PROGRAMMING AND THE CLOUD,

PREPARING THE DEVELOPER FOR TOMORROW

The Cloud is confusing… well it can be, and that's where CloudU™ comes in. CloudU is a comprehensive Cloud Computing training and education curriculum developed by industry analyst Ben Kepes. Whether you read a single whitepaper, watch a dozen webinars, or go all in and earn the CloudU Certificate, you'll learn a lot, gain new skills and boost your resume. Enroll in CloudU today at *www.rackspacecloduniversity.com*

Diversity Limited

Sponsored By: rackspace® HOSTING

Table of Contents

Introduction

The advent of Cloud Computing has removed infrastructure as a barrier to rapid and massive scaling of applications. Whereas in the past the need for physical hardware created a drag upon the entire scaling process, the ease of use and accessibility of Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) has made it possible for a developer to create an application one day and have it utilized by hundreds of thousands of users the next.

This paradigm shift in terms of scale, geographic spread and velocity has changed the development process markedly – no longer do developers have the luxury of creating an application and scaling that application in a controlled and generally sedate manner. Instead the velocity of today means that they need to think about scale and speed from the outset.

At the same time, modern web applications are architected in an entirely different manner with traffic to and from the application via Application Programming Interfaces (APIs) amounting for a significant proportion of total traffic. This change in the ways data is being created, stored and consumed by an application has also changed the demands on developers.

We contend that the developer roles of the future will mirror the other roles within the enterprise and become more collaborative and project specific. The developer of tomorrow will need to have an understanding of a range of different technologies and be able to build specific combinations of those building blocks for different application requirements.

Similarly, the developer of tomorrow needs to understand a wide variety of different languages, frameworks and application components in order to choose components that will create the most appropriate application stack for their particular needs. At the same time, the developer of tomorrow will have to have the ability to collaborate with others offering different specialties; this combination of broad knowledge and strong collaboration skills will define development in the future.

Finally, the move to a world with a myriad of devices, a plethora of input devices and a rapidly increasing velocity of change not only puts special requirements on languages and frameworks but also upon the people who use them.

What is Cloud?

Cloud Computing is a revolutionary development within IT. This revolution is enabled by a unique combination of technology shifts alongside new business models. As we detailed in a previous CloudU report[1], Cloud Computing is enabling an IT revolution through a number of means:

- Virtualization – The ability to increase computing efficiency
- Democratization of computing – Bringing enterprise scale infrastructure to small and medium businesses
- Scalability and fast provisioning – Bringing web scale IT at a rapid pace
- Commoditization of infrastructure – Enabling IT to focus on the strategic aspects of its role

Importantly, Cloud Computing is enabling more innovation to occur at the developer level than ever before – speed, agility and economics are combining to mean that building a concept product and getting it to market is a relatively painless task. Given this fact, we predict a burgeoning number of developers over the years ahead.

For a deeper background into what Cloud Computing is, we recommend spending some time exploring the content developed for the CloudU Cloud Computing certificate[2]; but with a basic understanding of Cloud Computing out of the way, we will give some guidance into developing for the Cloud era.

Development Democratized by a New Generation of Platforms

While the bulk of this report looks at development of software from a bare building blocks perspective, mention must be made of the variety of platforms that exist to enable businesses, rather than technical users, to create software.

Broadly categorized under the term PaaS, a subject we covered extensively in a previous CloudU post[3], these platforms often allow line of business staff to take core business data and build specific applications around it. As we detailed in the previous report, PaaS can be broadly split into two distinct types:

- A collaborative platform for software development, focused on workflow management regardless of the data source being used for the application. An example of this approach would be Heroku, a PaaS that utilizes the Ruby on Rails development language.

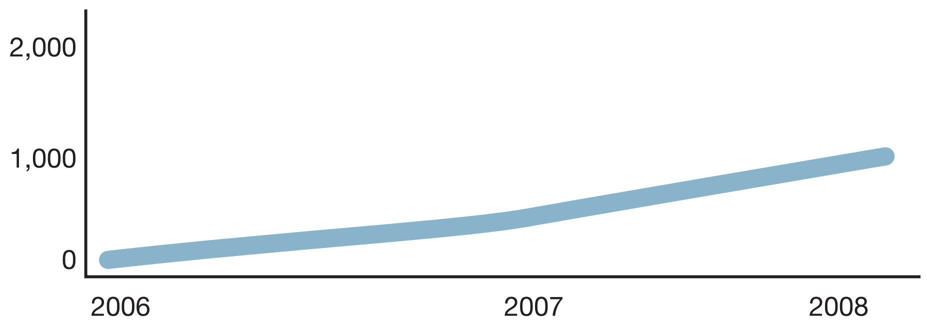
- A platform that allows for the creation of software utilizing proprietary data from an application. This sort of PaaS can be seen as a method to create applications with a common data form or type. An example of this sort of platform would be the Force.com PaaS from Salesforce.com which is used to develop applications that work with the Salesforce.com CRM.

For the purposes of this report, however, we will talk about bare bones software development and will come back to PaaS in a future report.

Front-end and Back-end, The Future of Development

Before looking at specific languages and frameworks of relevance to the modern developer, it is worthwhile setting the scene by explaining the context of modern development. Over the past several years there has been a marked increase in the number and variety of different devices which are reading from, and writing to, applications. We are seeing the end of developers only having to think about writing an application for a couple of different web browsers; instead there is a demand for a vast array of access options. This demand is met by the move away from developers creating their own rigid application front-ends, and a move towards the provisioning of Application Programming Interfaces (APIs) that allow for application access without constraints.

This fact is well illustrated by a graph that John Musser from ProgrammableWeb showed during the 2011 Glue conference. ProgrammableWeb has been tracking the growth of APIs over a number of years, and the graph below demonstrates just how fast the number of APIs that exist is growing. These APIs exist largely because of a demand for application data unimpeded by front-end constraints.



API Growth, Source ProgrammableWeb[4]

Perhaps one of the best examples of a back-end service untethered to any particular front-end is Twitter. Using a full range of APIs, Twitter has enabled a legion of developers to build new and innovative ways of integrating with Twitter's core data and at the same time opened up the ability to capture new data and create a new business model for itself.

Many people predict that 2012 will be an inflection point for development where the growth of mobile use of applications and the rapid rise in the number of connected device types will result in a strong polarization of the development process between those focusing on the back-end and those more interested in the front-end of the application.

Back-end – It's About Scale and Balance

Keeping with the example of Twitter, it is obvious that the developer of tomorrow who wishes to focus primarily on back-end development is going to have to understand the realities of building applications using a number of different component parts and having those parts all work together no matter how much the demand scales.

Back-end developers need to have a broad understanding of the different parts of the application stack (for example database, API servers and legacy systems) and need to deeply understand how different layers of the application stack interact. They also need to think about how that interaction will change over time and need to consider application demands potentially scaling both massively and erratically.

Back-end developers tend to be concerned with the logic tier or the data tier – two concepts we discuss later. However, they also need to think about DevOps or Development Operations. Essentially, DevOps is an emerging discipline that seeks to ensure a collaborative integration of development and operations so that (1) the application is written in such a way as to be efficient to operate and scale and (2) operations meets the particular needs of the application.

DevOps has developed in response to the emerging understanding of the interdependence and importance of both the development and operations disciplines in meeting an organization's goal of rapidly producing software[5]. Writing code that manages the servers and allows for the addition, configuration and deletion of servers based on demand is also a key DevOps role.

This approach can be contrasted with a more traditional development approach where developers created an application and shipped it to an operations team who had the responsibility to deploy and manage the application. A good analogy to explain why a DevOps approach is better is to liken DevOps to a cook who also washes his own dishes. It's fair to say that if a cook knows he will have to clean up after himself, the cooking process will be more efficient than if he simply had others to clean up for him. So too in the development process has it been shown that developers who have an active role in the operation of their application will often write better and more efficient code – hence the rise of DevOps. Modern day back-end developers don't ship code, they deploy it.

Front-end – It's about Interaction, Device Specifics and Beauty

If the back-end part of development is involved with balance and efficiency, the front-end role is much more humanistic. Front-end developers spend much time thinking about how their users will integrate with the application, and they also have to consider that interaction occurring on a range of devices (web, mobile, touch, different form factors etc.).

This is a role that has become increasingly complex. Whereas in the past a front-end developer may have had to consider an interface on a browser and a mobile device, they now have to consider users' experiences across multiple mobile operating systems – Blackberry, Windows Mobile, iOS, Android – along with traditional web browsers. If that wasn't enough, front-end developers now have to consider the application interface on a variety of form factors (smart phones of various sizes, multiple tablet sizes and traditional web). An individual user is today likely to have a range of devices that spans multiple operating systems and form factors – they demand consistency and appropriate user experience across these devices.

For this reason there has been a trend towards the specialization of front-end developers with people tending to focus primarily on one or two operating systems. As we will see, however, the rise of non-proprietary, standards-based interface tools such as HTML5 is tending to lessen this complexity.

Back-end – The Nuts and Bolts

The back-end system upon which developers build applications can be separated into three distinct areas or tiers – the logic tier, the database tier and the infrastructure tier. The highest back-end tier is the logic tier upon which data is processed. In order for this data to be available, the next tier, the database tier, handles the storage and retrieval of data. The database tier separates data storage from the servers upon which the business logic is processed. Finally, the infrastructure tier is where the servers that run both the data tier and the logic tier are maintained. It is at the infrastructure tier that server management and automation occur. Between all of these tiers there are additional processes such as caching and load balancing.

The Logic Tier

The logic tier is made up of two distinct parts – the application language and the application framework. The easiest way to understand the two is to use the analogy of our own everyday languages. The programming language is the words we use every day – essentially our vocabulary. The programming framework on the other hand are the rules, methods and techniques of putting those words together, a combination of grammar, commonly used phrases and the like.

There are a huge variety of different languages, but some of the more well used ones, as detailed by Barton George,[6] include:

- Java/.NET – The incumbent enterprise development languages. Very powerful but relatively difficult to learn and time-consuming to program in.
- C++ — A statically typed, free-form, multi-paradigm, compiled, general-purpose programming language. It is regarded as an intermediate-level language, as it comprises a combination of both high-level and low-level language features.
- Dynamic languages – These are popular for creating web applications since they are both simpler to learn and faster to code in than traditional enterprise standards. This offers a substantial time to market advantage.
  - PHP – A server-side scripting language originally designed for web development to produce dynamic web pages. PHP is infamous for being very quick and easy to get started with but turning into a mess of

"spaghetti code" after years of work and different programmers.

- Perl – One of the original programming languages of the web, Perl emphasizes a very "Unix way" of programming. Perl can be quick and elegant, but, like PHP, can result in a pile of hard to maintain code in the long term. Perl was extremely popular in the first Internet bubble, but it has since taken a back-seat to more popular development languages such as PHP, Java, and Rails.
- Python – Like all dynamic languages, Python emphasizes speed of development and code readability. Python emphases broadness of functionality while at the same time being a proper, object oriented programing language.
- JavaScript – Once a minor language used in web browsers, JavaScript has become a stand-alone language known and used by many programmers. Most web applications will include the use of JavaScript.
- Ruby – Ruby and Python are very similar in ethos, emphasizing fast coding with a more human-readable syntax. Ruby became famous with the rise of Rails in the mid-2000s and is still very popular. Ruby can also be run on top of the Java virtual machine, providing a good bridge to the Java world.
- Scala – A somewhat exotic language, Scala is good for massive scale systems that need to be concurrent. Scala runs on the Java Virtual Machine and Common Language Runtime. Interestingly Twitter moved much of its back-end systems from Ruby to Scala as it sought to handle scaling issues.
- R – A programming language and software environment for statistical computing and graphics.
- Node.js (aka "Node") – Node takes JavaScript, which was originally designed to be used in web browsers, and uses it as a server-side environment. It is intended for writing scalable network programs such as web servers.
- Clojure – A recent dialect of the Lisp programming language, Clojure is good for data intense applications. It runs on the Java Virtual Machine and Common Language Runtime.

The Data Tier

The data tier is the foundation of a database application; it is the tier that manages the data to be created and consumed by the logic tier. At the data tier, developers are concerned with the storage and retrieval of data as well as managing the access to the data from potentially multiple different areas of the logic tier.

At the data tier, developers will be thinking about the makeup of their data both in terms of quantity (the volume of data items and the velocity at which data must be processed) as well as the quality (is it structured data, i.e., able to be organized into rows and columns, or is it unstructured data where there is no comfortable way of structuring the different data points?).

Depending on the requirements of the particular application, developers will either be looking at using one of a variety of relational databases for structured data or a so-called NoSQL database for unstructured data. Some examples of these two database types are as follows:

Relational Databases

- MySQL – The most popular open source Relational Database Management System (RDBMS)
- Drizzle – A version of MySQL that is specifically targeted at the cloud
- PostgreSQL (Postgres) – An object-relational database management system Oracle DB
- SQL Server

NoSQL Databases

- MongoDB – An open source, high-performance database written in C++
- Riak – A NoSQL database/datastore written in Erlang
- Couchbase – A database powered by Apache CouchDB
- Cassandra – A scalable NoSQL database with no single points of failure; distributed and high performance
- Mahout – A scalable machine learning and data mining library

Alongside the different approaches towards databases, developers will often utilize systems for handling the distribution of processing of large data sets across widespread computing resources. MapReduce, enabled on the Hadoop platform, is a software framework that makes it easy to write applications to process large amounts of data. Hadoop is an open source platform that is well suited to manage the processing of large volumes of unstructured data. It manages MapReduce jobs across a large number of individual servers.

There are a number of different Hadoop utilities for different purposes, as Barton George details in a recent blog post[7]. Examples of these include:

- HBase – The Hadoop database that supports structured data storage for large tables; provides real time read/write access to big data
- Hive – A data warehousing solution built on top of Hadoop
- Pig – A platform for analyzing large data that leverages parallel computation
- ZooKeeper – Allows Hadoop administrators to track and coordinate distributed applications
- Oozie – A workflow engine for Hadoop
- Flume – A service designed to collect data and put it into a Hadoop environment
- Whirr – A set of libraries for running cloud services
- Sqoop – A tool designed to transfer data between Hadoop and relational databases
- Hue – A browser-based desktop interface for interacting with Hadoop

The Infrastructure Tier

As mentioned previously, the infrastructure tier is where the servers that actually run the database and logic tiers are created and maintained. At the infrastructure tier, developers will need to think about the operating systems that best fit the choices they make for the other levels (for example, using Linux as the operating system for an application that uses MySQL and PHP).

Developers have a number of tools to aid in the creation, monitoring and management of the infrastructure layers, including:

- enStratus – A Cloud management and governance provider
- Cloud monitoring tools – Help to ensure servers are performing as expected; examples include CloudKick, Uptime, Nagios and Ganglia
- Puppet – A configuration management tool designed to automate the setup and management of infrastructure; a key DevOps tool produced by Puppet Labs
- Chef – A configuration management tool designed to automate the setup and management of infrastructure; a key DevOps tool produced by Opscode, who hosts a cloud-based version of Chef called the Opscode Platform

These three tiers – logic, data and infrastructure – together provide the back-end application; however, it is the integration with the results of the front-end developer's work where end users get to enjoy the benefits of the back-end developer's hard work.

Front-end – Experience, Aesthetics and Context

If back-end development is primarily focused on ensuring a robust, effective and efficient application, it is at the front-end where developers attempt to build a product that interfaces to a wide variety of user types, contexts and devices.

Front-end developers take processed data from the application and present it in a way that is in context to individual user situations. This job is all the more complex in today's paradigm where end users may well be accessing data on a vast variety of device types, screen sizes and input methods. Adding to that complexity is the fact that for some situations developers will be able to use generic approaches across devices (with HTML5 for example), while in others they will need to create custom presentation offerings for different devices (iOS, Android etc.).

To add to the complexity, applications will utilize client-side processing, and hence front-end developers will need to understand the ramifications of a client side approach and the languages that enable this. Client-side processing can be enabled using a number of approaches, including[8]:

- Native client platform such as Android/Java and iOS/ObjectiveC
- Ajax – Asynchronous JavaScript, which provides new methods of using JavaScript and other languages to improve the user experience
- Flash – Adobe's multimedia platform used to add video, animation and interactivity to web pages; becoming less relevant as time goes on
- JavaScript – A language for creating and delivering rich Web applications that can also run across a wide variety of devices
- JQuery – Cross-browser JavaScript library designed to simplify and speed up the client-side scripting of HTML
- Microsoft Silverlight – Microsoft's browser plugin that enables animation, vector graphics and high-definition video playback
- HTML5 and CSS3 – The latest HTML proposed standard combined with the latest proposed standard for CSS; natively supports much of the client-side functionality provided by other frameworks such as Flash and Silverlight

Mobile Presentation

Currently developers are split between those creating so-called "native" applications (applications written entirely to run on one particular mobile operating system) and those who prefer to use generic approaches towards presentation (HTML5 for example).

It should be noted that there is a double pronged move away from creating native applications, first by the use of frameworks enabling the use of HTML5 and CSS to deploy applications to multiple devices (Sencha Touch is an example of one of these frameworks) and secondly from tools that allow a single code base to be deployed to multiple platforms via native applications (PhoneGap being one such example).

Developers we have spoken to tend to prefer either fully native application development or fully generic development via HTML5. Running a single code base across multiple native applications tends to have implications in terms of speed and efficiency.

Don't Forget the Web

As this paper is likely going to be used by individuals planning their career entry, it is important to introduce a degree of balance to the "mobile first" discussion. We firmly believe that the ever-increasing ubiquity of mobile devices, the availability of fast mobile internet and the move away from fixed computers will increase the proportion of traffic that goes through mobile devices.

That said, it is important to not discount the web's ongoing utility. While mobile access to applications is useful and will continue to grow as services like geo-location become more popular, there will continue to be a need to use desktop devices for a subset of functionality – advanced photo editing, intensive research, cataloguing etc.

Entrepreneur and Venture Capitalist Mark Suster wrote an excellent post[9] in which he exhorted developers and startups to follow a strategy of "Web Second, Mobile First." In other words, while mobile is undoubtedly paramount and gains lots of attention, the vast majority of applications should continue to think about standard web interfaces to their products.

Appendix - What Tools are Real Developers Using?

In December 2012, a discussion was started with the CloudU LinkedIn group to ascertain what languages members of the group were using for building Cloud applications[10]. This is an ongoing discussion, but it is worth noting some of the responses from contributors. The over-arching theme of the comments reiterates our contention that no single language or approach is applicable for all situations. Rather, developers need to consider their particular requirements and approach development languages and frameworks as modules with which to build the more appropriate stack for their application.

This point was well made by Robert Taylor[11] who said that:

> "first, since we're discussing applications for the Cloud and not Cloud itself, I would suggest the rule of web application programming languages applies. Namely: Cloud applications are to be used as services by others and the underlying code is irrelevant. That's the beauty of hosted applications/services: the code doesn't matter.
>
> What does matter, however, is sustainability. I would not use Lisp on any project because I am not conversant in it. However, I also would not use Perl, my most comfortable language, on any project that needed more programmers than myself and some friends. Why? Because fewer people today are Perl experts, it would be hard to find those resources in an affordable price range. Yes, programmers are commodities in this sense.
>
> If I were starting a Cloud app project that had the potential of growing into a large development team I would select the language based on the population of programmers in the relevant Cloud community and the predominate language they are comfortable using.
>
> Oh, regarding the features and performance characteristics of various programming languages and frameworks: meh. Time to market is much more vital than a few nanoseconds here or there — especially in Cloud!"

While we agree with this well considered approach, it is informative to hear what developers are using in the wild. Some responses to the discussion include:

Kowsik Guruswamy[12] gave his experience saying that:

> “we use Ruby/Sinatra for the web app, jQuery/underscore.js for the UI, C++ for high-performance traffic generation. We will also be using node.js for certain real-time capabilities. For the web app, I [personally] think it comes down to Python vs. Ruby and we picked the latter. Real-time used to mean C/C++ for high performance but node.js + v8 gives you something very close in terms of performance.”

Jos Uijterwaal[13] had a not too dissimilar approach:

> “Our IDE is JavaScript from front to back (Node.js), with a dash of C++. We traditionally have a JavaScripters team, before launching our IDE we worked on a UI framework for a long time. Node.js enabled us to leverage these skills on the server.”

Subraya Mallya[14] had a slightly different selection:

> “On the UI side it could any of the frameworks like Django, Ruby on Rails, CakePHP, Zend but majority of the heavy lifting like messaging (JMS), search (Lucene, Solr) and high volume transaction processing on the server side is still done in Java (primarily due to its maturity and robustness). For the UI side Python, PHP, Ruby on Rails are equally productive and mature.”

Nic Wise[15] reiterated some of the views of Robert Taylor but also had some specific suggestions:

> “JavaScript has to be the #1 as it’s all the front end, and it’s useful in the back end too. Other than that: ALL OF THEM. The platform, libraries and other services matter more than the language you wire them up with...
> So, for plumbing (memcache, DBs etc) java, .net but mostly c++. For “front end” (not client) ruby, python, java (and scala/groovy) and .net.
> Of course, the granddaddy, and king of most of it, is PHP. Sadly, as it’s a bit of a mess from a language standpoint .
> One big advantage of “cloud” IMO is it’s easy and quick scale out, which allows comparatively inefficient languages (in the extreme, like python and ruby, and

much lesserly* java and .net) to be used for high traffic sites without spending massive amounts on hardware.

Those languages, however, allow for massive developer productivity, and developers writing, or worse, maintaining, code, are orders of magnitude more expensive than another cloud-based server."

Cornelio Tusnea[16] had perhaps a more conservative selection of approaches:

> "We are using .Net (C#) on the server side and JavaScript like everyone else for the client side. C# is a very good and efficient language and in combination with MVC3&Razor it can be highly productive. And it integrates natively with C++ which makes interoperability for performance a breeze.
> As an ex Java developer I think C# is way better than Java but it does not have the same tooling set as Java. On the positive I don't even feel like I need the same tooling set and libraries that I had in Java. I like Ruby on Rails but I would not touch PHP/Python if I'd like to develop a cloud app."

Similarly Kevin Akermanis[17] had some suggestions but emphasized the notion that there is no "best language;" it's very much a case of suitability to a particular purpose:

> "Tabbed quote>Have used JavaScript, HTML5, Ruby on Rails and various other JavaScript libraries for front-end UI. As for back-end business logic - have traditionally used java but that's more because of the product suite I've lived in. On a personal sentiment (and one that seems to be repeated above), there isn't ONE best language - each have their own fit for purpose strengths and weaknesses."

All of these comments reinforce our contention that the options open to developers are wider than ever before and that developers, both front-end and back-end, need to understand the problem they are trying to solve and build a technology stack that best helps to achieve those aims.

Conclusion

When it comes to modern software development, there is no black and white. Developers need to know a number of different languages and frameworks along with a number of different underlying components so that they can tailor a technology stack to the particulars of each application.

The one thing that is clear, however, is that modern developers will be cleanly split between those developing the front-end of applications and those developing the back-end.

Front-end developers will need a keen design and humanistic sense and will need to understand the various technologies that put data at the hands of users no matter where they are or on what device they seek that data.

Back-end developers on the other hand will need to understand the component parts that make an application scalable. As such, they will combine true development disciplines with the rising function of DevOps.

Whichever discipline developers chose to aim for, Cloud Computing means that there is a very exciting future coming for developers.

Acknowledgements

This paper has been written with the help of a number of people involved in Cloud and its various developer-related areas, and we would like to take this opportunity to acknowledge their contribution. In particular we would like to thank:

- Sam Ramji, VP Strategy, Apigee
- Barton George, Director of Marketing, Dell

About Diversity Analysis

Diversity Analysis is a broad spectrum consultancy specializing in SaaS, Cloud Computing and business strategy. Our research focuses on the trends in these areas with emphasis on technology, business strategies, mergers and acquisitions. The extensive experience of our analysts in the field and our close interactions with both vendors and users of these technologies puts us in a unique position to understand their perspectives perfectly and to offer our analysis to match their needs. Our analysts take a deep dive into the latest technological developments in the above mentioned areas. This, in turn, helps our clients stay ahead of the competition by taking advantage of these newer technologies and, also, by understanding any pitfalls they have to avoid.

Our Offerings: We offer both analysis and consultancy in the areas related to SaaS and Cloud Computing. Our focus is on technology, business strategy, mergers and acquisitions. Our methodology is structured as follows:

- Research Alerts
- Research Briefings
- Whitepapers
- Case Studies

We also participate in various conferences and are available for vendor briefings through Telephone and/or Voice Over IP.

About Rackspace

Rackspace® Hosting is the service leader in Cloud Computing and a founder of OpenStack™, an open source Cloud platform. The San Antonio-based company provides Fanatical Support® to its customers across a portfolio of IT services, including Managed Hosting and Cloud Computing. Rackspace has been recognized by Bloomberg BusinessWeek as a Top 100 Performing Technology Company and was featured on Fortune's list of 100 Best Companies to Work For. The company was also positioned in the Leaders Quadrant by Gartner Inc. in the "2010 Magic Quadrant for Cloud Infrastructure as a Service and Web Hosting." For more information, visit www.rackspace.com.

About the Author

Ben Kepes

Ben Kepes is an analyst, an entrepreneur, a commentator and a business adviser. His business interests include a diverse range of industries from manufacturing to property to technology. As a technology commentator, he has a broad presence both in the traditional media and online. Ben covers the convergence of technology, mobile, ubiquity and agility, all enabled by the Cloud. His areas of interest extend to enterprise software, software integration, financial/accounting software, platforms and infrastructure as well as articulating technology simply for everyday users. More information on Ben and Diversity Limited can be found at http://diversity.net.nz

Endnotes

[1] http://broadcast.rackspace.com/hosting_knowledge/whitepapers/Revolution_Not_Evolution-Whitepaper.pdf

[2] http://www.rackspace.com/knowledge_center/cloudu/curriculum

[3] http://broadcast.rackspace.com/hosting_knowledge/whitepapers/Understanding-the-Cloud-Computing-Stack.pdf

[4] http://blog.programmableweb.com/2012/02/06/5000-apis-facebook-google-and-twitter-are-changing-the-web/

[5] http://en.wikipedia.org/wiki/DevOps

[6] http://bartongeorge.net/2012/01/17/web-glossary-part-one-application-tier/

[7] http://bartongeorge.net/2012/01/18/web-glossary-part-two-data-tier

[8] http://en.wikipedia.org/wiki/Web_development>

[9] http://www.bothsidesofthetable.com/2012/18/web-second-mobile-first/

[10] http://lnkd.in/JqB_e7

[11] http://www.linkedin.com/in/rjamestaylor

[12] http://www.linkedin.com/in/kowsik

[13] http://nl.linkedin.com/in/josuijterwaal

[14] http://www.linkedin.com/in/subrayamallya

[15] http://uk.linkedin.com/in/nicwise

[16] http://au.linkedin.com/in/corneliutusnea

[17] http://au.linkedin.com/in/kevakermanis