

Moving Java EE.next to the Cloud

Ralph Ellison said, "When I discover who I am, I'll be free".

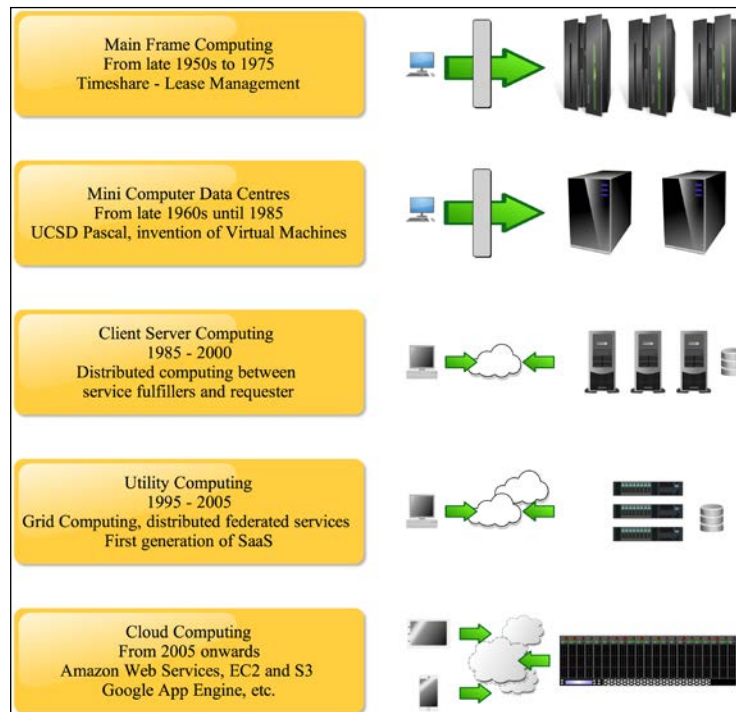
The expert group of Java EE 7 originally proposed the platform version to aim for standardization of cloud - computing platforms known as **Platform-as-a-Service (PaaS)**. From January 2010 to July 2012, the expert group ran with this initial idea, until they realized that Java EE 7 priorities and project aims were misaligned and that the enterprise Java cloud - computing market place was premature; it was too early to standardize The Cloud and fortunately for Java EE 7 they withdrew the notion at the very last moment.

The current digital scope of the Java cloud providers is in a state of flux and in principle moving Java EE to the cloud is a good investment for the enterprise Java platform. The question is how to specify the Java inside a cloud environment so that it is fair, competitive, and sustainable to all participants. This chapter lays the ground work for the tough, challenging, engaging technical projects that are surely going to come further down the line.

Ever maturing Java

Java itself is divided into three editions, originally by Sun Microsystems: the Standard Edition (SE), the mobile edition (ME), and the subject matter of this book, the enterprise edition (EE). Furthermore, Java is growing into other devices, desktops, and form factors.

In 2013, if you have solid, respectable, and current enterprise Java skills, you are well placed in the software industry. The following shows exactly that:



Our journey so far

Let us consider our journey from the past to the present. We ought to understand why we have cloud-computing now. In terms of enterprise computing, there have been four epochs in the digital age.

During the 1960s enterprise computing was achieved by running programs on mainframe and mini computers and was certainly achieved by time-sharing. The expensive resources in this epoch were processing speed, amount of memory, and disk space. The invention of multi-tasking marks the first step of virtualization.

At the beginning of the 1970s, **Xerox Palo Alto Research Centre (PARC)** institution was established. This famous Californian institution invented Windows, Icons, Mouse, and Pointer. PARC would be the inspiration for the desktop user interface in Windows, Mac OS X, and Linux operating systems. In 1969, AT & T Bell developed UNIX, one of the first multi-tasking operating systems.

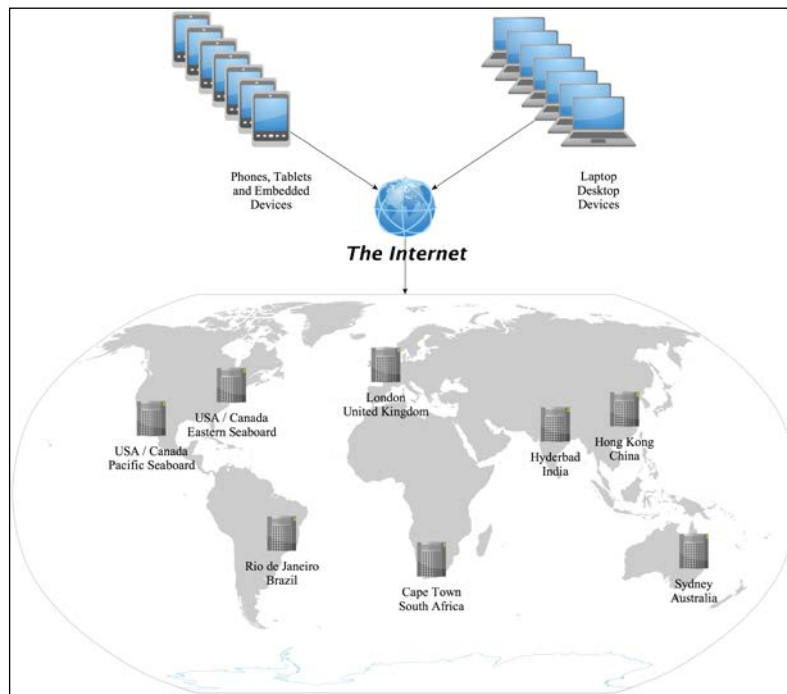
The second epoch began sometime in the late 1980s with the idea of client/server computing. The industry pushed for workload to be distributed across computers. Business functions were executed on a client, which requested a service, a unit of work to be completed on a server. Client/server computing relied on the existence of a network in order to spread the architecture of a distributed application.

The 1990s saw the birth of the World Wide Web, the exponential growth of the Internet, and the wider adoption of client/server computing. Client-server computing grew popular. It eventually caused businesses to re-evaluate the logic of executing code using time-sharing on large mainframes and mini-computers. The increasing demand for client-server was symptomatic for spread of commoditization.

The third epoch, from late 1995 to 2005, saw the growth and appeal of grid computing. There were several attempts to provide a utility computing platform; Sun Microsystems is notable for providing grid computing as a pay-as-you-go for amount of CPU/hour. Unfortunately, history has shown that the business model was wrong.

Commodity hardware had been connected together in very large clusters of servers in order to solve big-compute problems. The effort of the grid was restricted to scientists, government researchers, military research, and projects where high performance computing was required. However, during this epoch of utility computing, which for the very first time saw complex Internet applications accessing and invoking services running on a grid and utility computing environment. Well-known examples of these achievements were Google Mail, Yahoo Mail, and Microsoft Hot-mail, the earliest forms of **Software-as-a-Service (SaaS)** for free.

The fourth epoch, since 2005, is properly marked for the term cloud computing. Amazon released the **Amazon Web Service (AWS)** to the public. The release of **Amazon Elastic Computing Cloud (EC2)** grew rapidly in popularity as developers, service providers, and entire businesses started to build their business model around the technological shift known as The Cloud.



Cloud computing attempts and delivers to business the prospect of high-availability, utility computing, and scalable resources across geographic time zones with sustainability and economy.

Too early to standardize

This book was written in the 2012 to 2013 time frame and we witnessed constant acceleration of changes in the state-of-the-art in cloud computing providers and solutions. Business demands from the customers and the further boarding of emerging nations onto the Internet as we progress through the decade means that there is a demand for server-side computing infrastructure. In order to handle this demand, businesses are looking at the cloud. Why has there been this sudden demand around 2010 and afterwards? Part of the reason is the movement of the consumers from using desktop and laptop PCs to mobile devices and smartphones.

Post-PC phase

In 2010 by *Mary Meeker*, a venture capitalist and former Wall Street securities analyst, successfully predicted that the global number of smartphones and tablets would surpass the production of desktop and notebook PCs in 2012. In fact, there were 51 million units of tablet computers shipped globally in Q4 of 2012; that just about exceeded the notebooks and far supplanted the 37.5 million desktop PCs that were shipped.

The aggressive momentum in mobile computing is the main driver behind the economic surge in scalable computing on demand, though admittedly cloud computing was in existence since mid-2005 and serving users on desktops and laptops. Nevertheless global mobile traffic is expected to surge higher for the rest of the decade. Mobile computing is setting the profit margins and revenue indicators.

In February 2013, Google released a brand of wearable computer with a head mounted display called Glass. The next demand on cloud computing could possibly be to provide services readily and accessibly to people who wear mobile computers. Only time will tell and only when we can all look back in five, or maybe ten years, can we completely judge the present and what the future might shape up to be. For now, we ought to get a good handle on this thing that everyone is calling The Cloud.

The Cloud

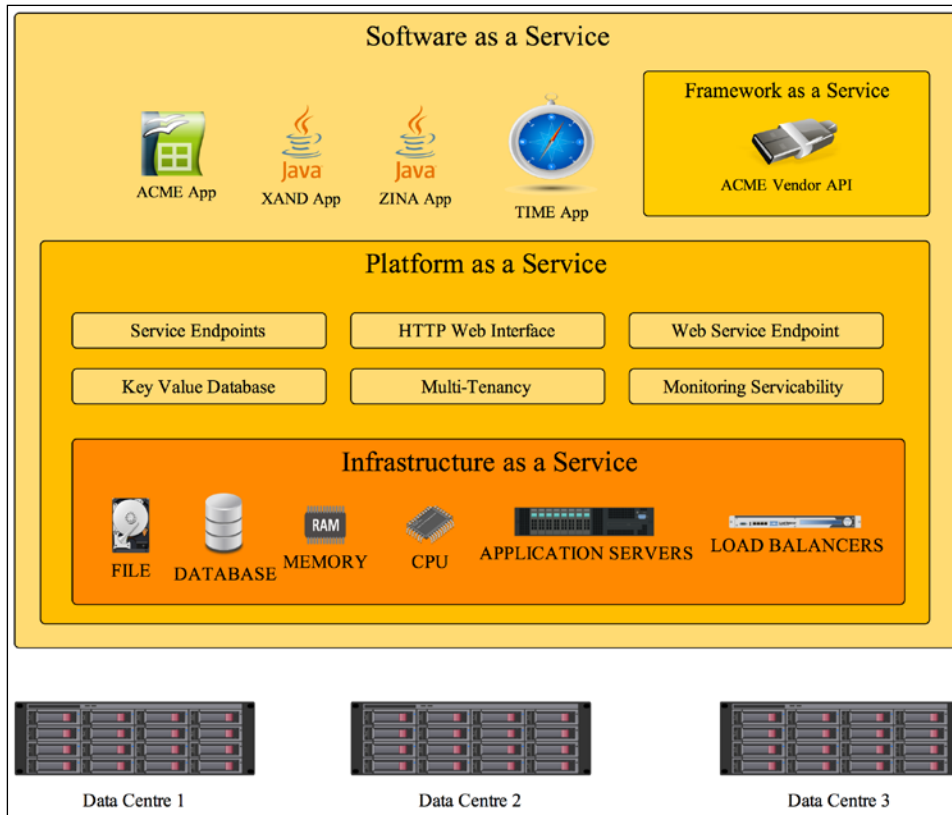
The phrase Cloud Computing is the hottest buzzword in computing in the beginning of the twenty first century. Why do people call it the cloud? It probably has to do with drawing that system architects used to communicate how their requirements affected the balance of forces. The drawing of a squiggly cloud on a whiteboard is synonymous with some sort of abstract networking and remote communications between at least two distributed systems.

These are the fundamental elements of cloud computing:

- **Pooled Computing Resources** into generic clusters that consumers can subscribe to
- **Virtualization** of operating systems, sometimes several, onto shared commodity machines that consumers can control in order to reduce cost of ownership
- **Elastic Scalability** of resources and virtualization in order to scale up (and scale down) against demand

- **Automation**, the ability to create new instances with operating systems and virtual machines, and remove existing ones as the client or business requires
- **Pay-As-You-Go** business model that permits billing to be charged for only the services that the subscriber uses.

The following diagram illustrates the Russian doll encapsulation of the various anything-as-a-service. The lowest level is IaaS and the highest level is SaaS.



XaaS, where X stands for one of Infrastructure, Platform, Software, and Framework. The data centers are ideally distributed in wide geographic area. If you want to get closer to the so-called "metal", pick IaaS. If you are only interested in buying software, then choose SaaS, and if on the other hand you want to develop your managed applications then choose **PaaS**.

Infrastructure as a service

The lowest level of a cloud service is quaintly known as **Infrastructure as a Service (IaaS)**. It is the most basic level of service for a cloud platform, other than providing hardware as a service. IaaS, then, is an environment for building native applications. **Amazon Elastic Compute Cloud (EC2)** is a good example of IaaS.

As the computer industry moved from centralized servers to distributive architectures, we have seen the move into virtualization. IaaS vendors offer virtualization of operating systems that are tailor made to run on specific commodity hardware. A customer can subscribe to IaaS and install their own specific operating system image or more often than not, the IaaS providers will have default images, which they have already tested with their cloud environment.

Infrastructure as a service has the following attractive advantages:

- Lower cost of ownership
- On demand allocation of virtual machines to cope with peak time and foreseen operational challenges
- The ability to provision custom operating system images that the subscriber has put together
- The transfer of the professional infrastructure support to a vendor who has the expertise to maintain the environment and therefore reduce the operational cost of the customer's own business overheads
- Ability to experiment with new operating systems and other development environments with reduced cost of buying into the technology wholesale and up front

The low cost of entry into a service-oriented architecture has high appeal to business entrepreneurs. So it is not surprising that those startups have taken a keen interest in services such as Amazon Cloud Foundation, Microsoft Azure, and other such examples of IaaS providers. There are disadvantages to the IaaS and some of these are:

- It is the subscriber's responsibility to manage both the infrastructure and the distributed cloud application
- Subscribers need, therefore, to have necessary up front expertise in the operating system, virtualization, memory, and performance utilization
- Subscribers are responsible for managing their own CPU and bandwidth

IaaS is the low-level cloud computing and Java only fits in here with the provision of Java Virtual Machines.

Platform as a service

Platform as a Service (PaaS) is the model of the cloud provisioning that is one level above the infrastructure as a service. Instead of just providing a virtual operating system, there is an entire application service interface available to developers, designers, and architects. PaaS, then, is an environment for building a managed application that executes only inside a runtime container. An excellent commercial example of Platform as a Service would be Google App Engine.

The advantages of Platform as a Service are:

- Increased level of abstraction above that of Infrastructure as a Service.
- Business subscribers can develop applications often more quickly and deploy such application with less knowledge of virtualization practices.
- PaaS often delivers ready-made computing resources by default. Services can include persistent storage, memory cache, load-balancing, federated servers, system monitoring, logging services, point-of-control operations, and security options.
- There are often a great number of people on the recruitment market who have the skills required to be application developers rather than infrastructure operations support.

The PaaS solution provides resources such as persistent data storage, web services, message queues, distributed applications as part of an overall service level agreement. The platform provider is ultimately responsible for CPU and bandwidth whereas the subscriber is charged on demand for the resources that are used subject to a contractual agreement.

Disadvantages of PaaS are:

- Locked into the platform, especially if it heavily leverages proprietary interfaces
- Less flexibility to perform fine tuning of the managed platform
- Subscribers can only develop applications in the supported programming languages in the PaaS solution
- Security of the PaaS may not be appropriate enough for data privacy and banking transactions

Although Platform as a Service is easier to get into, the application architect must think carefully about portability, security, and performance.

One plausible reason why PaaS may not be appropriate could be a situation where the business foresees that it will need direct control of CPU resources and bandwidth in order to provide performance tuning down to the hardware level. A solution such as a Google App Engine may not be appropriate for such a compute intensive and/or highly available application, therefore the client may consider an IaaS instead.



Cloud architecture and PaaS

PaaS solutions work well in scalability and high availability for particular types of technical requirement. As an application architect, it is your responsibility to consider the workloads and demands of your application – your PaaS cloud provider will not do this for you. Different vendors offer incompatible PaaS level of service and of course there is the permanent engineering idiom to contend with trade-off – the ability to configure and control versus the ease of use.

If your application has particular obligations on special configuration of Java Virtual Machines or certain types of memory requirement, then PaaS could be too generic for your needs. This reasoning also applies to cache mapping scenarios and input-output metrics.

Enterprise Java certainly fits the PaaS concept and in fact several vendors already offer these types of solutions for Java EE 7 before any standardization by the JCP.

Software as a Service

Software as a Service (SaaS) is one level above PaaS, where the service is a software "on demand" solution. In other words, the final working software is the only deliverable product in this case, in comparison to PaaS. Usually the SaaS application solves a specific business need, which may be horizontally aligned across the industries. SaaS, then, is managed and packaged software applications. Examples of SaaS solutions are systems designed to provide **Customer Relationship Management (CRM)**, **Enterprise Integration Platforms (EIP)**, or **Human Resource Management System (HRMS)**. The industry often positions an example of a SaaS solution as being www.salesforce.com.

What is the difference between a typical application service and a cloud platform SaaS solution? The SaaS solutions share the common theme of a XaaS: the software is delivered as a part of distributed dynamically scalable cloud platform. The software is wholly provisioned by the cloud vendor and the subscriber does not load or install the application on their systems. The business model is an on demand, **Pay As You Go (PAYG)** and subscription based license model. The customer only buys the access to the software and the provisioning that it requires for the SaaS solution.

SaaS solutions are level above the PaaS because they are generally about a fully completed application running in the cloud. There may be a form of customization allowed in order to, for example, personalize a particular SaaS application to the branding of the business.

A SaaS vendor may offer a **Framework as a Service (FaaS)** as part of the overall solution, which permits customer developers to further configure the SaaS application to their needs. The overall FaaS is much smaller in scope, more restrictive, and less generic than a PaaS. In other words, business subscribers are not usually able to build their own SaaS application from a FaaS.

FaaS, then, is an environment for building a module for another sort of cloud application, usually **Enterprise Resource Planning (ERP)** in the business technology sector.

The advantages of SaaS solutions are:

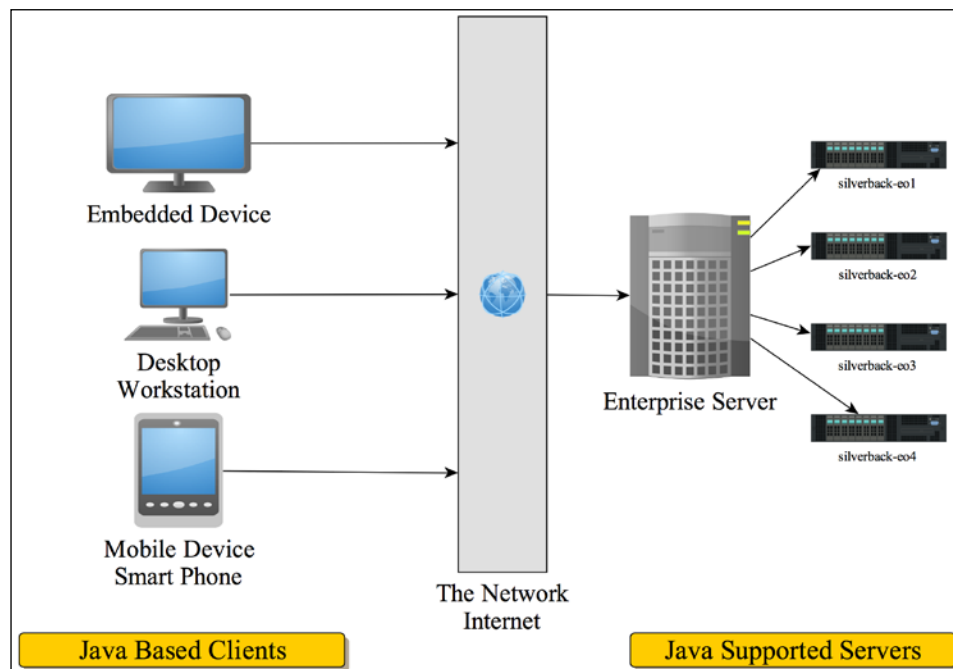
- It is a direct solution for businesses, if and only if the SaaS solution meets their goals, solves a particular business process solution, saves money, increases profit, improves productivity, and so on. The business may find that SaaS offers a so-called one-stop shop solution.
- The scaling of SaaS solutions is on demand. It can be grown or reduced, as and when the business subscriber requires it to be so.
- There is a usually very minimal up front software developer cost that may mean absolutely zero cost to the business. The subscriber just pays for computing resources, pushes any change-request, feature enhancements over to the SaaS provider and they deal with it.
- The operational thought after purchase of a SaaS solution can be treated afterwards as simply business-as-usual.
- The SaaS provider might also offer a Framework-as-a-Service (FaaS) available to the business subscriber to extend and/or brand their view of the product. Such FaaS function can be a personalization kit. It could also be a plug-in architecture.
- SaaS solutions often exude the ability to share and collaborate on information, sometimes in real system. Multiple members of staff, employees, contractors, and managers can work together in a group. Often this type of SaaS solution may offer a work flow feature as part of the product.
- Upgrades and bug fixes of the SaaS solution are the responsibility of the provider rather than being part of the subscriber's hassle budget.

The disadvantages of SaaS solutions are:

- Business subscriber is locked into the vendor that provides the SaaS, which is really saying nothing more than current business-as-usual practices. If a business buys a commercial product there are usually more than two proprietary features.
- There is a considerable lack of extensibility in SaaS solution. The business does not own the source code; it cannot further develop the solution and release new changes. The loss of ultimate configuration in a SaaS solution could be an issue if the business suddenly faced urgent change.
- The business has less control over the quality of their data. Stakeholders must think harder about how to provide evidence and audit information for regulatory compliance. They must also decide on a clear strategy on data security and privacy of customer data and how the external cloud systems integrate with in-house ones.

At the time of writing, a viable Java SaaS application could be built from the PaaS vendor stack, albeit with the risks of the vendor-lock and proprietary APIs. In other words, this is a business risk.

The following diagram illustrates the basic client-server architecture of Java enterprise applications today:



This is the way that architects have built Java EE applications, with multiple types of clients invoking endpoints on a cluster of application servers, each running on a separate JVM.

Data-as-a-Service (DaaS) is a new cloud term on the block, which is closely associated with SaaS though it is a popular notion to understand that relational databases do scale as well as infrastructure, platform, and software. There is, however, considerable movement with some innovations into scaling databases by partitioning database tables, adding hardware CPU optimization for global transactions, and very high speed networking between servers using proprietary technologies, such as **Infinispan**, in order to replicate data across nodes. Data can only be supplied on demand through **Service-Oriented Architecture (SOA)**. There is an increasing demand for data that is pre-processed in order to clean it from artifacts and allow it to be sold to businesses whenever they need it. Data on demand is about fast agility, lower affordability, and high quality.

Multi-tenancy

In order to achieve some of the cost efficiency of deployment, XaaS typically utilizes the principle of multi-tenant Architecture in a cloud computing environment.

The history of multi-tenancy is not new at all. In fact, the very first data centers with mainframe exhibit this form of computing access through time-sharing. Business in the 1960 achieved economies of scale by charging for access to machine. It took the form of renting computer time. A multi-tenant was given a special ID to log on and the user (the entity customer identifier) was charged for their mainframe access.

Multi-tenancy changed in the 1990's when **Application Service Providers (ASP)** hosted business applications on behalf of their customers. Large scale applications, then, were often spread across separate machines and shared access to several customers. These distributed applications, often clustered, load-balanced, and federated servers, were the forerunners of today's cloud applications.

In cloud computing, the concept of multi-tenancy permits a provider to segment and isolate individual groups and users. A multi-tenancy application, then, is a single of the application, which runs on a server and serves multiple client organizations (tenants) simultaneously. Multi-tenancy is a key driver of cost reduction. By allowing simultaneous access, physical hardware, memory, and storage costs are reduced, because these resources are shared.

The key provisos in multi-tenant applications are that each tenant must be isolated from interference from other tenants. Each tenant must have security in memory and storage. There should be no corruption of each other's data through unauthorized access, accidentally or deliberately.

The advantages of multi-tenancy:-

- **Huge Cost Saving:** There is a huge reduction of cost for hosting each customer on a separate private server
- **Aggregation of Data:** For all customers for example in a SaaS, FaaS solution, it is relatively easy to mine customer data in one place for market research purposes
- **Collaboration of Data:** Certain aspects of customer data can be easily shared between customers, if both are in agreement
- **Broadcast Release Management:** Releases apply to all customers on a given set of nodes

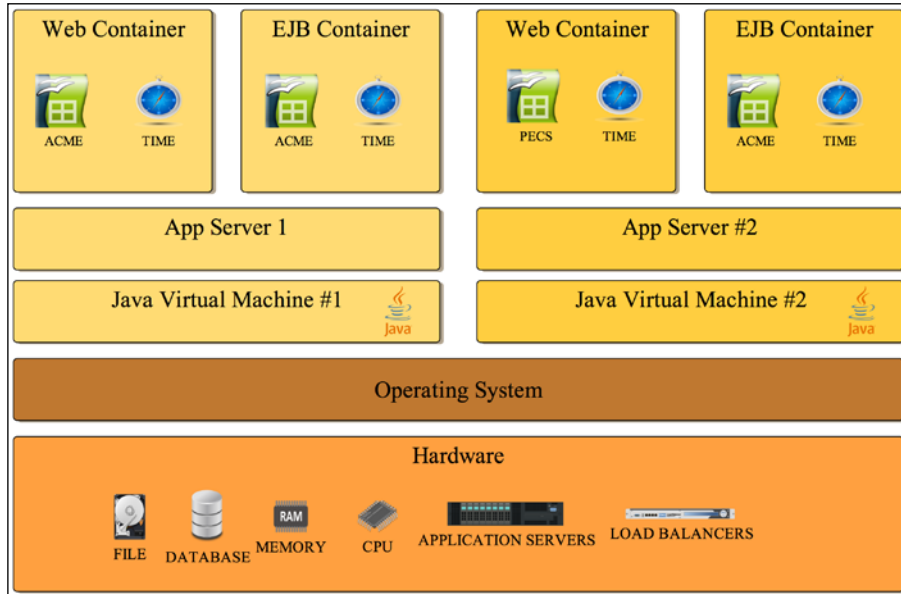
The disadvantages of multi-tenancy:

- **Insecurity:** There is risk of insecurity if the application architecture is ill conceived, or inadequately tested and certified. Although security is the chief responsibility of the XaaS provider, a serious flaw in the application could be a gateway to hackers.
- **High Maintenance Cost:** The business data may not be appropriate for multi-tenant implementation. A plausible scenario is when the gamut of users outpaces each user's private data segment by more than an order of magnitude.

Java has supported some idea of multi-tenancy for a while. A Java EE application server can readily serve separate enterprise applications (EAR), and a web container may host multiple web applications (WAR). Java EE 7 does not support data multi-tenancy for a single application.

If the next edition of Java enterprise did add data multi-tenancy into the standard, then it would be entirely possible to write e-commerce applications in the cloud that allowed different business customers to effectively share the same storage. Data multi-tenancy, therefore, will require large modifications to the JPA standard and may well affect other areas of Java EE, including CDI, EJB, and JCA.

The following diagram depicts multi-tenancy in a possible PaaS configuration:



There are two JVMs running on a machine, and each one contains a managed application server. Two application servers are both running the TIME application and they share the same database, memory, and resources. A customer that uses TIME should be completely unaware and isolated from other customers regardless of which application server is handling their data.

Java EE for the Cloud

In 2012, the expert group was heading toward cloud computing standardization, until they removed these requirements from the Java EE 7 specification. Java EE 7 was released as a specification on June 12, 2013. So what happened to the cloud parts of the specification? These details were parked for discussion in the expert group in the next Java EE release.

The expert group originally conceived that Java EE would function mainly as a PaaS and have some features of SaaS. The SaaS support is through multi-tenancy requirements of the individual JSR to make up the overall umbrella standard.

For the purposes of this discussion, since I do not know whether these will be included in the next standard or be a point release in a future specification, I will refer to the next Java EE as **Java EE.next**.

Java EE.next adds contractual agreements for business, and their enterprise developers, targeting the cloud as a deployment environment.

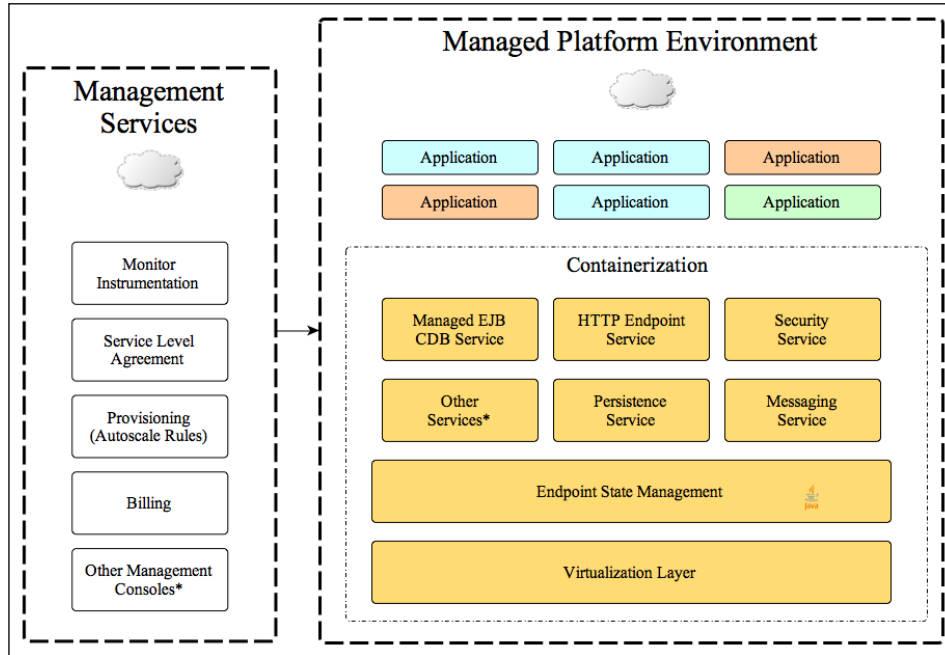
In the earlier Java EE specifications, including J2EE, positioning a load-balancer in front of a set of clustered Java EE servers provided for service provisioning. In this traditional deployment model, a request would arrive from an end user client, a web browser, and then be routed to a particular Java EE server. Each Java EE server would run a given instance of the application. In common, many of these application instances would share a single database server. The database would normally be a relational, and of course, as we know, it is hard to scale relational databases.

Relational database tables are much harder to partition across distributed servers, if they are to maintain atomicity, consistency, isolation, and durability. It is possible to replicate tables across systems, however it is always going to slower and costlier.

Because of these current constraints on relational databases' inability to scale massively on demand, the emphasis has been on key-value database stores, so-called No-SQL databases. To the average developer, key-value stores look like hash tables, and from a computer science algorithm perspective one can think of them quite like that. Key-value stores in the cloud are designed from the outset to be distributed; the data is easy to replicate across several servers, and therefore it is easier to scale. There are limitations, however, such as eventual consistency, the lack of two phase commit transactions.

At the moment in 2013 there is no standard Java API designed for key-value storage directly; however some providers have facilities that look like Java Persistence API. Developers might think that they can easily migrate to the cloud, an existing application written against existing JPA and JDBC application. There are design and implementation consequences and therefore it might be foolhardy to attempt this move without a database refactoring plan, partitioning, and execution analysis. It is simply not recommended.

The following diagram is a depiction of the probable Java EE.next cloud platform:



Multiple applications will sit inside a **Containerization** structure that has generic infrastructure (in order to support fast auto scaling). The containers are managed Java EE platforms of multi-tenant cloud applications. Although we cannot be sure of the actual details, there will be a **Persistence Service**, which may support non-relational databases, a **Security Service** (obviously), and a **Messaging Service**. The most crucial of these services will be the HTTP interface layer, which may have restrictions or could possibly be extended. It is all up for debate.

Perhaps the most important requirement in the Java EE.next cloud platform will be business rules that will dictate how a provider interacts with the **Provisioning**, **Billing**, and the provisioning services. These have technically nothing to do with Java EE; however, they require thought on how a satisfactory **Quality of Service (QoS)** will be achieved. It is also very plausible that modular applications themselves will want access to some of this very important metadata about their parent container.

Java EE.next would explore extending JPA 3.2 to support cloud-computing deployments and key-value storages. In particular the next standard could possibly define automatic provisioning through annotations of data source, which would be essential in on demand situations.

In Java EE 7 there is a leak of the original ideas of application cloud provisioning through the JMS 2.0 annotations. It serves an example of intent. There are two annotations `@javax.jms.JMSConnectionFactoryDefinition` and `@JMSDestinationDefinition` which can be applied to a Java Servlet or stateless session EJB. Java EE.next would look to apply these ideas to the other specifications such as JPA, EJB, Servlets, JAX-RS, and web services.

Currently, the Java EE 7 release does provide containers that already have a degree of multi-tenancy. It has always been sufficient to cluster a set of EJB end points across distributed servers using a load-balancer. Scaling endpoints, such as Servlets, Web Services endpoints, and stateless EJB is not normally an issue. (The key issue has always been the relational database. Hence certain cloud providers already can provide a cloud solution around GlassFish 4.0 application server.) Not surprisingly, cloudifying a Java EE application that makes relatively little stress and demands on a relational database, for instance one where the application is heavily read-only database versus write-only database, can be amenable to scaling massively.

Let us look at the responsibilities of Java EE.next in the cloud:

- Support for separated instances of the same application for different tenants
- One application instance per instant
- Tenants correspond to units of isolation
- Each instance customized and deployed for a single tenant
- Mapping for tenant is sanctioned, provided by the container

Java EE.next would allow single enterprise application to be multi-tenant capable through discriminator columns in a database server or key-value store. The issue is how to share this database in a safe, secure, and consistent manner. True multi-tenancy, however, relies on a future version of the Java Runtime Environment, where the JVM itself is multi-tenant capable.

Modularity and Java enterprise platform

In July 2012, *Mark Reinhold*, chief architect of the Java platform announced that the important **Project Jigsaw** was delayed. Project Jigsaw is the effort to bring modularity to the Java platform. The earliest we, mere mortal developers, can expect to see a standard and statically modular Java Runtime Environment would be in Java 9.

This shock announcement caused, shall I say, a not unnoticeable degree of muttering amongst the chattering developer classes. There was some unrest in the worldwide Java community. The knock-on effect is that Java EE is probably not going to have modularity until Java SE 9 is released.



No modularity in Java SE 8, but could we have more profiles in Java EE?

There is positive news however. Java EE 6 defined profiles, web and profile. So it is entirely possible that Java EE 8 and the expert group could define additional profiles in the future editions of the specification.

Cloud deployment features

The whole point of cloud computing is to make life easier to deploy modular applications into an infrastructure that can automatically scale dynamically. The business can add more servers (and machines) in to order scale horizontally and also scale the resources on a single machine in order to scale vertically. Cloud computing makes this easier without losing business, stopping an application and then restarting it. It all happens seamlessly, and this only can happen foreseeably in a Java EE.next platform that supports modularity and a notion of versioning.

Cloud computing is perhaps new to most Java EE developers. Essentially to move to a cloud solution involves:

- Defining a cluster infrastructure. How do you want the application to scale out?
- Define a quality of service agreement. What services does your application need? When does it need them?
- Deploy the application to the cloud. Have you decided on a monitoring plan?

We covered the essentials of cloud computing and its association with the Java platform. The Java EE 7 originally defined a set of business and application developer roles and extended them to PaaS cloud computing platforms.

Java EE.next roles

The Java EE.next standard defines a set of roles that are common to all platforms. The roles are associated with responsibilities, and while you do not have to be overly familiar with them, you should be aware of their existence. The standard defines these roles in a strict separation. It is up to the Java EE implementations and providers to decide how they make these roles available to the software developer.

Java EE product provider

The `Java EE.next` product provider is the role reserved for the provider of the Java EE components and application programming interfaces. It is the responsibility of the `Java EE.next` product provider to make the API available through the defined containers. The product provider is normally the enterprise server vendor, web host vendor, the middleware vendor, a vertical industry specialist and vendor, or, in the case of a business like Oracle, a database vendor. It can also be an operating system provider, for example like Red Hat does through Linux, and since Java EE 7 it can be cloud service provider.

Application component provider

The role of application component provider is the role of the person who generates a specific component. Exactly how the component is created varies according to the skill set. It may be an HTML5 and CSS wizard, a specific enterprise developer, or database schematic expert. The application component provider may use tools in order to generate application components. The tool could be straightforward, such as an everyday Java IDE. The tool could be special and it might be a code generator. We do not preclude specialist tools such a model-driven architecture, vis-à-vis *Eric Evans*. In this case, the input is a domain specific language and then one would use a special tool like Eclipse's **XText** or JetBrains' **Meta Programming System** to parse the DSL, semantically analyze it, and then generate components.

Whether a tool is used or not used, the end result for the application component providers are a set of components that are compatible with `Java EE.next`, such as an EJB, a Servlet, Message, or Web Service endpoint, which then can be bundled together into a Java EE application suitable for deployment.

Application assembler


An application assembler is responsible for bundling together the different `Java EE.next` components, such as Enterprise Java Beans, Java Server Faces, Managed Beans, and other pieces into a complete application. The application assembler may also configure the base level dependencies between the components together, and this may be achieved through configuration files. The result is a fully assembled **Java Archive (JAR)**, **Web Application Archive (WAR)**, and/or **Enterprise Archive (EAR)** file.

Deployer

The role of a deployer is to take a pre-assembled bundle and then install it on the Java EE.next server, in fact, into the containers. The deployer is also authorized to undeploy applications, although some enterprises may decide to further split this specific role for ultra-sensitive security reasons using the Java EE.next product features.

During an installation, the deployer takes a JAR, WAR, or EAR file and then uploads the bundle onto the server. If there is already an application running on the server, they may have to stop it beforehand. The deployer will then configure the resources and the facilities that this application requires. Finally, the deployer will start the application into execution mode. The application, then, is ready to receive web HTTP and web services requests.

The situation is reversed during an uninstallation. The deployer stops the running application from execution. After a graceful termination of the application, they will then remove the instance from the server. Depending on the actual server, if the situation is warranted, the application's resources can be further removed; database connections and message queue endpoints can be deallocated and then deleted.

 This style of deployment is manual, and the description is wholly about human interventions. In the cloud environment, deployment is automated in a style called provisioning.

Systems administrator

The systems administrator is a role specifically separate from a deployer. The administrator looks after the Java EE.next product. They make sure that it is operating successfully, that there is enough memory, processes, and CPU time. In other words they monitor the server and validate its availability, scalability, and performance over the longer time. If there is a problem with the Java EE.next product, for example if it runs out of available memory, they are responsible for getting the entire server back on line. System administrators are those people who restart the servers very late at night or very early in the morning.

With this tremendous responsibility comes a certain modicum of skills; system administrators not only need to be infrastructure experts, and familiar with the operating systems, they will also have database administration skills. A really good systems administrator will have messaging system knowledge and a notion about the Java EE.next product, and if they are truly blessed, they will have essential know-how about JVM garbage collector.

Once again, the role of system administrator is different in a cloud computing environment. They will require all the mentioned skills, and will be rather familiar with monitoring systems, which aggregate lots of logging, monitoring data across several server racks. In fact, they will need skills commensurate with a PaaS engineer. Those skills will include greater in-depth knowledge of partitioning data, networking infrastructure, regionalization of service, service escalation procedures, and provisioning of application and data management.

Tool provider

The tool provider role is a specific responsibility for Java EE.next products. The role of a tool is to assemble Java components for a Java EE.next product server by eventually producing a JAR, WAR, and/or EAR files. A deployer uploads these end-product files to a server.

The tool provider role in my opinion is considered legacy and redundant, because de-factor software already covers this requirement in the specification. Tools such as **Integrated Development Environments (IDEs)**, such as Eclipse, NetBeans, JetBrains IntelliJ IDEA, and popular standalone software build tools like Gradle, Maven, and Ant.

System component provider

The system component provider role is responsible for interfacing between systems that are not necessarily Java compatible. Environments that required **Java Connection Architecture (JCA)** are those that require a specialist provider. System components can be messaging systems, environment where there is a requirement for Enterprise Application Integration, and remote web services. Often these other systems are written in other programming languages and platforms. An example of such a foreign system would be a mainframe system.



Refactor out Java connector architecture

In 2003, a bank had already realized that they could not rely on this transaction manager to control storage of data into multiple resources, databases, and EJB. The client instead decided to avoid using the JCA implementation, then, newer J2EE projects. The licenses to upgrade the JCA product proved to be exorbitant. Every upgrade of the application server product caused a testing phase with the JCA implementation. The bank simply had no choice but to verify the operation of the application with the JCA dependency before they could upgrade to a new application server version. In the end, another department in the same division, in the next year, wrapped a XML SOAP web service interface around the JCA. The moral of the story is to first look at open source solutions before getting yourself locked into vendor, and then, even then, always hedge your bets.

Cloud PaaS provider

The role of cloud provider describes any entity that hosts a Java EE.next product in a cloud computing environment. The responsibility of delivering a standard conforming product with the qualified Java EE.next application programming interface lies with the cloud provider.

Cloud PaaS tenant

The cloud tenant is a customer, an individual, a business, including a legal entity that subscribes to a Cloud Provider and has one or more applications hosted on the Cloud Provider cloud-computing platform.

Cloud PaaS application submitter

The cloud application submitter is a specific role with the responsibility to upload and install (and of course uninstall) Java EE.next applications to the cloud provider's cloud computing platform. The submitter configures the application and ensures that the resultant artifacts, such as JAR, WAR, and/or EAR are suitable for the target cloud computing platform.

The standard makes a differentiation between a tenant and submitter, because they may not share the same position in some circumstances. An example of this could be a software house that provides dedicated over-the-counter services to a business. The business customer ultimately could choose a third-party supplier to deliver cloud computing services with an existing business application.

Cloud PaaS account manager

The cloud account manager is responsible for managing the account of the tenant. This person decides whether to allow scaling of resources across the cloud computing platform. The account manager will buy resources on expected sales days, and reduce resources in the holiday season, when business is off-peak. This role is a further indirection for the purpose of business. An account manager could be a third-party business managing resources for two or more tenants, for example.

Cloud PaaS application administrator

The cloud application administrator is a further distinctive role for maintaining the availability of the business customer's application as it is running inside a cloud computing environment. The role is different from a system administrator as it has a smaller subset of responsibility. The application administrator ensures the tenant is running by monitoring resources such as memory, disk space, and network bandwidth. There might be limits imposed by the cloud provider, which is responsible for the cloud `Java EE.next` product, to restrict the business tenants from interfering with each other. The very likely restrictions would be security imposed on operating system permission, database table access, and network ports.

A future `Java EE.next` specification should reduce the role types and retire redundant roles. In my opinion, the application assembler and tool provider are no longer relevant. I do think that a cloud PaaS resource manager is an extra role that could be useful. Businesses will want to monitor, manage, and control how much money and dedicated hardware goes into supporting a scalable application.

Java enterprise providers in the Cloud

At the time of writing, there are only two Java EE product providers so far, GlassFish application server, which is the reference implementation for Java EE 7, and also Red Hat JBoss application server.

Following are the current providers in the cloud-computing arena that have varying degrees of conformation to the Java EE specification. All of them allow Java application to be deployed from their platform through a bundling service, which might be a WAR and/or EAR.

In some cases, there are alternative means to deploy an application other than a Java Archive format, which demonstrates the split between the application component provider, deployer and the tool provider. Heroku permits and prefers application to be deployed using the Git tool, which is an open source distributed version control tool.

This is the view of the cloud providers a couple of months after Java EE 7 was released:

Product Name	Vendor	Java EE 7 Support	Java EE Conformance	Notes
Amazon Beanstalk	Amazon	No (Push deployment and changes with proprietary Amazon tools.)	Amazon supports Java 1.5 upwards. Deployment through simple WAR files. PaaS is through specific Apache Tomcat extension. Only Java Servlet 2.2, Java Server Pages support is available.	AWS might allow SEAM Framework and JSF to run as separate JAR uploaded to WEF-INF/lib. Beanstalk supports all popular Amazon services, Elastic Compute Cloud (EC2), Map Reduce, Simple Storage Service (S3), Simple DB, and so on.
Cloud Bees	Cloud Bees, Inc.	Yes (Push changes with Git. Some SVN support is available.)	A PaaS solution for Java offering complete end-to-end environment. As of June 2013, Cloud Bees offers deployment to GlassFish 4 container in the cloud with MySQL support. Cloud Bees was the first Java EE 7 PaaS solution in a cloud environment.	Cloud Bees is all about supporting building continuous integration/delivery applications in the cloud. If you love Jenkins then you will probably like this offering.
Cloud Foundry	VM Ware	No	Cloud Foundry calls itself a PaaS solution, however it has a certain IaaS style by allowing the customer to choose amongst a set of platforms that includes Spring, Grails, Scala, Play, node.js, Ruby, Rails, Sinatra.	Professional open source PaaS solution. Java is not exclusively supported, whereas Spring Framework is.

Product Name	Vendor	Java EE 7 Support	Java EE Conformance	Notes
Google App Engine	Google	No (Deploy changes with WAR files.)	Java EE 6 support is restricted to Java Persistence Architecture and parts of the Java Servlet. Java Runtime Environment is limited. You must use Google's own thread factory to spawn limited threads.	Restricted PaaS availability. Relatively easy to build, scale, and maintain. Support for Groovy, GWT, Spring Framework.
Heroku PaaS	Heroku	Maybe (Push changes with Git.)	Applications are deployed to the Heroku cloud with embedded server (so called container-less build.) The Java EE 7 reference embedded GlassFish 4server ought to work.	Heroku PaaS expects application to bootstrap a server. Tomcat and Jetty are two servers known to work.
IBM Smart Cloud	IBM	No (IaaS like to push in order to deploy and provision applications.)	Provides a set of technologies with both IaaS and PaaS type capabilities with the concept of supporting a virtual appliance.	Commercial offering is a virtual stack of the application, addition libraries, and the operating system.
Java / PHP PaaS	JElastic	Yes (Push changes with Git and SVN.)	Billed as Java/PHP and cloud hosting platform that can scale any Java/PHP application with any unnecessary code changes.	James Gosling at Liquid Robotics made use of GlassFish 3. There is a high chance that GlassFish 4 will also work.
Oracle Cloud	Oracle	Not Yet*	Since Oracle is backing the GlassFish server and also WebLogic server. At the time of writing Oracle has future plans to provide Java EE 7 and Java SE 6 WebLogic server offerings in the cloud.	The Oracle Cloud is the professional upgrade path from GlassFish, but it may have restrictions.

Product Name	Vendor	Java EE 7 Support	Java EE Conformance	Notes
OpenShift	Red Hat	Yes and Not Yet* (Push changes with Git.)	PaaS solution for developers to build, test, run, and deploy applications. Open Shift does run Glass Fish 4 with extra customization. JBoss WildFly, which is Java EE 7 compliant, is assumed to work Open Shift.	A neat idea of Open Shift is the notion of nodes, gears, brokers, and cartridges. These are pluggable facilities to connect to other popular frameworks.

At the time of writing, only Cloud Bees had announced the availability of a full Java EE 7 PaaS solution. Red Hat offering of **JBoss WildFly** application server is targeted at a fully compliant Java EE 7 application server, which means it will be available to the OpenShift PaaS solution. Red Hat had yet to announce this feature, however. The Oracle Cloud solution, unfortunately, looks like the slowest person on the drive, because it only supported Java SE 6 and Java EE 6 at this time.

It is clear from these offerings in PaaS for Java EE 7 that providers vary in the control of the provisioning. Heroku, in particular, expects your application to bootstrap and start an embedded server, whereas Cloud Bees offers GlassFish 4 managed instance controlled by the cloud provider, but bound by your own administration. JELastic apparently gives the application and the business a lot of control, so much so that it moves closer to the infrastructure as in IaaS. This is probably the reason that *James Gosling* preferred this high degree of control as well as the attractive open source technology of **JELastic** for the Liquid Robotics oceanographic communications and network architecture.

The cloud providers are following the fashion of continuous delivery deployments by supplanting WAR deployment with pushing artifacts through a distributed version control such as Git. For Heroku, who are one of the pioneers of version control deployment, it is their sole mechanism to get an application into their cloud offering. Heroku started as a Ruby-on-Rails vendor because they knew that market very well through the founder and then branch out to other languages including JavaScript with `node.js`, then added Java and even Scala support.

With these differences in provisioning, resources, deployment, and monitoring and even ultimately how does an application scale effectively, it is transparent and comprehensible why these fledgling efforts in the wider cloud computing movements could not be standardized in the Java EE 7 specification and its time frame. Hopefully, PaaS solution providers in the future will get together to agree a standard measure of console management and provisioning in the next Java EE specifications.

Currently, developers write Java EE 7 applications on their own machines or the company's machine and then provision it on to the provider's PaaS environment. In future, this practice could change so that PaaS applications may be developed in the cloud itself.

All of the compatible Java EE 7 PaaS solutions aim to take care of the business infrastructure so that the customer concentrates on their focus around their own business logic. This is the logical part of the business that makes each application unique. The PaaS infrastructure handles high availability, scalability, and on-demand provisioning of resources. The difference between each Java EE 7 PaaS solution currently is how they achieve this provisioning, the limits of their own particular environment, and how much the customer can control the process, and of course the cost of the solution.

Minimize Lock-in



As application architect, you should expect some vendor lock-in. For instance, if you are searching for a PaaS solution for a particular framework or language or even a Java EE 7, then you need to find a PaaS that supports it.

Ask yourself the question, "What do I really have to do, if I especially have to move my business application off that PaaS solution in jig time?"

Summary

There are several Java related products available today for Java developers, designers, and architects. Unfortunately, many of them are proprietary and if you invest your time and business into them, although they advertise as easy to get, they will lock you into the vendor, because you will have to develop against their API in order to get the maximum benefits.

Take, for example, Google App Engine, which defines a white list of the only Java API that you are allowed to develop against. The same conundrum applies to Cloud Foundry and, of course, to close proprietary API inside Microsoft Azure and Amazon Beanstalk.

It is expected that Java EE 8 will be a standard that gives businesses the benefit of portability between cloud computing providers. We, the architects and designers, should see a huge benefit in the economy of business, when Java applications can switch cloud provider in order to save costs and increase performance. It is an idea that must be planned for. Only you, the rest of the community, JCP specification committee, and expert groups for `Java EE.next` can make it happen. So please go and prod your local Java User Group or better yet get in contact with the Expert Group and send them your suggestions, criticisms, and ideas.

The cloud computing mantra is about three items, namely:

- **Provisioning:** This is the ability to deploy an application to multiple servers and automatically associate the application with resources it requires to function through allocation by configuration
- **Elasticity & Automatic Scalability:** This is the ability to scale an application on demand
- **Multi-Tenancy:** This is the ability for a cloud provider to host multiple applications from different business subscribers, who normally are separate legal entities as defined by the governing jurisdiction

As the cloud computing platform takes shape on the server side in the remaining years of this decade, we shall expect a new edition of the Java EE specification.