS, and helpful programming libraries).

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

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

`Constant width`

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

`Constant width bold`

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

`Constant width italic`

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

> **TIP**
>
> This icon signifies a tip, suggestion, or general note.

> **CAUTION**
>
> This icon indicates a warning or caution.

Using Code Examples

Supplemental material (code examples, exercises, etc.) is available for download at http://examples.oreilly.com/9781449334901-files/.

This book is here to help you get your job done. In general, if example code is offered with this book you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Programming for PaaS* by Lucas Carlson (O'Reilly). Copyright 2013 Lucas Carlson and Doug Baldwin, 978-1-449-33490-1."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at permissions@oreilly.com.

Safari® Books Online

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

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

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training video and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Pres Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us onlin

How to Contact Us

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

O'Reilly Media, Inc.

1005 Gravenstein Highway North

Sebastopol, CA 95472

800-998-9938 (in the United States or Canada)

707-829-0515 (international or local)

707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at http://oreil.ly/programming-paas.

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

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

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

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

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

Acknowledgments

First, I would like to thank Doug Baldwin for all his assistance putting this book together; it could never have been done without him.

If it were not for Meghan Blanchette's patience and persistence and Mike Loukides believing in me, would never have started or finished this book.

Thank you to Kara Ebrahim, Meghan Connolly, and the whole O’Reilly crew for making this possibl

This book would be appallingly low quality were it not for our technical reviewers, who spotted bugs problems, and conceptual errors: John Purrier, Alex Parkinson, Larry Hitchon, Andrei Matei, Chad Keck, and Troy Howard.

To my wife, my son, my daughter, my dog, my mom, my dad, my brother, and all my family and family-in-law, thank you for your everlasting support and love.

Finally, to the programmers and inventors who created PaaS and the companies that supported them, thank you for making all of our lives easier.

Chapter 1. The Cloud for Developers

One day, not long ago, Jason Gendron had an idea.

What if he could create a community of Twitter friends, so that instead of just following each other, users might actually have bidirectional communication? Jason, a Chicago-based developer, wrote some code, registered the domain name twitclub.com, and deployed it on a dedicated server. Success was immediate. In a few months, over 80,000 people were using the service. But with success came challenges—the kind of challenges you always say you will be glad to have if only you succeed.

With 80,000 users, Jason was spending half his time handling operations and half his time doing development. He spent less time innovating and more time running the actual app. Before long, hackers compromised his self-configured servers. The hackers sent out terabytes of data, leaving him with an enormous bill. The net result: he was devoting most of his time to battling with servers and not enough to writing code.

Only a few months later, Jason turned to Platform-as-a-Service (PaaS), which allowed him to outsource all the maintenance responsibilities, from software updates to security patches. The benefi were immediately clear. Suddenly he was able to stop thinking about the operations side of his idea, letting his PaaS provider deal with it. That enabled him to spend 100% of his time innovating. Soon h was able to actually quit his day job and devote all his time to his startup, bootstrapping it into a profitable company.

PaaS changed Jason's life, and it can change your life too. It can let you spend more time coding and less time managing servers.

Jason's plight is a familiar one, and its solution—deploying services on PaaS—is one that holds enormous promise, pointing the way toward a future in which cloud-based innovation is drastically easier and much less expensive.

The Developer's Plight

Developers are everywhere. They work in small businesses, in agencies, in enterprises, in startups. A developers are facing the same challenge today: dealing with operations that are coupled with development. The problems may look different depending on your working environment, but the central issue is there nevertheless.

As an example, let's look at the traditional waterfall development process. Typically, the developer works on code and gets it running in a dev/test environment. Then he "throws it over the IT wall," at which point the operations people spend a few weeks to a month getting it quality assured and deployed, creating a huge delay in getting it into production mode. Timelines are delayed. Product testing is delayed. Ultimately, and perhaps most costly, innovation slows down.

Velocity, or the lack thereof, becomes an issue especially with social and mobile applications. You

might have a marketing campaign that needs to go up in a hurry and may only last for a few weeks. Going through the typical process of throwing it over the wall could delay you a critical amount of time, especially if there are troubles with the app and you need to make modifications, or if there is simply not enough velocity for today's social market campaigns.

On the other side of the spectrum are developers in small businesses who are simply trying to get the jobs done, and individual developers—innovators like Jason Gendron—who are trying to come up with ideas for the next Instagram or Facebook. PaaS helps them solve the issue of velocity while at th same time providing significant savings by letting them focus on coding.

Looking to the future, these savings promise to have a profound—and very positive—impact on the creation of startups.

What the Cloud Has Done for Innovation

The cloud has transformed how developers create software, speeding up the way developers work around the world, from venture-backed startups all the way to large Fortune 100 corporations.

Modern developers are being asked to do more with less. They are given tight budgets and tighter deadlines to accomplish miraculous feats of development. Cost and convenience have led to widespread adoption of the cloud. The cloud has been the mechanism that lets enterprise developers bypass IT departments and allows entrepreneurs to have access to effectively unlimited-sized data centers without any up-front hardware purchases.

Figure 1-1 tracks venture capital deals over a 10-year period beginning in 2001. It shows normalized data for the total number of deals versus total deal size. As the graph illustrates, the two sets of numbers tracked very closely for several years. Significantly, that changed in 2006, the year Infrastructure-as-a-Service (IaaS) went mainstream when Amazon introduced Amazon Web Services with EC2 and S3. At that point, the cloud decoupled the two trajectories, progressively driving down the deal size and driving up the number of deals.

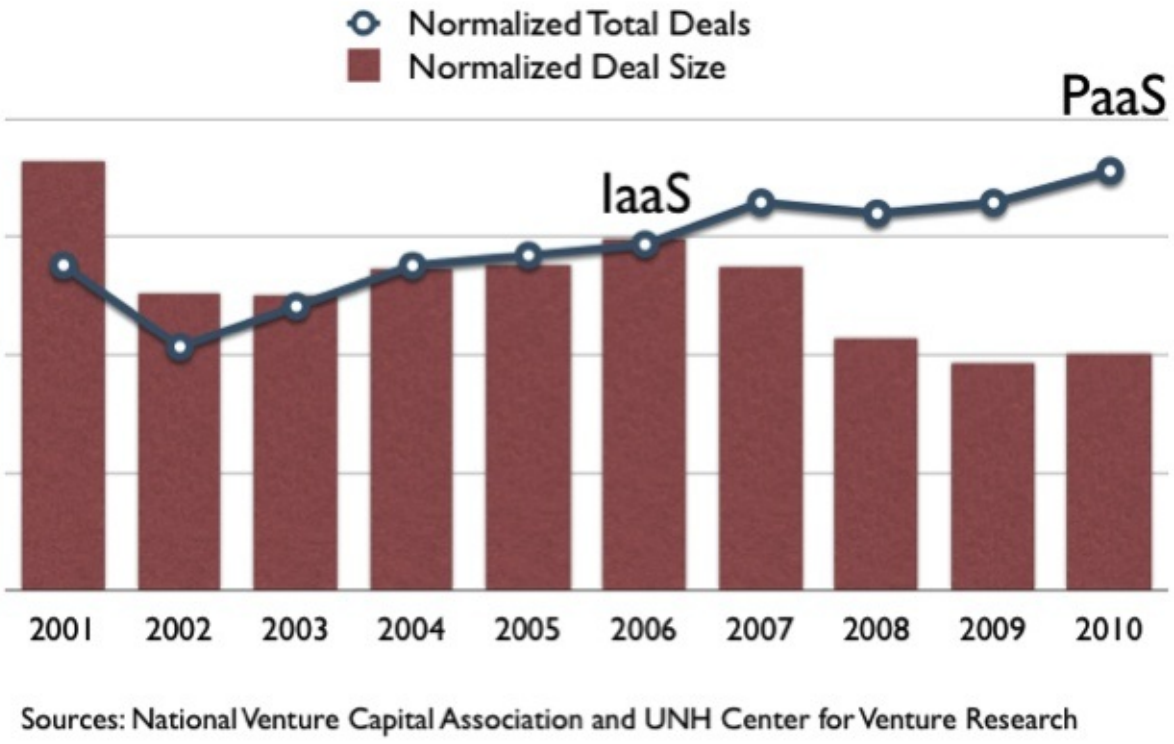

Figure 1-1. Total deal size vs. normalized total deals, 2001–2010

What does this mean? What are the implications?

For the first time in history, there is a decoupling of the total number of deals versus the deal size. By 2010, venture capitalists had to pay only half as much money to fund new companies as they used to. These high-tech companies no longer needed as much capital. They did not need to build out data centers anymore; they could rely on the cloud. That is a key benefit of cloud adoption in venture capital deals.

Through cloud technology like IaaS and PaaS, the 50% cost savings are going to translate into all sor of business: not just startups and VC-backed companies, but in the mid-market and enterprise worlds as well.

Infrastructure-as-a-Service gets rid of half the problem—the issue of buying and managing data centers. The second half of the problem is managing application operations. Decoupling operations from development is the second and true promise of the cloud, and it's one that Platform-as-a-Servic is uniquely poised to deliver.

The Cloud: A Brief History for Programmers

What is the cloud? It is a loaded term and overly used.

Is the cloud just Dropbox? Is it the iPhone? Is it just Gmail?

Maybe for some people these myriad examples are the cloud, but not for a developer.

For a developer, the cloud is a set of foundational technologies built on top of each other to enable new ways of building and running technology. If you can't build a new kind of foundational

technology on another foundational technology, it is not the cloud.

Many apps and SaaS are built on foundational cloud technologies. Dropbox and Gmail are SaaS apps built on foundational cloud technology. But they themselves are not cloud technologies.

The 1990s saw the rise of data centers. Third-party companies would put servers into a room and rent them out. This was less expensive than previous constraints, when companies had to buy their own servers and data centers.

With the rise of centralized data servers came virtualization, which represented the first step into the cloud. With virtualization, data centers could consolidate into massive servers subdivided into smaller ("virtualized") servers. VMware and Microsoft were among the early companies to create software that was critical to the development of virtualization.

Introducing APIs

In 2006, on top of virtualization came the next big step into the cloud: the application programming interface (API). APIs added a sophisticated layer of automation, enabling you to control, start, stop, and create new virtual machines through simple and real-time commands. First created by companies such as Amazon with Amazon Web Services, APIs paved the way for Infrastructure-as-a-Service, which heralded a lot of what we now consider cloud standards.

At this point in the history of the cloud, it was so easy to spawn a multitude of virtual machines that managing servers became a headache. With unlimited servers at your fingertips, how do you take care of managing them all? That is where DevOps came into play.

Along Comes DevOps

DevOps became a mainstream force around 2010. It arose from the need of developers to do their job faster. They were tired of waiting for operations to deploy their code. They were tired of not having tools to easily manage services. They were tired of having to do it all manually. Out of these frustrations, programmers moved into the cloud and became DevOps gurus. They built systems to manage and maintain infrastructure and to interact with servers in a programmatic fashion.

Important DevOps tools like Chef and Puppet were introduced, providing systems to manage thousands of servers, upgrade code, replace code, replace servers, deploy new servers into a template and make modifications, all of which had been very labor intensive and difficult from a developer's point of view. We even saw the rise of automated DevOps tools like RightScale and ScaleXtreme.

DevOps offers developers the most control out of all the cloud tools. The trade-off is that you still need to spend time building and managing operations; if things go wrong, you are still the go-to person. And so, DevOps is not the ultimate answer to the hopes of developers. Here's why.

As a developer, you will most likely spend time writing code that works with Chef and Puppet, while your ultimate goal is probably to spend less time managing operations. If that is your goal, Platform-as-a-Service can handle operational tasks for you today, freeing up even more of your time. With PaaS, you do not have to write the Chef cookbooks or the Puppet scripts to manage servers. You can spend more of your time writing the code that interacts with your users.

DevOps technologies were—and remain—essential for application lifecycle management tools, which we'll discuss in a moment. Application lifecycle management would not be possible without DevOps so DevOps is not going to go away. DevOps is a core foundational technology, and there will always be a need for it in a cloud world.

DevOps tools are also a big part of how any Platform-as-a-Service runs under the covers. For platforms such as Heroku, EngineYard, AppEngine, and AppFog, DevOps tools are essential behind-the-scenes tools.

The Arrival of Application Lifecycle Management

Thanks to DevOps, one can now manage thousands of machines easily, but one still sees the world through the glasses of infrastructure. Applications care more about services (e.g., MySQL and MongoDB) than the details (e.g., what version of *libxml* is installed across virtual machines). Managing services and applications remains outside the grasp of DevOps and even IaaS. The next layer of cloud technology is application lifecycle management.

App lifecycle management tools are typified by technologies like Cloud Foundry, OpenShift, and Cloudify. They are not yet really PaaS (though they sometimes claim to be), because they still need an operator to run them (hence not really removing the big pain of running operations yourself). However, they do provide a cornerstone to PaaS technology. These tools know how to start, stop, and deploy applications. They often know how to run and manage some of your services, like MySQL, as well.

App lifecycle management tools handle applications holistically, managing applications across many different servers, so an application can run on hundreds or even thousands of them. Traditionally, IaaS and DevOps tools have had trouble thinking in terms of applications—of the needs, resources, and services that span hundreds of servers. App lifecycle management understands applications, treats them from a services point of view, and knows how to manage, scale, and maintain them.

In many cases, you can run app lifecycle management tools on your own laptop. Cloud Foundry has a Micro VM that can do a proof of concept very quickly. OpenShift and Cloudify have similar tools. Taking app lifecycle management to the next level and building out a production-ready version of the same tool can be daunting; it often takes teams of 5–10 operations people and involves managing dozens or hundreds of machines, even for small production installations.

The benefits of app lifecycle management are great, though. Take, for example, a Drupal website that is going to be hit by massive traffic overnight. In this example, you'll need to go from a single server to 1,000 servers to handle this huge influx of new users.

Before app lifecycle management, you would need to engage a team to manually figure out how to incorporate hundreds of new servers into the process. You would need to deploy the right code and take a very hands-on approach. It required significant human intervention and it needed to scale with human resources.

Even with DevOps, this can be a daunting task. How do you ensure that the servers that Puppet creates are all working the way you expect them to? How do you maintain these servers? How can you see in real time how the application loads are affecting each server? These are only some of the

shortcomings of DevOps tools.

An app lifecycle management tool such as Cloud Foundry changes all this. You can tell it that your application needs 1,000 more instances, and Cloud Foundry will do all the plumbing. It makes the changes necessary to run those applications across your servers. It greatly simplifies the process.

But there's a hitch. You still need people watching and monitoring the app lifecycle tool itself. You don't get rid of operations; you are simply shifting your attention from an individual application to t running of your lifecycle tool.

In our example, prior to the advent of app lifecycle management tools, when you needed to scale up the number of servers the development team would interact directly with the operations team to ensure success. Sometimes the same team, or even the same person, would be responsible for both, b the roles would be separate. The operations people would be in charge of manually figuring out how hook all the servers together.

Now an app lifecycle management tool can do all that for you. It knows about the application, it knows how to execute the code, it knows how to add the servers and instances to the application. But the app lifecycle management tool itself needs operations. People are required to operate the tool. Th difference is that it is an abstract tool into which you can put anything, any application. As a developer, it does not matter if the operations people running the tool are in-house and sitting right next to you, or if they are hundreds of miles away in a place like Portland, Oregon.

Thus we enter the realm of NoOps and Platform-as-a-Service.

The Next-Generation Cloud with Platform-as-a-Service

The cloud has been instrumental in moving us away from a paradigm that involved significant expenses, not only in terms of buying servers, but also of running those systems, with all the personn involved.

In the past, any startup would have had a similar experience. You would need to hire people to handl your core competency, and then spend millions of dollars to handle all of the data center and management pieces.

The central concept and the main benefit of the cloud, beginning with Infrastructure-as-a-Service, is reduce the cost of buying data centers. You do not need to buy data centers anymore. But you still need operations people to run the servers that you are now renting.

NoOps fully removes the need for operations and development people to work hand in hand. Operational needs are still being fulfilled, but they are being accomplished independently of any individual application. NoOps refers to the idea of outsourcing operations roles and providing a way for developers to get their jobs done faster. Developers don't have to wait for another team to deploy their code, and the systems that automate those processes for developers operate seamlessly.

NoOps is a controversial term, because some people read it as "No Ops," suggesting that operations are no longer relevant. On the contrary, however, operations are more relevant and important today than ever. The intent behind the term NoOps is to highlight that from the developer's perspective, yo no longer interact as directly with the operating guts of running the application. It's the same as how "NoSQL" doesn't mean that SQL is no longer relevant, but rather that there is another way of

approaching data storage and retrieval where a developer doesn't interact with SQL.

Platform-as-a-Service outsources operations. It is not getting rid of operations, but decoupling them from development, so that you can get your job as a developer done easier and faster.

The original iterations of managed Platform-as-a-Service, which included systems such as Force.com and the Google App Engine, were very constrained. They required you to develop your code against their APIs, so that they only worked within their contexts. Their Platform-as-a-Service features were accessed only when you tied them directly to their systems. They promised that you could access special data and that you could take advantage of features like auto-scaling—but doing so could be difficult.

There were two downsides to early PaaS. The first was that you needed to learn a new set of APIs in order to program against them, so there was a learning curve just to get acclimated. The second downside involved portability. If you were to program against the Google App Engine, for example, it would be difficult to move your application and try it out on a different platform. There were upsides as well, though such as the extractive system data and the services that they provided.

Then new players began to arrive on the scene, and with their arrival came a movement toward platforms like Heroku and EngineYard. They broke the mold of saying that you needed to code against APIs. They said, in effect, "We are going to take your application as is. You don't have to make any changes that work against proprietary APIs. You can run this code just as easily on your own hardware as you could on our systems, and you aren't going to have to change your code to make that happen." For developers, this was a revolutionary moment in the history of the cloud.

Initially, in order to provide their innovative services, these PaaS providers limited the language technology. Early users of Heroku and EngineYard, for example, could only pick Ruby. So while the upside was that you didn't have to program against APIs, the downside was a restriction on what languages you could pick.

PaaS quickly evolved with the help of next-generation PaaS companies like AppFog and dotCloud, which were no longer tied to a single language. Today, many larger PaaS companies, and even older ones like Heroku and EngineYard, support many programming languages. This represents a significant transition from restrictive platforms to those that are more open.

Some PaaS companies are still tied to single programming languages, however, claiming that there is more benefit in having deeper domain knowledge for a single language.

A popular trend in PaaS with companies like AppFog is multi-infrastructure PaaS. This allows you to run apps in many different infrastructures at the same time, even operated by completely different companies. For the first time in history, you can easily run your app on Amazon Web Services and Rackspace at the same time. If Amazon Web Services goes down, Rackspace can act as a disaster recovery program that keeps applications up and running on a separate system. This has been a dream for most programmers that PaaS can uniquely fulfill.

The Core of the Cloud

For developers, "the cloud" can be a loaded term with many different interpretations. How does the

cloud help the developer get her job done better and faster? It can be difficult to clearly determine what is simply an application and what will actually change your life.

To some people, the cloud means Gmail, Dropbox, and similar services. But these are applications *built on* the cloud. They do not change a developer's life. What really changes a developer's life are the core fundamental cloud technologies, the essential technologies that form the backbone of the cloud.

The foundational cloud technologies are virtualization, infrastructure APIs, DevOps, app lifecycle management tools, and NoOps. They all build on top of each other and combine to enable the next generation of technologies. Each one is necessary for the others. For example, you can't have infrastructure APIs for virtualization without virtualization itself.

As a developer, you can interact with any one of these foundational technologies and extract great benefits. For example, you can interact with virtualization directly. Many DevOps technologies do so managing KVM or Xen directly, usually to virtualize different operating systems and test application in those systems. You can test an application to determine whether it is a software app, a web app, or mobile app; with this foundational technology, you can virtualize all of these environments.

Using APIs for virtualization, many developers build on Amazon Web Services and similar OpenStack APIs in order to get their jobs done better and faster. It enables them to spawn servers and manage processes and packages with great speed.

But the problem is that you, Mr. or Ms. Developer, are still the one on call when the servers go down at 4 a.m. And the servers inevitably do go down at 4 a.m. Even with virtualization. Even with infrastructure APIs. Even with Amazon Web Services. Even with Cloud Foundry.

As a developer, half of the problem is getting the resources you need; that is what Infrastructure-as-a Service solves. The other half of the problem is running and managing your application; that is what Platform-as-a-Service solves.

From a developer's point of view, you can utilize any of these core technologies successfully. The higher up the stack you move, the more time you can spend writing your own code. As a developer, you can spend time at the IaaS level: you'll have more control over some of the lower features, those closer to the infrastructure. The trade-off is that you'll need to spend more time in VMs and less time on the code that you are writing for your users.

The higher up the stack you move in the cloud and the closer you get to PaaS, the more time you can spend innovating. You'll have the time to be like Jason Gendron and improve your product, to pivot and try different things with your users, to figure out how to build the next Google or Facebook, rather than worrying how to keep the servers up at 4 a.m.

Managed Platforms versus Productized Platforms

Our discussion of PaaS has centered on managed or "public cloud" PaaS, in which a developer or company outsources maintenance responsibilities to the PaaS provider. A productized or "private cloud" PaaS offers a different set of attributes.

In a productized platform, you are utilizing app lifecycle management tools and Platform-as-a-Servic

tools on your own hardware with your own resources. The benefit is that your operations team is in control. They can incorporate the platform's tools in a way that works with many of your application because they are familiar with how they work. Another advantage: your operations people can reuse much of the hardware in which you have invested and determine how to get all aspects of your syster to work together well.

A managed PaaS provider, like Heroku or AppEngine, is in charge of running and maintaining operations for you. As we saw earlier, one of the biggest benefits of Platform-as-a-Service is that it provides the ability to decouple development from operations. Once development is decoupled, the operations do not need to be in house anymore. And if they do not need to be in house, you have to as yourself, "Is it cheaper to run them in house or is there an economy of scale to be had with an outside provider?"

A managed platform takes PaaS technology and provides it as a service for your company in real tim without you needing to worry about service level agreements. Uptime is guaranteed, so that when things go wrong at 4 a.m., it is the public provider's job to fix it, not yours.

There are various productized platforms to choose from. There are open source app lifecycle management tools like Cloud Foundry or OpenShift that can be run on-premises, and there are more commercial tools like Cloudify and Stackato that can be licensed. Some companies, like AppFog, hav both a public cloud PaaS offering and a private cloud, on-premises license of the same software.

Running app lifecycle management tools like Cloud Foundry on your own can be very hard. Production-quality service for these tools requires dozens of individual components that interact with each other. It can be a complex matter making sure that these components interact well, that they are healthy and managed, and that if something goes wrong, they are replaced. Those are the kind of issues handled by a managed platform—issues that will need to be dealt with manually if you are trying to incorporate those tools into your own processes.

The Cloud's Promise (or Hype)

From a developer's point of view, part of the challenge of thriving in this new landscape is determining whether or not the cloud is all hype.

Is Gmail going to make a dramatic difference in the life of an individual developer, a corporation, or an agency? Probably not. It might make an incremental improvement, but not a drastic one. However understanding how to best utilize foundational cloud tools like DevOps and PaaS within the operation of a modern business, whether you are starting one or running one, and figuring out how to leverage cloud technology to do your job faster and more cheaply than before is not hype at all. On the contrary, it is how high-tech companies are going to be built and operated in the future. It is reality. Without a doubt, we are rapidly headed *en masse* to the cloud.

When technology is built on technology, it creates a feedback loop that grows exponentially. When you consider, for example, that the maturing process that led from Morse code to the advent of the telephone took more than 40 years, the current rate of technological maturation is nothing short of astounding. That's one of the lessons of Moore's law on transistors, which observes that the number transistors on integrated circuits doubles approximately every two years. In high-tech companies and

in the cloud, innovations are happening more rapidly than ever. The timeline for new foundational technologies being built, coming to market, and maturing is shortening quickly, with major overhaul on entire industries happening in timelines as short as a few years. In recent history, it took around a decade for virtualization to gain widespread adoption. IaaS has matured nearly as much in only five years. PaaS is likely to be adopted widely in a mere one to two years.

As a developer, learning to adroitly adapt to these technologies is key for growing your career in this modern age.

The Cloud in Five Years

PaaS is maturing quickly, but it is still not perfect for every application. Will the next Twitter or the next Facebook be built on PaaS? The answer today is “not yet,” but most companies are at least starting to consider the advantages of moving to PaaS.

As of 2013, PaaS has not been proven at scale yet, much like Ruby on Rails hadn’t been in 2006. A few large companies have shown success with PaaS (e.g., Groupon using EngineYard). Once a few more large success stories have shown the potential of PaaS at large scale, we are likely to see a mass adoption shortly after.

The other factor for mass adoption is the maturity of PaaS behind the firewalls for large corporations Heroku and EngineYard are fantastic public cloud solutions, but large companies have already invested in large numbers of servers that are not likely to be retired very soon. Being able to run large applications in PaaS on existing hardware will go a long way to making mass adoption a reality.

If you look ahead 5 to 10 years, PaaS will be a cornerstone technology. We are going to witness a new generation of high-tech companies being built on the tools of the cloud. We are going to see billion-dollar companies being built solely on PaaS technology.

The Promise Fulfilled

As we saw at the beginning of this chapter, PaaS literally changed Jason Gendron’s life situation. It enabled him to focus on his core talents, writing code and running his business, while it lowered his cost of innovation. It gave him the time and the server power to help turn TwitClub into a successful venture.

By removing the need for an IaaS provider, by managing the nitty gritty of day-to-day operations, by handling glitches and crashes, PaaS is changing the lives of millions of other developers.

Whether you are inside an agency dealing with hundreds of clients, inside an enterprise with thousands of users, or working on your own to develop the big idea, PaaS offers you the tools to realize your potential.

It reduces the price of innovation for individual developers. It reduces the price of innovation for venture capitalists looking to invest in the next Google. And it is going to bring the same cost saving into enterprises as enterprise adoption of PaaS becomes ubiquitous.

In the coming chapters, we’ll look more closely at PaaS and managed services, examine how to write

apps for PaaS, and explore a future in which PaaS becomes a major player in cloud technology.

Chapter 2. What Is PaaS?

Developers are migrating to PaaS to get their jobs done faster and better.

Faster: because you spend less time setting up and managing servers, or waiting for someone else to do it.

Better: because PaaS lets you implement best practices without thinking about it.

Developers all have their own unique sets of considerations and challenges. Before writing this book, myself had challenges rooted in a childhood web success that turned into what would have been a crushing failure had I not learned some critical lessons that strongly influence how I write code today

Conjuring a Website

In 1996, my family signed up for an Internet connection and received a shared hosting account tied to our Internet Service Provider. I downloaded Fetch, a free FTP client, and found a Webmonkey tutorial that taught me the basics of HTML. I entered my FTP credentials and was on my way. When I fully realized anyone in the world could see my Hello World web app, I was hooked. A simple “hello” transformed the rest of my life.

I love magic. I find it fascinating how you can prepare for days or even weeks, practicing detailed finger work for a trick that can last a few seconds. I wanted to combine my passion for doing magic tricks with my passion for this new toy called the Web. So I created my first web page, a page for magicians to exchange ideas and tricks. It started out as a simple HTML page and quickly grew into a dynamic PHP website that I called The Conjuring Cabaret.

As the community grew, The Conjuring Cabaret became more and more dynamic, adding content, functionality, and design. Many magicians contributed magic tricks and tips. There were tests that visitors had to pass in order to get in to see those tricks, making sure that only real magicians could access this inner sanctum.

The site grew to the point that it needed its own server. The Conjuring Cabaret was hosted on a dedicated box, and I was thrilled. By 2001, it was among the top magic websites, with hundreds of tricks and thousands of members. I was installing and configuring Apache and MySQL and spent countless nights tuning things to be just right.

Then one day I woke up and the website was not working. I tried to log into the server. No luck. I tried to email the managed server provider and didn’t hear back. I kept emailing them, and a few days later when they finally replied, they said, “Sorry, your server died.”

I said, “What do you mean my server died?”

Like many developers before and since, I had never gotten around to backing up my server. I lost all of the data, all of the magic tricks, and all of the users. Ironically, that server had performed a magic

trick of the highest order: it made everything disappear.

That was the end of my first website.

It was a painful episode and a pivotal moment for me, deeply influencing my professional life. Years later, I was able to realize a dream that came out of that pain when I began to use Platform-as-a-Service and founded a PaaS provider called AppFog.

Early Options for Developers

Episodes of scaling problems and losing data are unfortunately common, mainly because application developers have had only a few options in the last generation. Chief among them were shared web hosting and dedicated web hosting. Now we can add into the mix a powerful, relatively new alternative called Platform-as-a-Service. Before we look more deeply at PaaS, let's examine the first two options.

Shared Web Hosting

Traditionally, shared web hosting has been the easiest way for web developers to get started. Examples include GoDaddy, Yahoo!, Comcast, and many ISPs to this day.

The central concept revolves around an FTP account. You get credentials for your FTP application server. That server hosts thousands, sometimes tens of thousands, of websites. Usually the server is large, but it can very quickly get bogged down. If one or two of those websites start to get popular, even with a powerful server, it might be enough to soak up all of the resources. The result: tens of thousands of websites could become very slow or might not even respond.

The upside to shared hosting is price. Shared hosting is cheap, sometimes even free. Also, it's a hand free account. You don't have to do security patches. You don't have to manage the software. You don't have to know anything about how the software is written. All you need to know is how to write some HTML code; you hand it over, and everything else is handled for you. But because it's so inexpensive, your provider can't afford to handle things well at all times. It can't scale. It can't go beyond its capabilities, but it usually does work for simple situations.

While it's an economy of scale for the hosting provider, it's not a reliable system for putting up a storefront, a complicated website, or even a site that you want your clients to be able to reliably visit to get your contact information.

Another pitfall of shared hosting is security. Your code can be on the same system as over 10,000 other pieces of code. Keep in mind that a normal website can easily get hundreds of security attacks every day. Multiply that by tens of thousands of pieces of code that aren't yours, and you can see the risks involved in running a production site on a shared system.

Despite its disadvantages, developers still find shared web hosting useful to host personal web pages, to put up simple ideas, and to try out new ideas. It's useful for development when you're not sure if you want to invest the kind of money it would take to run your own servers, and when you don't need to scale yet. The trouble is that when you do need to scale, it can become a painful process to move from a shared to a dedicated system.

Dedicated Hosting

Developers, especially web developers, have traditionally used shared web hosting to get started because it's cheap and easy. When they look to graduate beyond that, they often turn to dedicated we hosting. This could be as simple as hosting your own server at home using your Internet connection. But there are several other options that provide varying degrees of service and scalability.

Here is a generalized list of dedicated hosting options, sorted by decreasing order of control (and typically, performance):

- Colocated servers
- Managed servers
- Virtual private servers
- Infrastructure-as-a-Service

Let's now take a look at each of these in more depth.

Colocated servers

With colocation, you usually buy your servers yourself. Then you ship them to a high-bandwidth data center where you pay a monthly fee. The colocation facility gives you Internet access and sometimes will even help debug or restart a server. But in addition to the up-front costs of the machines, you are responsible for maintaining and managing the servers.

Managed servers

The term "managed server" is a bit of a misnomer. In reality, the management of the server can be quite limited. If the RAM becomes corrupt, they will replace it for you. If there are hardware issues, the provider will replace your hardware. But while they replace disks or RAM, they do not replace your data, so it's critical to make sure that you have off-site backups.

There are various benefits to using managed servers. Often you do not have to buy the servers yourself; you lease them from the same provider that is hosting and managing them. They are faster than some of the other dedicated alternatives, as well as more reliable and more robust. Managed servers can and do die, but they usually don't die very quickly. You generally have to wait for a disk failure, which on average could take a year or two. Compare that to the ephemeral servers on Amazo Web Services, which could die in a matter of days or weeks. The downside is that provisioning new servers can easily take anywhere from weeks to a month.

Virtual private servers

Virtual private servers, or VPS, are similar to managed servers, but virtualized. In desktop environments, you may be familiar with Parallels, VirtualBox, and Fusion. For server-side virtualization, the tools of the trade (known as *hypervisors*) include XenServer, KVM, Virtuozzo, Vserver, and Hyper-V.

Virtualization allows large servers with many terabytes of RAM and hundreds of processor cores to b

sample content of Programming for PaaS

- **1,411 QI Facts to Knock You Sideways pdf**
- read The Eagle in the Sand (Eagle, Book 7)
- How We Forgot the Cold War: A Historical Journey across America here
- *The Film Paintings of David Lynch: Challenging Film Theory pdf*
- read online Pigeon English here

- http://drmurphreesnewsletters.com/library/Dietary-Supplements-Pocket-Companion.pdf
- http://deltaphenomics.nl/?library/Cooking-Jewish--532-Great-Recipes-from-the-Rabinowitz-Family.pdf
- http://redbuffalodesign.com/ebooks/Israel-on-the-Appomattox--A-Southern-Experiment-in-Black-Freedom-from-the-1790s-Through-the-Civil-War.pdf
- http://unpluggedtv.com/lib/The-Film-Paintings-of-David-Lynch--Challenging-Film-Theory.pdf
- http://toko-gumilar.com/books/Dead-Men-s-Dust--Joe-Hunter--Book-1-.pdf