CHAPTER 1

Java EE and Cloud Computing
This book looks at Java Enterprise Edition (Java EE) and cloud computing and how you can best get them working for you on the Oracle Java Cloud Service. In each chapter, we review the various aspects of Java EE and then walk through how to get those pieces up and running on the Oracle Java Cloud Service. This initial chapter provides background on both Java EE and cloud computing. We survey the benefits and drawbacks of choosing Java EE on the cloud, as compared to traditional Java EE implementations on private servers. We then review solutions provided by various Java PaaS (Platform as a Service) cloud vendors and discuss some of the competing technologies.

Java Editions
Java has been around for a long time. Considering the pace at which technologies tend to get outdated, Java’s 18-year journey has been most remarkable and highlights the capabilities and the staying power of the technology.

Java 1.0 was released in 1995, and back then, Java had no such thing as an enterprise edition. Only in 1999 was the idea of Java editions (Java SE and Java EE) introduced. Fast-forward to today, and we have three editions of Java:

- **Java Standard Edition (Java SE)** Most Java beginners tend to think of Java SE as Java. However, Java SE as such is a software development platform that provides the Java language, the Java Virtual Machine (JVM), and development and deployment tools for building Java applications.

- **Java Enterprise Edition (Java EE)** Java EE is what we encounter the most in this book. It was first introduced in 1999 as J2EE (or Java 2 Platform, Enterprise Edition). J2EE's mission was to enable enterprises to build highly available, secure, reliable, scalable, multitier, distributed applications. Each subsequent version has sought to enhance these capabilities. The Enterprise Edition continued to be known as J2EE until J2EE 1.4, released in 2003. However, the naming convention was changed in 2006, so what would have been J2EE 1.5 became Java EE 5. Java EE 6 was released in December 2009, and Java EE 7 in June 2013.

- **Java Micro Edition (Java ME)** Java ME was the edition created to address the need for a slimmer version of Java that would work well on the hardware constraints of devices such as mobile phones.
Although most people tend to think of the micro edition as the mobile edition, in reality, the micro edition is used not just for mobile phones, but for all kinds of devices, such as television sets, printers, smartcards, and more. Java ME provides an API and a small-footprint Java Virtual Machine (JVM) for running Java applications.

**NOTE**
Oracle has announced that their longer-term strategy is to converge Java ME and Java SE and provide a modularized solution. The project that aims to design and implement the standard module system is known as Project Jigsaw and is expected to be part of Java 9. Although modularization with Java 9 is still some way away, Java SE 8 has introduced compact profiles. Compact profiles are three subset profiles of the full Java SE 8 specification that could be used by applications that do not require the full platform.

### Java EE Applications

Although the official definitions may differ, for all practical purposes, Java EE is the Java platform for building web and distributed applications. It is essentially a set of libraries that provide most of the core functionality you would require while building your application, which in most cases, is a web-based application. Although you will often hear these applications referred to as “enterprise applications,” do not let “enterprise” scare you away from building a Java EE application. The “enterprise” in EE is simply meant to denote an application that offers some mix of security, reliability, speed, scalability, distribution, transaction, and portability. In an age when mainstream applications were standalone, desktop based, and isolated, the denotation of “enterprise” made sense. However, today, almost all software is meant to be online, social, scalable, and, in a sense, “enterprise.”

As shown in Figure 1-1, Java EE sits on top of the basic Java platform (Java SE). Because most developers begin learning Java with Java SE, it is important to note that all the things learned with Java SE will continue to be true with Java EE.

You will build Java EE applications using your knowledge of the Java language and your understanding of the libraries provided by Java EE. While building
these applications, you also need to adhere to certain rules and conventions defined by Java SE and Java EE. With the help of a Java EE application server to work its magic, your application will be up and running in no time.

**Application Servers**
The application server is the workhorse of Java EE. This is the software that implements Java EE and runs a Java EE application that has been developed, as specified by Java EE, and has been deployed on the application server. Application servers have to stick to the Java EE specification to be Java EE compatible. We will talk more about “specification” and “compatible” later in this chapter, but for the time being, you can proceed with the common English meaning of both terms.

Application servers come in all shapes and sizes and are provided by various vendors, both commercial and open source.

**Open Source vs. Commercial**
Although many Java EE application servers charge top dollar, there are also many open-source Java EE application servers. If you are wondering why someone would pay for a commercial version, the answer lies in the additional features, tools, and services that come with the paid version. Paid versions provide one or more add-ons, such as Control Dashboard, 24x7 Support, Priority Bug Fixes and Patches, Additional Caching, and Performance Tuning.

You can certainly get all the standard Java EE functionality with an open-source server, but in the case where you need that little bit extra for your
enterprise applications, you can opt for the commercial versions. GlassFish, Apache Geronimo, Apache TomEE, Caucho Resin, and JBoss (now WildFly) are some of the popular open-source application servers. With many open-source servers, you find that the same vendor also offers a paid commercial version with add-ons.

Oracle WebLogic, SAP NetWeaver, and IBM WebSphere are popular commercial application servers. The commercial servers often come at a significant cost and are also often customized for specific business needs or even bundled as part of other commercial products.

Oracle WebLogic is Oracle’s commercial application server product and the one that runs on Oracle Java Cloud.

**NOTE**
The open-source servers are freely available for download and use. However, even most commercial servers, such as Oracle WebLogic, offer a trial/developer license that will enable you to download and use the server. Oracle introduced an OTN Free Developer License for WebLogic in 2012, which makes it even easier for developers to try out WebLogic.

Application servers vary primarily on the following factors:

- **Licensing** Software licenses are a vast topic. There are many kinds of commercial licenses and many kinds of open-source licenses, so there are times when a server might fulfill all the requirements of an organization and yet not be considered for adoption because of some license terms and conditions.

- **Support services** Many server vendors, especially those offering commercial variants of open-source servers, rely on support services for their revenues. The quality of the support services is often a crucial factor when deciding which server to adopt.

- **Cost** Application server costs vary drastically. Also, you find that each vendor has its own way of pricing a server. Costs can vary based on many factors, such as number of server instances, number of processors, number of users, and more.
6 Java EE Applications on Oracle Java Cloud

■ **Ease of use**  Although some servers are easy to install, use, and manage, with some servers, just getting them up and running might be a tough task.

■ **Reporting and management features**  The richness of the administration UI and its reporting capability vary a lot across servers. Whereas some servers will give you fancy charts, graphs, and timelines, others will have you digging through log files.

■ **Standalone/bundled**  Application servers are often bundled as part of a larger software suite. For example, if a company buys a certain Oracle application suite, they also get WebLogic bundled with it because it is the foundation for all their products.

■ **Disk and memory requirements**  Some servers seem to do a much better job at managing disk and memory requirements. These differences are, at times, only visible when a server is tested with large loads.

■ **Performance**  As with memory management, some servers are simply better at performance. You will find vendors claiming that their server is the fastest based on the results of certain benchmark tests. For example, in 2012, Oracle announced that WebLogic had set a world record for two processor results with an industry-standard benchmark designed to measure the performance of application servers. These results tend to be used to pitch a product as a faster alternative and a better buy for customers.

■ **Backward compatibility**  Although the Java EE specification itself provides for backward compatibility, some vendors go beyond Java EE backward-compatibility requirements and, at times, even support seemingly outdated technologies. This, however, can be an important feature for companies with legacy software setups but still wanting to migrate to newer versions of the application server.

Let’s now look at an application server that is, in a sense, a first among equals: the reference implementation application server.

**Reference Implementation**

The reference implementation is a definitive implementation of a specification and is developed concurrently with the specification. The reference
implementation for Java EE is the GlassFish Server Open Source Edition, which is available for download at www.glassfish.org.

Because the Java EE reference implementation is developed alongside the specification, it not only shows that the specification can be implemented, but also makes the implementation available as soon as the final specification is released. So although GlassFish 3 was the production version for Java EE 6, GlassFish 4 was being developed while the Java EE 7 specification was being finalized. The day Java EE 7 released, GlassFish 4, the reference implementation, was readily available for developers to try out Java EE 7.

Once the final specification is released, most vendors release their own implementations of the latest version of Java EE. Because these implementations have historically taken up to a year or two, developers looking to try out the latest technologies often opt for GlassFish. GlassFish has grown rapidly over the past few years and is today a popular application server that is used in many production applications.

We will now take a closer look at how the Java Community Process (JCP) works, how a specification is finalized, and what it takes for a server to get certified for a Java EE specification. This process is one of the key factors that makes Java EE stand out from competing technologies (both open source and commercial).

**API and Technology Specification**

Java EE isn't one big block of code or a single technology, but a collection of technologies brought together under the banner of Java EE. Java EE is commonly thought of as an end-to-end technology platform that comes with numerous features and the software providing those features. In reality, however, Java EE is simply a set of specifications about things that need to be done and how they need to be done. Although Java EE is all about building software, the Java EE specification, as such, does not include any code. The specification only provides rules, instructions, and guidelines on which the code is to be written.

You can even think of Java EE specifications as a set of PDF files available for download on the Java Community Process (JCP) website (http://jcp.org).

Having said that, the specification is released along with a reference implementation (RI) and the Technology Compatibility Kit (TCK). The RI is a code implementation that proves the specification can actually be implemented. The TCK consists of tests to check the implementation.
Java Community Process (JCP) and Java EE

Java EE, like all Java technologies, is driven by the JCP. Rather than having one organization make all the decisions, the JCP process involves the developer community and industry in the development of Java specifications. An Expert Group, consisting of members of various companies (such as Oracle, Red Hat, IBM, and SAP) and independent Java EE experts, is elected to manage each new iteration of the Java EE specification. This Expert Group is charged with inviting suggestions from other developers and drafting the specification for a Java Specification Request (JSR). Each version of the specification has a corresponding JSR to which you can refer for complete information about the specification.

For example, JSR 316 is the umbrella JSR for Java EE 6; it lists a high-level view of Java EE 6 and documents the technologies that comprise the specification. JSR 342 covers Java EE 7, whereas JSR 244 covers Java EE 5. All Java EE 5, 6, and 7 technologies and JSRs are listed in Appendix A for reference.

The executive committee is responsible for approving the final specification. The vendor companies usually compete on the actual implementation of that specification.

If you download JSR 316 or 342, all you get is a PDF file. How a company implements the specifications of that PDF makes all the difference in the marketplace. As long as the developer develops as per the specification, his code should work fine on all the Java EE–compatible implementations.

So if Java EE is just a PDF file, where is the code? Where is the implementation? You will remember from the previous section that the application server actually implements the specification. So the application server is where the implementation code is. The server vendors are responsible for writing the code to implement the specification.

NOTE

Although there are specific JSRs for the various Java EE versions, please note that one JSR does not provide the specifications for all Java EE technologies. The Java EE JSR refers to various other JSRs for details on the various technologies in Java EE. Do check out the JCP FAQ at https://jcp.org/en/introduction/faq for more about the journey of the JSR from proposal stage to the final release.
Java EE Compatible

The implementation of the Java EE specification is considered compatible only if it clears the compatibility tests for Java EE that are part of the Java EE Compatibility Test Suite (CTS).

All JSRs need to provide the following:

- The specification
- The reference implementation of the specification
- The Technology Compatibility Kit (TCK)

The fact that all certified Java EE servers will run your Java EE application is one of the key features of Java EE. It brings portability to the platform and drastically reduces vendor lock-in, both vital features for large enterprises. Portability here means the option of migrating to a different application server or a different vendor if required.

However, for a customer to be assured of this portability, all the application servers that claim to be Java EE compatible need to have passed a definite set of compatibility tests. Therefore, Oracle supplies a comprehensive Java EE Compatibility Test Suite (CTS) to Java EE licensees. Only if a server passes the tests in the CTS is it certified as Java EE compatible. This certification for a certain version is a guarantee to the end customer that a server supports Java EE as mandated by the specification for which it has been certified.

NOTE

All Java EE–compatible application servers (that is, all application servers that fulfill the Java EE licensing and compatibility testing procedure) are listed on the “Java EE Compatibility” page on the Oracle Technology Network (OTN) website at http://j.mp/JavaEE-Compatibility.

I would like to reiterate that although the various Java EE server vendors agree on the specifications, they are free to innovate on the actual implementation, as long as it does not interfere with the specified behavior.
NOTE

Oracle Java Cloud Service is not certified for any particular version of Java EE, but supports a mix of Java EE 5 and Java EE 6 technologies that Oracle determined was appropriate for enterprise customers.

Refer to Appendix B for a list of implementations compatible with Java EE 5, 6, and 7.

Profiles

Until Java EE 5, you had no choice but to go with a full-blown Java EE application server, which had support for all the technologies that have made it to Java EE over a decade or so. However, the Java community saw that although most Java EE applications were using only a subset of the technologies, they still had to lug the baggage of all Java EE technologies.

So in Java EE 6, the idea of profiles was introduced. Profiles enabled the creation of smaller Java EE subsets based on requirement. Although it was expected that many profiles would emerge, considering the support and maintenance implications, Java EE 6 and Java EE 7 have only one profile, the Web Profile. So with Java EE 6 and Java EE 7, you can choose the full Java EE server or the slimmer Java EE Web Profile Server.

Web Profile

The Web Profile specification lists only 15 technologies as required components. This naturally leads to a slimmer application server that packs more than enough punch for most web applications. The Web Profile excludes several technologies from the full Java EE specification. However, with Enterprise Java Beans (EJBs), the Web Profile introduces a trimmed-down version called EJB Lite. Download size is not a definite indicator of the difference between versions. However, it is worth noting that the full Java EE (English, Windows) version of GlassFish 3.1.2.2 is 53MB, whereas the Web Profile (English, Windows) version is merely 33MB.

Refer to Appendix A for list of Web Profile technologies in Java EE 6 and 7.
Cloud Computing

There has been immense buzz around cloud computing over the past few years. As seen in Figure 1-2, a Google Trends search for “cloud computing” reveals that cloud computing first gained popularity in 2009 and was at its peak in 2010/2011. By 2012/2013, we see the trend maturing past buzz and into real products, action, and adoption.

It’s fair to say that most software developers have at least taken a cursory look at cloud computing. However, the number of actual cloud users is still small, so let’s quickly review the fundamental ideas underlying cloud computing.

Going Around in Circles

Someone building software in the 1980s and then checking back today might be amused to see the fuss about cloud computing. To him, the cloud was always the most obvious way to go; he would have heard of it as time-sharing, client/server architecture, or thin clients. Even ideas such as hosted services, utility computing, and grid computing from a few years back were closely aligned with what we now know as cloud computing. In each case, you have minimal data and processing on the device at the user's end and...
most of the heavy lifting being done on remote hardware that is far more powerful and capable. The computing power and data were supplied to the user’s device on demand.

So what’s different with cloud computing? The primary difference is the universal Internet connectivity. Whereas in the past, networks were limited to a campus or an area, today, each of us can access information from any corner of the globe. Hosting my data or my application server on some remote corner of the planet and relying on Internet connectivity for access was previously impossible; not so with cloud computing.

Cloud computing is also possible today because data centers are highly scalable and because of virtualization. Virtualization, as the name suggests, is primarily about creating virtual machines (VMs) that don’t run on the actual hardware but instead run on top of the operating system that’s actually talking to the hardware. So a single hardware box that has a Linux operating system can have many VMs running on it—some running Windows, some running other flavors of Linux, and more. Each of these VMs could be used by one or many different users.

What Is Cloud Computing?
Cloud computing, in layman terms, stands for renting computing power and data storage capability as per your requirements at a certain point in time. You could be renting hardware, foundation software platforms running on that hardware, or full-fledged software applications.

As core business needs and developer roles evolve, business owners want to avoid buying expensive hardware, software developers want to avoid worrying about setting up and maintaining a software platform, and users want to avoid building software all together; they just want to use it. So business owners, developers, and users would all much rather rent what they need and then customize it, instead of setting up and managing the infrastructure.

These benefits have led to different types of cloud computing, where, in each type, you rent a different set of software or hardware. The most prominent types of cloud computing are IaaS, PaaS, and SaaS.

IaaS
Infrastructure as a Service (IaaS), also at times referred to as “Hardware as a Service,” got the ball rolling for cloud computing. Around 2009, when we first started hearing the term “cloud computing,” most of the talk was
about IaaS and, to a large extent, fueled by Amazon successfully renting out capacity via its Elastic Compute Cloud (EC2) service. EC2 lets you rent virtual computers on which you can install, deploy, and run software.

With IaaS, the vendor just provides the hardware, and the user is responsible for setting up the software platforms as required. Although this provides great flexibility to the user, it also necessitates that the IaaS user has the expertise on board to set up and manage the rented hardware.

**PaaS**

Although just renting hardware might suffice at times, developers often don’t just want the hardware, but also the basic software platform, installed and running. This brings great value to software teams because they now only need to focus on building their software application and don’t have to worry about the hardware or even the basic software platform setup. Enter Platform as a Service (PaaS).

**NOTE**

*Usually, software developers and network administrators are distinct teams. Not having to worry about the hardware setup and management means that developers are well placed to run the show on their own.*

If a software team was looking to build a Java, .NET, or PHP application, wouldn’t it be much easier if they got the hardware along with the operating system, as well as the Java/PHP/.NET software platform preinstalled and set up? How about also having an application server running, optimized, and highly scalable?

Whereas the first wave of cloud computing was around IaaS, the next was around PaaS. PaaS, however, is a much trickier space than IaaS, primarily because the PaaS vendor has to provide for the hundreds of ways in which software gets built. Many PaaS vendors, in an attempt to streamline and secure the software being built on their service, have defined strict dos and don’ts as well as the capabilities they can and cannot support. Having such a policy in place can be both a feature and a limitation. It is a feature because you can be sure that other users with whom you are sharing the platform are not free to do whatever they want and jeopardize your setup. However, it’s also a major limitation because you have to build your application as per the rules set by the PaaS provider.
PaaS adoption should grow over time, as one would expect organizations to look to delegate the hardware setup and management but continue to want a say on the software that is built and run on the hardware.

**SaaS**
The term “Software as a Service (SaaS)” has actually been around longer than the term “cloud computing.” I recall discussing SaaS in 2006, when cloud computing was unheard of. The meaning still stays pretty much the same, just that it is now thought of as a type of cloud computing.

If your business is using an online third-party service for accounting, email, invoicing, online campaigns, and email marketing, it is utilizing SaaS. SaaS is where you don’t bother renting the hardware or even the software platform on which to build your application. You directly rent the actual software application you need. You pay based on your expected usage, or in some cases, you pay as you go. SaaS is especially popular with startups because it significantly reduces their startup and running cost, while still giving the fledgling business all the flexibility it requires.

As shown in Figure 1-3, a SaaS offering builds over the capabilities of a PaaS, which in turn runs over the IaaS. For this book, we will mostly be working with PaaS.

**Why Cloud Computing?**
The top reasons for the interest and adoption of cloud computing are

- **Technology** Setting up and maintaining your own hardware and software can be an extremely expensive and tedious affair. Most would much rather have someone else do it.
Chapter 1: Java EE and Cloud Computing

- **People**  Most business owners would like to minimize/avoid the expense and effort of inducting, training, and retaining additional personnel.

- **Non-core**  Running and maintaining hardware and software is not a core business for most enterprises. Given the choice, most enterprises would rather focus on their core business and rent computing power from a large, reliable provider.

- **Scale as required**  You can rent capacity if and when it is required, and get to scale up and, just as importantly, scale down if required.

- **Pay as you go**  Businesses need not provision expensive resources to provide for possible future demand.

### Concerns About Cloud Computing

The primary concerns with cloud computing are

- **Security**  Most surveys show security as businesses’ primary concern with cloud computing. Many businesses are uncomfortable putting sensitive data on shared cloud environments.

- **Availability**  If the Amazon Cloud goes down, it takes with it a horde of websites and services. An Amazon outage in October 2012 took down many popular websites such as Reddit, Foursquare, and Pinterest.

- **Connectivity issues**  Although Internet connectivity has greatly improved over the past few years, it is still unreliable in most developing countries.

- **Lock-in and dependence on vendor**  A few large cloud vendors hold a bulk of the cloud market share, and as yet there is little standardization on cloud services, so you can easily get locked into a vendor.

- **Rigid**  Using a cloud service significantly limits the flexibility you enjoy if you were to run your own hardware and software. Most cloud vendors define fairly strict rules of operation to which users must adhere. You even need to build your applications with these limitations in mind.

- **Legal**  The cloud comes with its own set of legal implications based on locations and jurisdictions. This is a major factor, especially for large enterprises with sensitive data.
Private, Public, and Hybrid Clouds
Although the cloud is usually thought of as a public, shared setup, such a setup might not work in cases where there are serious security implications or where the applications would not work in a shared environment.

In such cases, you can opt for private clouds, which work on pretty much the same lines as public clouds, but with exclusive access for a particular organization, with the consumers being the various business units within the organization. Private clouds are usually run in an internal data center or “on premises.” However, that need not always be the case. A hybrid cloud is one that uses a mix of public and private clouds.

An important aspect of the cloud that often gets ignored is the cloud’s great ability to provide a level playing field for software development companies and developers.

Cloud as a Leveler
Today, we see that by using various cloud services, small software companies can build and run enterprise-grade Java EE software that is as fast, secure, and reliable as any application built by the mega software companies. Similarly, the cloud expertise of a solo developer or someone working in a three-member company is pretty much the same as a developer in a mega corporation and working in a team of hundreds. This is an important reason for developers to look at adopting the cloud. The cloud has drastically leveled the playing field for software companies and developers.

Now that we have had a look at Java EE and the cloud, let’s consider how Java EE is placed as a cloud platform and how it compares with some of the alternatives.

Java EE on the Cloud
Java EE has been the mainstay of server-side software development for over a decade and still today is one of the most widely used software platforms. It is arguably even the most prominent software platform on the cloud. So although there are no cloud-centric specifications or standards in the current Java EE version to date, many vendors are already offering robust Java EE solutions on the cloud. Today’s Java Cloud offers a number of service options, opening the doors wide for Java EE application development, deployment, and use.
Until recently, Java EE applications were thought of as applications that enterprises built and ran on their own dedicated server infrastructure. If a company built a Java EE application, it was presumed that the company would also set up the requisite hardware infrastructure and the teams to manage and monitor that setup. It was some time before the idea of shared hosting, which has been popular for a while with technologies such as PHP, was considered suitable for Java’s enterprise nature and demands.

We talked earlier of the benefits and drawbacks of cloud computing. You will find that most of the concerns with the cloud are features of a dedicated private Java EE setup, and vice versa.

Apart from the pros and cons for enterprises, one of the major issues with dedicated Java EE servers was that smaller businesses stayed away from Java EE because they did not want to set up and manage their own server infrastructure.

However, in almost all cases, enterprises used to run their Java on dedicated servers and hardly ever on a shared/cloud environment. Prior to the cloud wave, few web hosting providers bothered to offer decent shared Java hosting or cloud-like solutions for Java.

Once it was apparent that even enterprises were looking to go along the cloud path and spend big in the process, Java started being featured in every new hosting/cloud solution. Within no time, Java became one of the most widely used and supported languages on the cloud.

Java’s foray into the cloud has changed Java EE for good, as well as Java EE’s perception among the developer community. Developers, architects, and customers of all sizes are today increasingly looking to leverage Java on the cloud.

**Competing Technologies:**
**Alternatives to Java on the Cloud**

Java EE is just one of the many technologies you can use to build software. Considering that technology is just a means to an end, whenever a new project is initiated, developers have a choice of which technologies to use. Multiple technology platforms can enable a developer to build all kinds of software: Java, .NET, Ruby, Python, and PHP would figure to be at the top of the list of software platforms.

Although a number of Java EE hosting and Java PaaS solutions are available today, that number pales in comparison to the number of cloud/hosted
solutions for other web technologies. PHP, .NET, Python, and Ruby all continue to enjoy good traction among the developer community.

PaaS-like PHP hosting solutions have been around for quite some time—it’s just that they weren’t called PaaS back then. PHP works well even within a shared hosting setup, so a PHP app running on a fairly cheap, shared hosting solution has been the mainstay of web applications over the past decade. In comparison, Java has always struggled with shared hosting. Few hosting providers offered Java on a shared environment, and even those few usually offered a tightly sandboxed Tomcat instance that hardly ever worked out for real applications. Most hosting providers asked you to switch to a dedicated server as soon as you uttered the word Java or Java EE.

Oddly, back then, hardly anyone seemed bothered by Java’s absence from the shared hosting space. Only because the cloud triggered a surge in enterprises looking at shared cloud environments have we seen the emergence of many Java PaaS solutions.

Not Just Java and .NET for the Enterprise
The Java platform has often scored over the likes of PHP, Python, and Ruby because Java was thought of as “enterprise ready,” unlike many other technologies. However, with many large players now offering PaaS solutions for PHP, Python, and other technologies, we now see technologies around PHP, Python, and Ruby, as well as other JVM languages, offering much stiffer competition to conventional Java EE for the enterprise.

Whereas earlier, “enterprise software development” was primarily a two-horse race between Java EE and .NET, today we see many other technologies being considered. This is, to a large extent, due to the availability of enterprise-grade PaaS solutions for these other technologies.

Although each platform has its pros and cons, here are a few things that especially work in favor of Java EE on the cloud:

- Java EE was always meant for robust, scalable, distributed, multitier applications, precisely the things that you expect from a cloud application. This makes Java EE a great fit for the cloud.
- Java EE is already a mature platform on the cloud. You have many vendors, lots of choices, and all kinds of pricing models at your disposal.
The talent pool of Java EE developers is immense. Getting these developers to use Java EE on the cloud is a lot easier than building an entirely new skill set.

The Java EE community is vibrant and Java EE technology continues to get better and easier to use with every new version. Therefore, Java EE on the cloud is arguably the safest long-term choice on the cloud.

Standards and Java EE 7

Although Java EE is supported by several cloud vendors, we find that most such vendors are either supporting bits and pieces of Java EE or have built their own APIs that developers need to conform to. There have been some attempts at Java Cloud standardization, but so far, no clear standards have emerged for Java on the cloud.

Java EE 7 was initially meant to bring to life the long-awaited Java Cloud standardization. However in August 2012, 10 months before Java EE 7 was released, one of the specification leads for Java EE 7, wrote in her Java EE 7 Roadmap blog (https://blogs.oracle.com/theaquarium/entry/java_ee_7_roadmap) that it was felt that “providing standardized PaaS-based programming and multitenancy would delay the release of Java EE 7.” PaaS enablement and multitenancy support was moved out of Java EE 7 and is now targeted for Java EE 8.

Although there is no official Java EE Cloud standard being released soon, it is important to note that most Java Cloud vendors are presently looking to support standard Java EE applications.

Java EE Vendors and Alternatives

Although Amazon is credited for largely initiating the cloud computing wave, Amazon was primarily an IaaS vendor. Google App Engine (GAE) was the first PaaS solution that received widespread interest and attention. GAE opened with Python support, but introduced Java about a year later, in April 2009. We have been in a constant Java PaaS race since, with many vendors, new and old, offering Java PaaS products.

The Java Cloud vendor space is one that keeps changing rapidly, as most of the major software companies have either launched a cloud solution or are looking to do so. As of today, the prominent players in the Java Cloud
Java EE Applications on Oracle Java Cloud

space are Amazon Elastic Beanstalk, Google App Engine, Jelastic, CloudBees, OpenShift from Red Hat, and the Oracle Java Cloud Service.

Many of the PaaS providers run a Tomcat server, and some use Jetty and GlassFish. The Oracle Java Cloud runs on Oracle WebLogic Server release 10.3.6, which is the latest version in the WebLogic Server 11g line.

Some of these vendors also offer IaaS solutions where you can install your own Java EE server. However, here we are only looking at their PaaS offerings.

Amazon Elastic Beanstalk  Beanstalk is Amazon’s PaaS offering. It supports applications built in Java, PHP, Python, Ruby, Node.js, and .NET. There is no additional charge for Elastic Beanstalk, and you only need to pay for the Amazon Web Services (AWS) resources needed to store and run your applications. Beanstalk deploys applications to the Apache Tomcat server and therefore only supports those Java EE technologies that are supported on Tomcat.
http://aws.amazon.com/elasticbeanstalk/

Google App Engine  Google App Engine has only partial Java EE support and uses its own customized server and data store. If one intends to use Google App Engine, some vendor-specific learning is essential.
https://developers.google.com/appengine/docs/java/

Jelastic  Jelastic began as the “Java Elastic Cloud.” It no longer claims to be the “Java Elastic Cloud,” as it now supports both Java and PHP. Jelastic supports multiple Java EE servers (Tomcat, TomEE, GlassFish, and Jetty), as well as multiple SQL and NoSQL databases. Its positives are its Java focus and that it does not require you to use its proprietary APIs and technologies.
http://jelastic.com/

CloudBees  CloudBees claims to be the Java PaaS company that supports the entire application lifecycle, from development through production. It provides for source control repositories and Maven repositories, as well as continuous-build servers managed by Jenkins. It supports Tomcat, JBoss, GlassFish, WildFly, and Jetty application servers and supports many JVM languages and frameworks.
http://cloudbees.com/
OpenShift  OpenShift is the PaaS from Red Hat. It supports Java, Ruby, Node.js, Python, PHP, and Perl. It supports multiple Java EE application servers (JBoss AS 7.1, WildFly 8, JBoss EAP6, Tomcat, and GlassFish) and MySQL, MongoDB, and PostgreSQL databases. OpenShift’s support for a wide range of languages, servers, frameworks, and databases, is its major positive.  
https://www.openshift.com/get-started/java

Oracle Java Cloud  Although this entire book is about OJC, here’s a quick description: OJC is a Java PaaS service that runs the Oracle WebLogic server, which is an integral part of Oracle’s Fusion Middleware range and its Oracle Cloud Application Foundation. OJC supports a mix of Java EE 5 and 6 features. The WebLogic server, the standards support, the Oracle Database, support for Oracle frameworks such as ADF, and ease of use are some of its highlights.  
https://cloud.oracle.com/mycloud/f?p=service:java:0a f

Although these are the prominent players in the Java PaaS space, other notable mentions would be Heroku from Salesforce.com and Cloud Foundry from GoPivotal (formerly VMware). Already, a wide range and depth to the Java PaaS offerings is available. These offerings vary on multiple factors, so a close examination of multiple vendors is usually needed before one can pick the right Java PaaS for their requirement.

NOTE
The WebLogic server is integral to Oracle’s Fusion Middleware range and its Oracle Cloud Application Foundation, and is the server used on the Oracle Java Cloud Service.

Tens of factors go into why one would choose one PaaS over another. The following are the top considerations (in no particular order) and how the Oracle Java Cloud (OJC) performs on each one. There’s a fair bit of overlap between the features that these cloud services offer, but the key points to consider from a purely software development platform point of view are as follows:

■ Pricing and billing  Costs and pricing strategies vary widely across vendors. Some charge based on fine-grained usage details, whereas others provide duration-based subscriptions. You need to evaluate if you would like to go with subscriptions or with a “pay-as-you-go” model.
OJC: Offers a monthly subscription currently starting at $249 for a single WebLogic server instance. It does not offer a “pay-as-you-go” option. OJC offers three broad editions, each of which come with a definitive set of resources. You pick your edition and pay a flat monthly rental for the same.

- **Supported features and technologies** Are the supported technologies and features in line with your requirements? Is your chosen framework officially supported by the cloud vendor? Which version is supported? Is the vendor supporting standards, or do you have to write custom, vendor-specific code? Many vendors support only subsets of Java EE, and in some cases, require you to use and adopt their custom technology/API. If you intend to develop a pure Java EE application, check whether the vendor supports Full Java EE/Java EE Web Profile.

  OJC: Java EE standards support is one of the primary pitches for OJC. It currently supports most of Java EE 5 and many of Java EE 6’s capabilities, but isn’t yet fully Java EE 6 or even Java EE 6 Web Profile compatible. Its support for the Oracle ADF framework is a plus for those with existing ADF deployments and development setups.

- **Flexibility** Many vendors insist that you develop in a certain way using certain APIs. This isn’t always possible or easy to execute unless you have a team adept at developing as per that cloud vendor’s requirements.

  OJC: The OJC relies on Java EE standards and does not insist on you using any non-Java EE or Oracle-specific APIs, unless you wish to leverage any features specific to WebLogic or Oracle’s cloud.

- **Vendor standing** Considering that you are putting your application and data on the vendor’s hardware, you want to be confident with the vendor’s credentials and ability to be up and running, say, 10 years from now.

  OJC: Oracle rates highly on this count. The Java Cloud Service should be around for a long time.

- **Tooling and ease of use** Many cloud vendors have rich web-based UIs, and some even provide integration with popular integrated development environments (IDEs). Ease of use is quite important
because some cloud services can be rather confusing and, at times, even intimidating.

**OJC:** The OJC has a decent browser-based UI and supports integration with popular IDEs, including JDeveloper, NetBeans, and Eclipse.

- **Database support**  Most cloud vendors support at least one RDBMS and NoSQL data store. You need to check if it’s the one you prefer.

  **OJC:** The OJC supports the Oracle RDBMS, but there’s no NoSQL database as of now. However, considering most other vendors are offering a NoSQL option and Oracle has a NoSQL database, it should only be a matter of time before Oracle NoSQL is available on the Oracle Cloud.

- **Open/closed: vendor lock-in**  Is the vendor offering a closed stack that would lock you in? Would it be possible for you to migrate to a new server if the need arises, or are you getting locked in to a particular vendor?

  **OJC:** OJC fares well on this count because of its emphasis on Java EE standards-based development. Migrating from the Oracle Cloud to a dedicated server or another PaaS vendor should be possible.

- **Java friendly**  Whereas some vendors are focused Java Cloud players, there are others that support many different technologies. This does seem to affect the features, the documentation, and the overall priority areas for that service.

  **OJC:** Being a purely Java Cloud service, OJC is certainly Java friendly. The UI, features, and capabilities all seem to be built with a Java developer in mind.

- **Skill building**  How difficult would it be to build a team capable of developing and deploying for a PaaS?

  **OJC:** Again, due to OJC’s focus on Java EE standards, it is much easier to build a team for OJC than for other PaaS solutions, which require skill building on a vendor-specific technology.

Apart from the software platform issue, there are, of course, other non-software issues such as support, service level agreements (SLAs), and server locations that need to be considered.
Summary

In this chapter, we had a look at the basics and origins of Java EE and cloud computing. We discussed the various kinds of cloud services and why Java EE is a great fit for a cloud environment. We took a look at some of the prominent Java Cloud vendors today, as well as the things to consider while picking a Java Cloud.

We also looked at Java EE in more detail, reviewing JSRs, the various application servers, and the Java EE profiles. We followed this up with a discussion on Java EE PaaS and how the Oracle Java Cloud fares on some of the key parameters for a PaaS, concluding with a look at PaaS alternatives on other technologies.

Let’s now take off into the Oracle Java Cloud, set up our Java Cloud instance, and start looking at the nuts and bolts of Oracle’s Java Cloud offering.