Software Engineering in the Cloud

Eric M. Dashofy

Abstract The computing infrastructure on which engineers develop and deploy software has evolved significantly in recent years. The rapid growth of cloud computing services mean that infrastructure and platform components are becoming more decentralized (owned by others, often far away from the development or operating organization) and more elastic (with the ability to provision and de-provision them at will). Infrastructure-as-a-Service (IaaS) capabilities provide the raw resources needed to deploy software—computing, storage, and networking. Platform-as-a-Service (PaaS) offerings provide important software components as commodity services—databases, identity and access management, security, analytics, various kinds of middleware, and much more. New virtualization and packaging techniques for software that can take advantage of cloud computing, such as containers, provide new opportunities for rapid and automated testing, deployment, and scaling of software systems. This enables new software delivery models, such as continuous deployment of new software to production environments and frequent, transparent A/B testing of new features. These changes are having an impact on software development environments, as well, with more development tasks and workflow steps moving to the cloud. This chapter will briefly explore these technologies and their relationships to one another, and explore their impacts on the practice of software engineering.

1 Introduction

Developments in the 2000s and 2010s have had a significant effect on how modern software is built and deployed. High-bandwidth, ubiquitous Internet connections and the development of different virtualization technologies and techniques have enabled the emergence of a new set of technologies for software development

E. M. Dashofy
The Aerospace Corporation, El Segundo, CA, USA
e-mail: Eric.M.Dashofy@aero.org

S. Cha et al. (eds.), *Handbook of Software Engineering*,
https://doi.org/10.1007/978-3-030-00262-6_13

and deployment collectively referred to as "cloud computing" technologies [1]. These technologies have created tremendous new opportunities—and challenges—for software engineers.

In this chapter, we will explore cloud computing concepts and technologies, and how these can impact software engineering processes, decisions, and designs. First, we will examine virtualization technologies and how these enable cloud computing. Then, we will walk through the cloud technology stack, covering the three layers of the most widely accepted model of cloud computing: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) [2]. Finally, we will look at how cloud technologies are affecting the process of software development and deployment. While they can be complex, cloud technologies offer a wide variety of new architectural options for software engineers in developing and composing novel applications in an increasingly networked world.

1.1 Example

Throughout this chapter, we will examine how the design of a software application might be affected by, or take advantage of, cloud technologies through the use of a hypothetical example. Consider a company that creates a software product called the Media Manager. The initial version of the Media Manager is a traditional desktop application with a component-based architecture that allows a user to store, catalog, and play back digital media—songs, videos, and so on.

The architecture for the initial version of this application is shown in Fig. 1. This application runs on a single desktop computer. The user interacts with the application through a graphical user interface (GUI). This relies on a database component,

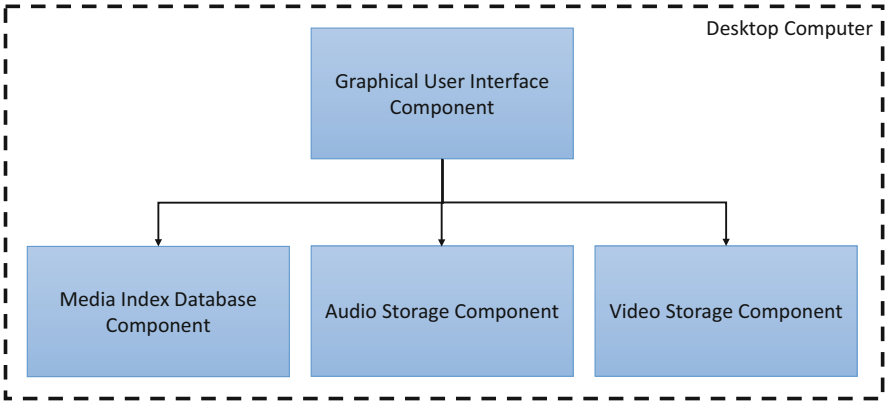

Fig. 1 Architecture for the traditional desktop version of the (hypothetical) Media Manager application

which indexes all the media in the system (for searching and organization), as well as two storage components for storing audio and video files, respectively.

The company wants to evolve the Media Manager application to take advantage of the cloud, and make it available to multiple users as a Web application. We will see how this transition occurs in the following sections.

2 Key Concepts

"Cloud computing" is a general term for technologies where software, services, and infrastructure are made available as commodities over a network. Usually, the provider of these resources is a separate, remote organization from the consumer, and the resources are made available over the Internet, though organizations can develop their own "private" clouds as well (see Sect. 3.1). One of the key advantages of cloud technologies is that they tend to be available *on-demand*, where consumers can allocate and deallocate resources as needs grow and shrink in near-real-time. This property is often called *elasticity*.

2.1 Virtualization

In addition to ubiquitous Internet access, *virtualization* has been the key enabler and driver of the development of cloud technology. Virtualization uses software to create an abstraction layer atop physical computing, storage, and networking resources. Virtual elements (machines, storage services, networks) function almost identically to their physical counterparts, but use software to enable on-demand provisioning and de-provisioning, as well as allow more flexible reconfiguration—also on-demand.

2.1.1 Virtual Computing

Virtual computing is made possible through the use of virtual machines (VMs) [3, 4]. Virtual machines allow a single physical computer to host multiple, isolated virtual computers that share the physical resources of the underlying computer: the processing capability, memory, mounted storage, and peripherals such as network connections or hardware accelerators.

Virtual machines confer several advantages over physical machines. First, they can improve resource utilization: in many applications, computing resources are not fully utilized—processors sit idle much of the time, only part of the computer's memory is used, and network bandwidth use varies. By hosting multiple virtual machines on the same physical machine, those resources can be more fully utilized. Of course, virtual machines hosted together will compete with each other for those

resources when demand is high. Virtualization engineers spend time analyzing this and moving virtual machines among physical machines in order to achieve optimal performance and minimize contention.

Even in applications with high resource requirements where few resources are idle, it can still be advantageous to use virtual machines. In this situation, it is common to see a single virtual machine running on, and using the full resources of, each physical machine in the environment. Virtual machines provide a single, consistent hardware interface to software applications running on them—regardless of the underlying hardware. Imagine a data center with hundreds of physical computers, where a recapitalization program replaces and upgrades one-third of the machines every year. The new computers may be of a different make or model than the older ones in the data center. In a situation without virtualization, software running on those machines might need to undergo significant retesting or rewriting to ensure that it remains compatible with the new hardware. Using virtual machines, it is much less likely that the software will run differently on the new hardware.

Virtual machines can more easily take advantage of increased computing capacity. An older physical machine might have the resources to host four virtual machines at the same time. A newer machine might have the resources to host six or eight. This helps to increase computing "density" in the data center—fewer physical machines providing more capacity, without significant changes to the underlying software.

Virtualization can also increase software reliability. Virtual machines can be replicated, reconstituted, or (with some types of virtualization technology) migrated "live" to other physical machines, usually within a few minutes of the request. This allows software to operate with minimal disruptions when physical machines fail, it allows physical machines to be brought down for maintenance more easily, and it can be used to relocate software services to locations that are more advantageous—for example, locations geographically closer to the demand for those services to reduce network latency.

Each virtual machine has its own operating system (OS) and software configuration. This allows multiple operating systems to run on the same physical hardware, which is useful when developing software systems where application components have different OS requirements. Additionally, although this is less common, a virtual machine can emulate a completely different computing architecture than the underlying physical hardware. This comes at a significant performance cost, but it is extremely useful when an application or component runs only on a legacy platform for which native hardware is no longer manufactured or available (such as DEC Alpha or IBM OS/360). This strategy is often used to substantially increase the life of these applications until they can be rewritten or replaced.

To take full advantage of virtual computing, software engineers must work hand-in-hand with virtualization engineers, and must often develop their applications with virtualization in mind. For example, to enable improved scalability or reliability, a front-end load balancer or proxy may be necessary to route incoming user requests to a family of virtual machines that provide the main software services. Software applications working in an environment with live migration may need

to listen for events preceding and following a migration to checkpoint or restore state (although improvements in virtualization technology are making migration increasingly transparent).

2.1.2 Virtual Storage

Storage virtualization pools many interconnected physical storage devices (e.g., hard disks or solid-state drives) into software-managed virtual storage devices. This permits storage elasticity, where storage can be added or removed from a pool in real time without disrupting operations. It can also improve storage reliability, since pools can be configured to replicate data across physical storage devices in case individual devices fail. More advanced storage virtualization technology will constantly and transparently move data among heterogeneous storage devices to optimize cost and performance. For example, it might move the most-used data to more expensive, highly reliable, high-performance solid-state drives and the least-used data to cheap, less-reliable, lower-performance commodity hard drives.

Storage virtualization generally has minimal impact on software engineers, except to provide additional flexibility and optimization in storage. The exception would be when software has specific performance assumptions or requirements for storage—common in high-performance computing and database applications. In these cases, it is important for software engineers to work with virtualization engineers to ensure that these requirements are met by configuring the virtual storage to allocate the right kind of storage for these parts of the application.

2.1.3 Virtual Networking

Virtual networking is the third major trend in virtualization, but today it is less mature than virtual computing or storage. In traditional networking, machines are physically connected in configurations that permit desired data flows and isolate machines that should not be communicating. Software configurations on elements such as routers, switches, and firewalls restrict or otherwise guide network data flows, and are managed on a device-by-device basis.

With network virtualization technology, sometimes known as *software-defined networking* (*SDN*) [5], all devices are physically interconnected to SDN-capable routers and switches. Configurations, similar to programs written in domain-specific languages, are deployed onto these devices and create virtual networks and routes between the devices. In this way, two endpoints that are physically connected to the same router might be on completely separate virtual networks, able to communicate with other devices on those same virtual networks but not with each other. This isolation can be used to increase security and reduce possible interference between applications. Network virtualization can also be used to increase the reliability of distributed software systems—if an outage occurs, a virtual network can be quickly

reconfigured to transparently route traffic to a backup server or data center without disrupting the application.

As with other virtualization technologies, software and virtualization engineers must work closely together to take maximum advantage of network virtualization. Network virtualization introduces new options for security and reliability that may supplement—or supplant—those traditionally implemented in software. For example, software-defined network rules may eliminate or reduce the need for software firewalls on individual machines in a distributed application. As noted above, software-defined networking can be used to handle certain types of failover—a job traditionally handled by proxies or load balancers. Software engineers must decide carefully where and how to implement these capabilities as part of their applications' architecture.

2.1.4 Example

How might the media manager application evolve to take advantage of virtualization and become a Web application at the same time?

The architecture of this version of the application is shown in Fig. 2. The core architecture remains, with a few changes. The GUI has been replaced by a Web interface, the internal database component has been replaced by an off-the-shelf relational database (like MySQL or Postgres), and the storage components now rely on virtual storage provided in the environment. Since this is still a one-machine

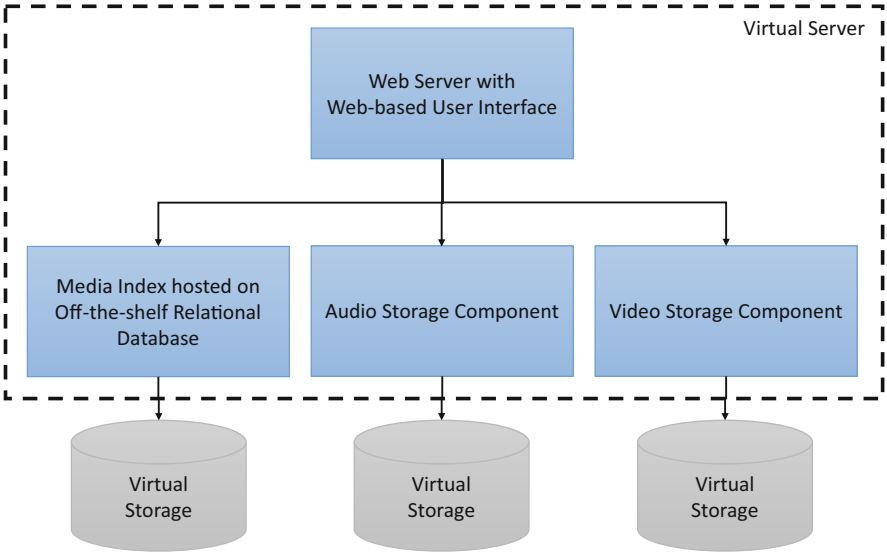

Fig. 2 Architecture of the first Web-based version of the Media Manager application, taking advantage of virtualization

architecture, it doesn't take obvious advantage of virtual networking, though it might use software-defined networking features to limit incoming connections to ensure that users can only connect to the Web interface and not, for example, to an administrative interface on the database component.

2.2 The Three-Layer "as-a-Service" Model of Cloud Computing

Cloud computing starts with the abstractions made available by virtualization, and adds layers that turn them into consumable services. A common model used to discuss cloud services breaks them up into three types: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) [2].

2.2.1 Infrastructure-as-a-Service (IaaS)

Infrastructure-as-a-Service (IaaS) providers offer basic computing resources such as computing, storage, and networking as commodity services. Software applications are then deployed on these infrastructure elements.

Computing is usually provided in the form of virtual machines created and destroyed on demand. Often, some amount of local storage is provided along with the virtual machine. Additional storage can be accessed through IaaS storage services, described below. IaaS computing providers often offer a variety of different (virtual) hardware configurations that differ in the amount of processing power, memory (RAM), storage, network bandwidth, and peripherals available. Configurations for data storage and retrieval might have modest processing capabilities and memory capacity, but large high-performance storage. Configurations for real-time data processing may have fast processors and large amounts of memory, but little available storage.

Virtual storage services are also part of many IaaS offerings. Storage may come in the form of "block storage" or elastic filesystems, where the storage is mounted directly on IaaS computing devices and acts as a network drive. It may also come in the form of "object storage" where individual files are stored and retrieved one at a time based on unique keys. Object storage does not have the nested directory structure that block or filesystem storage has, and is not mounted as a network drive.[1] Instead, files in object storage are stored and retrieved through network-based application programming interfaces (APIs), usually based on the HTTP protocol. For applications that store and retrieve discrete files, like a photo or video management system, object storage can be a good architectural fit.

[1] Some creative software developers have built adapters that can treat certain object storage services as a filesystem, but the performance and latency are usually worse than a filesystem-based service.

Virtual networking services are usually provided as part of IaaS computing offerings, though what capabilities are exposed to the consumer vary from service to service. More advanced IaaS offerings let consumers set up virtual private networks between their IaaS machines and control the configuration and routing of those virtual networks.

Hosted Bare Metal—IaaS Computing on Physical Machines
There's no fundamental reason why IaaS computing services have to use virtual machines; they could also be implemented in a manner where a pool of physical machines are allocated, configured, and deallocated on demand. Sometimes called "hosted bare metal" [6], this approach can be useful in certain situations. Virtual machines introduce a small performance cost that can be undesirable in very high-performance applications, such as high-frequency stock trading or real-time control systems; this can be avoided by hosting directly on physical machines. Additionally, the licensing and management costs of virtual machines might be undesirable in an application where the hardware requirements are consistent and well-known, and the software makes efficient use of the hardware resources.

Example

Let's look at how the Web-based Media Manager application might further evolve to take advantage of IaaS services.

The IaaS-based architecture of the Media Manager application is shown in Fig. 3. Here, each major component of the architecture has been moved to its own

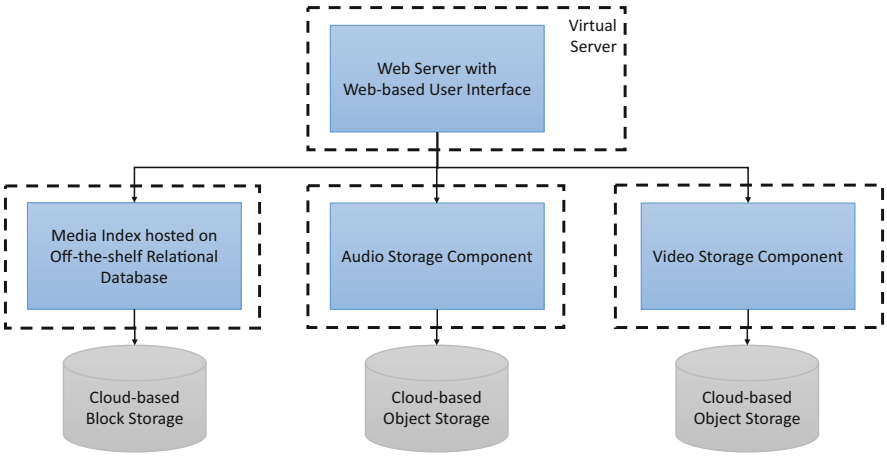

Fig. 3 IaaS-based architecture of the Media Manager application

virtual machine, possibly connected to the others by a software-defined network that carefully controls interconnections within and outside the application. Furthermore, the storage components use cloud-based block storage (for the database) and object storage (for the media). If use of the system increases, some of these virtual machines may be replicated (with appropriate routing or load-balancing infrastructure added) to increase capacity or scale.

Controlling IaaS Resources

When beginners start to use IaaS offerings, they generally provision resources manually, usually through a point-and-click Web-based or command-line interface. Many IaaS providers also expose network-accessible APIs that can be accessed directly by software. Advanced applications that are IaaS-aware can use these APIs to provision and de-provision additional resources on demand when necessary, and this can have a significant impact on a software system's architecture.

One example of an IaaS control API is Amazon's "Query API" [7] for controlling resources in its Elastic Compute Cloud (EC2) service. Commands are sent to this service via specially formatted HTTP requests, similar to those made by a Web browser. The Query API uses individual requests for each command, and implements a simple, consistent request-response protocol. More complex APIs may implement a RESTful interface [8]. An example request:

```
https://ec2.amazonaws.com/?Action=RunInstances
    &ImageId=ami-31814f58
    &InstanceType=m3.medium
    &MaxCount=3
    &MinCount=1
    &KeyName=my-key-pair
    &AUTHPARAMS
```

The main request goes to a well-known, published location (`ec2.amazonaws.com`) and invokes the `RunInstances` command. This command takes a number of parameters that modify how the action works. In this example, they are:

- **ImageId:** The identifier of a previously developed Amazon Machine Image (AMI) that is to be deployed on the newly created virtual machines.
- **MaxCount:** The maximum number of virtual machines to create.
- **MinCount:** The minimum number of virtual machines to create.
- **InstanceType:** What type of virtual machine(s) to create. Amazon offers a catalog of named virtual machine configurations with different capabilities. The `m3.medium` instance specified here has 1 virtual CPU, 3.75 GB of RAM, and 4 GB of storage space. An `m3.xlarge` instance, in contrast, would have 4 virtual CPUs, 15 GB of RAM, and 80 GB of storage (but would cost more).

(continued)

- **KeyName** and **AUTHPARAMS:** Authentication data previously set up by the user to ensure that the user has permission to invoke this operation.

HTTP-based APIs like these are accessible from any programming language or computing environment that can open a TCP connection, making them very widely available. To make software-based management of IaaS resources even more convenient, providers like Amazon often make software development kits (SDKs) available in popular programming languages that wrap these network-based calls in interfaces that are tailored to those programming languages specifically.

2.2.2 Platform-as-a-Service (PaaS)

The definition of what constitutes a Platform-as-a-Service (PaaS) offering has changed over the history of cloud computing. Here, we will try to address how its use and conception has evolved.

In the early days of cloud computing, PaaS referred to a service offering that allows software applications to be uploaded directly to the service for execution on remote computing resources. These PaaS offerings often support specific programming languages such as Java, Python, or JavaScript. Programs running on the platform have access to the runtime library of the programming language, with some exceptions to prevent the programs from using too many resources (frequently, access to APIs that read or write files from the filesystem or open network connections would be restricted). Different platforms may also provide access to standard or custom libraries that provide developers additional capabilities not normally part of the programming language's standard runtime library.

For developers who want to deploy a new application or network service quickly, these PaaS offerings provide several advantages over deploying the same software directly on an IaaS offering. For example, consider a developer that wants to create and deploy a simple Web service, or perform a one-time computation. With IaaS, the developer must allocate a machine, log in, configure the operating system, install any dependencies for the software (Web servers, dependent libraries, and so on), configure those applications, install the software, and run it. Once the software is running, the developer remains responsible for patching and upgrading the operating system and its applications, which is an additional cost. However, with an appropriate PaaS offering, the developer can just upload the software itself directly, and the platform provides all the necessary dependencies.

This kind of PaaS offering is still available from many providers. However, over time, the term "Platform-as-a-Service" expanded to encompass a related set of offerings: network-hosted software services that can be incorporated into applications to provide additional capabilities—but which are not complete applications themselves. These PaaS services can be thought of as the cloud equivalent of

software libraries. Examples of PaaS services available from commercial providers today include:

- **Load-balancing services,** which can provide scalability by balancing incoming requests across a number of machines or endpoints that provide service, and can provide reliability by routing requests around malfunctioning machines
- **Database services**, both relational and object-based or "NoSQL" [9]
- **Identity and access management services**, providing user management, authentication, and authorization
- **Content delivery services**, which make it easier to disseminate content efficiently, with automated caching throughout the network to replicate content in locations close to endpoints that need it
- **Natural language processing services**, which use machine learning and artificial intelligence techniques to parse and produce written and spoken language
- **Image and video processing services,** which provide transformation, processing, and recognition algorithms for images and video
- **Large data processing services,** which provide the ability to apply transformations or analysis algorithms to massive amounts of data efficiently

Both platform- and service-style PaaS offerings can be combined in the creation and deployment of applications—a software engineer might develop an application for deployment on a PaaS platform offering that also uses PaaS services such as a database service and an identity management service for user management and authentication.

For obvious reasons, PaaS can have a tremendous impact on how software engineers architect and design applications. The previously laborious and technical process of setting up and configuring resources, components, and services is minimized. In a PaaS-driven architecture, software integration becomes at least as important as the development itself.

Example

The PaaS-based architecture of the Media Manager application is shown in Fig. 4. This architecture is quite different from the previous architectures. Many application-specific components (blue) have been replaced by PaaS-based services (green). The front-end and user interface of the system now runs on a PaaS-based Web application hosting platform. A PaaS load-balancing server routes incoming requests between instances of the front-end that are dynamically created and replicated by the Web app platform, enhancing the scalability of the application. Databases and storage have been replaced by PaaS services. Note the absence of the block and object storage elements shown in the IaaS architecture—it's possible that the PaaS relational database or media storage services use these storage elements to do their jobs, but this detail is abstracted away from the developer. It's now the PaaS services' job to manage the underlying storage. Here also, a user management and

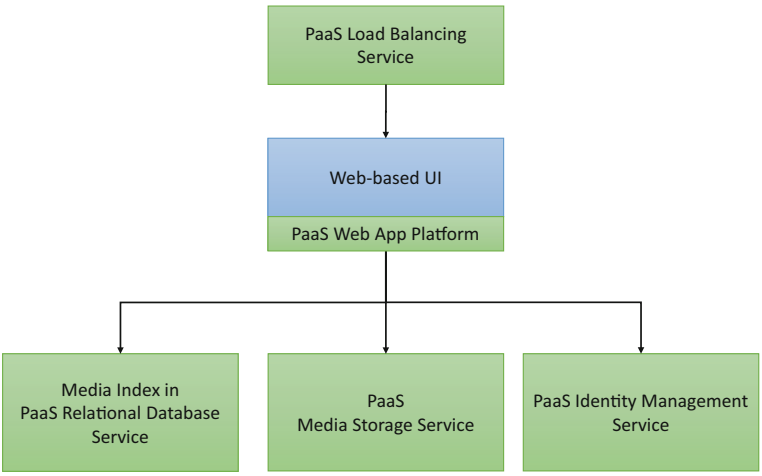

Fig. 4 PaaS-based architecture of the Media Manager application

authentication capability has been added to the system through the incorporation of a PaaS identity management service.

It is interesting to note how much of the application has been replaced with PaaS services. As noted above, this reflects how PaaS can affect architectures in general—by increasing the importance of integration in software engineering when compared to custom development. Here, the custom development is limited to just the unique and integrative elements that tie together the other, commodity services.

2.2.3 Software-as-a-Service (SaaS)

Software-as-a-Service is the practice of offering complete software applications to their users, for use over the network. In today's Internet, these are usually implemented as Web applications that run in a browser. Advances in browser technology have made it possible to implement increasingly capable applications; today, complex systems such as word processors and spreadsheets have been implemented as browser-based applications. The browser's interface and capabilities have not evolved to the point where these applications are equal to their desktop counterparts, but the gap is closing over time.

As a delivery model, SaaS is attractive for several reasons. First, there is no client software to install and update—all deployment of updates happens to the software on the server-side, and users get those updates automatically whenever they access the software. Second, access to the software can be sold as a subscription, creating continuous revenue streams and also virtually eliminating software piracy.

Probably the most important issue facing software engineers developing SaaS applications is the need to carefully design the model by which they will update and evolve the software. SaaS applications tend to be in continuous use. Pushing out an update to a production service while users are actively using it can have negative consequences depending on the implementation of the update. For example, it is unlikely that you want users to start a transaction using one version of your application and finish the transaction using another.

A solution to this problem leverages other cloud elements such as IaaS and PaaS components described above. Using these techniques, a new version of the software could be deployed on infrastructure parallel to the old version, and connected to a load balancer or front-end proxy in an inactive state. When ready to go live, the load balancer or proxy can be programmed to start routing some or all new interactions to the new version of the application. When all transactions have moved to the new version of the application, the old version and its infrastructure can be de-provisioned. This process ensures a smooth transition.

Many mature SaaS vendors take advantage of this strategy for software testing, as well. Using this strategy, it is possible to take a subset of application users, perhaps 5%, and route their requests to a new or different version of the application. Those interactions can be monitored using probes built into the software to test it and compare it to the previous version—for reliability, usability, or performance. With a big enough user base, a cloud provider can push and test updates like this multiple times a day, or try dozens of slight variants of a new feature to perform A/B testing on small (frequently unsuspecting) populations of their users. With feature updates in an SaaS environment being smaller and more frequent than in traditional environments, it's possible many users may not even consciously notice the changes.

Example

The SaaS version of the Media Manager application is shown in Fig. 5. Fewer changes are apparent here—in reality, as a hosted Web application, the PaaS version of the system is already inherently available in the Software-as-a-Service style. To actually offer it as a commercial service, though, the development company must add a few services. Here, we see some additional PaaS elements—one for a paywall, preventing anyone without a paid account from accessing the application, and a credit card processing service allowing people to pay for a new account—integrated with the user and authentication management system, of course.

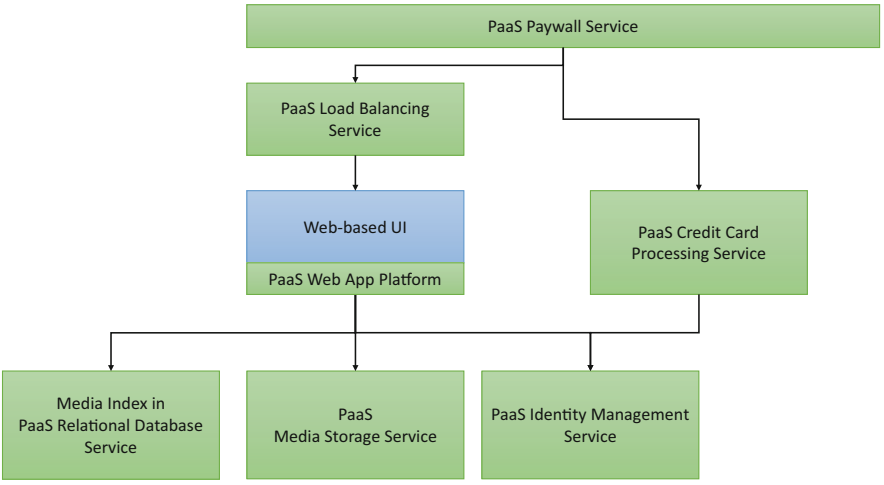

Fig. 5 Architecture of the SaaS version of the Media Manager application

Open-Source Software-as-a-Service

Open-source software projects make their source code available for examination and enhancement by end users. Today, open-source software is found everywhere, and open-source elements can be found in almost every large software system.

There are two general classes of open-source licenses. The first, called "permissive" licenses, includes the MIT, BSD, and Apache Licenses. These licenses permit users to modify the software, but don't create any obligations for the users to make the source code for those modifications available to others. For this reason, components with permissive licenses are frequently integrated into proprietary software systems.

There is a second class of licenses, called "copyleft" licenses, typified by the GNU General Public License (GPL) and its variants. Like permissive licenses, copyleft licenses permit users to modify the software. Unlike permissive licenses, copyleft licenses require that developers of modifications make the source code for those modifications available to other parties whenever they give those parties the compiled, or binary, versions of the software. This prevents companies and individuals from taking a copyleft-licensed piece of software, developing proprietary extensions to that software, and then distributing the extended software while keeping those extensions proprietary and thus unavailable for others to modify.

The copyleft model faced an interesting challenge with the rise of Software-as-a-Service (SaaS) applications. In a SaaS application, users of

(continued)

the software never have direct access to the software at all—neither as a binary nor as source code. This raised the possibility that someone could take copyleft-licensed software, create extensive modifications, and offer (or sell) the result as SaaS without making those modifications available in any form to its users, in conflict with the spirit of the copyleft model.

This resulted in the creation of new copyleft licenses, such as the GNU Affero General Public License (AGPL) [10], which include additional provisions that source code of modifications must be made available to network users of software, and not just those that receive the compiled or binary versions.

3 The Software Economics of Clouds

Before cloud computing, organizations wanting to offer new software services had to buy, configure, and install their own infrastructure in their own facilities, handling all technical and management aspects of this themselves. The costs to manage this, along with the skills required to implement and maintain it, are very high. This made entering the software market a daunting prospect for small companies and startups. Over time, companies began to emerge that specialized in offering some aspects of these services for sale—for example, so-called colocation facilities began to offer space in high-quality, well-managed data centers to small businesses at reasonable costs, but those businesses were still required to configure (and often install) the hardware themselves.

As cloud computing matured, infrastructure, platform, and software services were offered to the public, available to anyone with a credit card. Today, most vendors sell these services commercially in a "pay-as-you-go" fashion. Access to virtual machines is charged by the minute as long as the virtual machine is running—when it is shut down, the charges cease and the computing resources are made available to other customers. Platform services often charge by the volume of transactions or data processed. Software services are usually offered on a "subscription" basis, usually in the form of per-user-per-month charges. This has changed the economics of starting a software company from being a capital investment (with large upfront expenses in equipment and facilities and smaller ongoing maintenance expenses) to an overhead investment (with ongoing charges that scale with the needed capacity).

This shift has been a huge boon to small companies, startups, and innovators. The investment to host a new software application on the Internet is now minuscule, especially when the required capacity is small (as it would be for a brand-new application). The servers, bandwidth, and platform services might be available for a few dollars a day. As demand increases, more cloud resources can be acquired

on demand—again, without significant capital investment. A startup today can go from a one-server prototype application to a 500-server, globally available, high-scalability server farm in hours—a process that would have taken months of time and tens of thousands of dollars before the cloud.

Cloud providers, particularly infrastructure providers, have taken on the capital investment that still needs to occur to manage the thousands of physical machines hosting the tens of thousands of virtual machines available to their customers. This centralization has led to incredible economies of scale, where the biggest providers continue to hyper-optimize their services. Cloud provider data centers are gigantic, and tend to be geographically located close to major power and cooling sources such as dams and rivers. Some providers even work with hardware manufacturers to design custom computers that are built as bare circuit boards—installed in machine racks without cases to lower cost and maximize airflow for heat dissipation. These optimizations drive costs down and capacity up in ways that would be otherwise completely inaccessible to all but the biggest companies.

These commercial clouds are available to the general public and are used by thousands of individuals and companies who share the infrastructure. For this reason, these are often known as "community clouds" and are what people generally think of as being "cloud computing." Commercial community clouds are not the only kind of clouds, however.

3.1 Private Clouds

Clouds enable new levels of decentralization in software engineering—where systems are not just physically distributed across computers or sites, but where the responsibility for parts of the application is held by different parties. When customers (e.g., developers) acquire cloud services, they enter into contracts with cloud providers that establish the responsibilities of both parties. These will include a *service-level agreement* (SLA) [11] that specifies guarantees about characteristics of the cloud service upon which the customer can rely. These may be guarantees about reliability, performance, availability, support, and so on. If these guarantees are violated, the cloud provider will generally compensate the customer monetarily or with free services in the future.

Some companies and organizations want to take advantage of cloud technologies, but have reservations about commercial community clouds. They may be concerned, for example, that:

- Their services will be starved for resources when demand spikes for other customers in the same cloud
- Their data will be processed and stored on computers and storage devices that are used by other companies, and bugs or exploitable flaws in virtualization technology will expose their data to hackers

- They lack sufficient control over the architecture and evolution of the cloud to meet their future business goals
- They need to meet reliability or performance goals unavailable in SLAs from commercial service providers, or that the offered monetary compensation is insufficient to remedy a violation

In these cases, a company or organization may choose to invest in a *private cloud*. A private cloud is owned by a single company and provides infrastructure, platform, or cloud services similar to (and in some cases using the same technology as) a commercial community cloud provider. However, the private cloud resources will be exclusively available for the use of the owning company or organization, under its complete control.

Organizations that do not want to build and maintain a completely private cloud may partner with cloud providers to create a hybrid solution with additional protections for that company. For example, a large cloud provider may offer, for an additional fee, exclusive use of a subset of its cloud resources—guaranteed not to be shared with any other customers. This would be a commercial private (i.e., noncommunity) cloud.

4 Software Development and Deployment in the Cloud

Cloud technologies have, to date, had only a modest impact on the process by which software is developed. Most developers today still use full-featured development environments such as Visual Studio or Eclipse on their desktop to write code, even when that code is for a cloud-based application. They may, however, use SaaS cloud services such as Sourceforge or Github for configuration management, issue tracking, and so on. Recently, a few efforts have created integrated software development environments (IDEs) that are available as SaaS services themselves, running in the Web browser. Similar to other complex SaaS applications like word processors and spreadsheets, the Web-based IDEs don't yet have the features or performance of their desktop counterparts, and haven't yet achieved widespread adoption—though this may change in the future.

The packaging and deployment of software in the cloud is undergoing rapid evolution, however. As noted throughout this chapter, virtualization is a common theme and enabler of software engineering for the cloud. Early in the evolution of the cloud, this development focused on virtual machines, described in detail in Sect. 2.1.1. Recently, a new packaging and virtualization technology has emerged that is rapidly gaining traction among software engineers—container-based virtualization [12].

Containers are a lighter-weight form of virtualization than virtual machines. With virtual machines, each VM is almost completely independent of the others, and each machine can run a different operating system. Containers assume more homogeneity—all containers on the same host machine share an operating system.

Individual containers may use different libraries and binary tools, as long as they are compatible with the shared operating system kernel.

This difference in approach enables containers to scale very differently than virtual machines. A single physical machine will likely host only a handful of virtual machines (especially powerful servers might host 10 or 20 VMs) due to the resource requirements of virtual machines. Virtual machines usually take a few minutes to provision, boot, shut down, and de-provision, just like their physical counterparts. In contrast, a single host machine can support hundreds or thousands of containers, and containers can be created and destroyed quickly—often in sub-second times.

Typically, each container runs a single application and performs a single function. For example, a container might provide a proxy, a load-balancer, a Web service, or a database. A complete software application usually relies on multiple containers coordinating and communicating to implement the total system. This enables new architectures where applications are composed of small, single-function interdependent services that can scale independently from one another. This is known as a *microservices architecture*. In a way, these architectures are modern versions of the time-tested UNIX architecture, where a key design principle is to build small, single-purpose programs that can be easily combined (often in novel ways) to achieve user goals.

Being lighter-weight than virtual machines, it is easy for container management systems to scale the capacity of individual services by spinning up more instances of containers when necessary. Similarly, containers can be easily migrated to, or replicated on, other hosts to improve performance or application reliability. While some virtual-machine-based systems can also have this capability, it is easier and faster to do this with containers.

Containers can also help in application development. New software should never be deployed to production environments directly. Rather, it should first be deployed to staging environments that are (ideally) configured identically to the production environment, where it can be tested. In a virtual-machine-based architecture, keeping staging environments consistent with production environments can be difficult, since any change to the production environment must be made identically to the staging and other development environments. With containers, all dependencies for an application or service are packaged into the container when it is built, and these are independent of all other containers. Effectively, deploying a container deploys the application and its entire stack of dependencies all at once. This significantly reduces the likelihood of differences between production and staging environments. Additionally, since containers are lightweight and many of them can be deployed easily, each developer can effectively have his or her own staging environments, separate from other developers. This reduces the likelihood that developers' work will interfere with each other inadvertently.

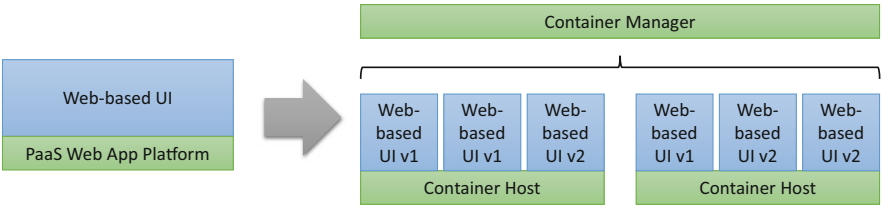

Fig. 6 Replacing the Media Manager's simple Web UI with a scalable, high-reliability container-based version

4.1 Example

Figure 6 shows how a portion of the Media Manager might evolve to take advantage of containers. Here, the single component deployed on a more traditional PaaS platform is replaced by a number of containerized replicas of that component, which may implement different versions (v1 and v2) of the service. An off-the-shelf container manager creates and replicates these across hosts as necessary for scalability and reliability. The application load balancer may be more intelligent, routing some users to version 1 containers and a subset to version 2 containers for A/B testing.

5 Seminal Papers and Genealogy

Cloud computing has been enabled and informed by a number of previous developments that have been combined and evolved into our modern conception of the field. Many of the latest developments, however, have occurred on the Internet and in the commercial world, somewhat separate from the academic conferences and journals that capture significant developments in other computing disciplines. As such, many of the seminal developments and milestones in cloud computing occurred and evolved spontaneously on the Internet, making it difficult to reference particular sources or works.

5.1 Foundations of Cloud Computing

As noted above, virtualization is a key enabler of cloud computing. The notion of sharing computing resources among multiple tasks to make efficient use of those resources goes back to the earliest days of computing mainframes. The notion of virtual machines can be traced back to the mid-1960s, with the IBM CP-40 and CP-67 computers and the CP/CMS operating system. Creasy [3] describes these early developments.

By the 1990s, personal computing platforms (particularly, but not limited to the Intel instruction-set-based x86 platforms) were growing powerful enough that hosting multiple virtual machines on the same physical machine was an increasingly attractive possibility, from an efficiency and economic perspective. Vendors like VMware worked through the technical challenges required to fully virtualize the x86 platform. Marshall [4] provides a high-level (and somewhat VMware-centric) retrospective on these developments. Later technologies, like Xen, described by Barham et al. [13], have made virtualization on commodity platforms much more efficient by reducing the overhead introduced by virtualization.

5.2 Precursors to Cloud Computing

Like virtualization, the notion of computing as a utility or service was conceptualized in the earliest days of the field. In 1961, at the MIT Centennial, John McCarthy noted:

> If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility... The computer utility could become the basis of a new and important industry.

Garfinkel and Abelson [14] expand on this discussion of utility computing.

Multiple users sharing computing resources was enabled by a huge variety of "time sharing" systems that relied on central computing resources accessed via remote terminals. One of the earliest and most seminal descriptions of the UNIX operating system [15] described it as a time-sharing system.

Utility computing was an inspiration for, and coevolved with, the notion of Grid Computing. Grid computing assembles computing, storage, and network resources from multiple locations to solve computing problems bigger than any individual computer or cluster could solve. Many of the goals and benefits of grid computing overlapped with those of cloud computing (shared pools of resources that can be allocated on demand and then released for others' use), the technology and capabilities of grid computing were largely used by the scientific and engineering communities to assemble virtual supercomputers. Foster and Kesselman [16] describe Grid Computing in their seminal book on the topic, now in its second major edition.

Utility computing and grids are largely precursors to Infrastructure-as-a-Service cloud computing, there are also precursors for Platform-as-a-Service. A key technical development that enabled the Google search engine was the development of a general-purpose map-reduce service implemented atop a vast array of commodity PCs. This effectively became an elastic platform for running distributed computing jobs that can be decomposed into the map-reduce algorithmic style, and is documented by Dean and Ghemawat [17]. Based on this work, others built open, nonproprietary versions of the technology, most popularly Hadoop [18, 19].

5.3 Cloud Computing

The evolution of the precursors noted above into what we call "cloud computing" today was gradual. Many cite the introduction of Amazon's Elastic Compute Cloud (EC2) service in 1996 as an early milestone in this transition.

Barr [20] made the announcement of the service's availability (in "beta") in 2006. As the field matured, in 2009 Armbrust, Fox, and a number of colleagues published a retrospective [1] on the early years of cloud computing. Much of this paper is dedicated to identifying fundamental and essential challenges in cloud computing, such as service availability, data confidentiality and integrity, performance unpredictability, and others. These remain key challenges in the field today.

A highly cited, succinct glossary of cloud computing terminology is provided in the NIST definition of cloud computing [2]. This lays out what cloud computing is, and the distinction between infrastructure, platform, and software-as-a-service. The notion of Platform-as-a-Service captured in the NIST definition reflects the early conception of that concept, where users deploy applications on a programming-language/operating-system runtime environment, rather than the evolving definition where the "platform" constitutes a set of generic (and sometimes domain-specific) services that can be integrated into applications.

Platform-as-a-Service offerings emerged following Infrastructure-as-a-Service cloud computing. Wardley [21] describes the 2005 emergence of a platform called Zimki. Google's App Engine [22] followed in April 2008, and its release was a milestone event similar to the EC2 release in 2006.

The history of Software-as-a-Service offerings is harder to capture. Shared, central hosting of applications dates back to the earliest days of computing, as noted above. In the 1990s, this spawned an industry of Application Service Providers (ASPs) that gradually evolved to host their services on the Web as the Web grew in popularity in the late 1990s. Bianchi [23] looks back at this early era of ASPs. Hotmail was an early, popular software-as-a-service offering that provided email, hosted remotely, via the Web. Craddock [24] captures the history of this early SaaS offering. Another early milestone in SaaS was the emergence of Salesforce.com, described by McCarthy [25], which provided customer relationship management and other capabilities entirely as a service with the slogan "No Software" (i.e., no software that needs to be deployed to the desktop).

5.4 Related Concepts

One of the key challenges in cloud computing is ensuring that you "get what you pay for" in terms of performance, availability, and so on. The cloud service provider and consumer negotiate this via service-level agreements (SLAs) that describe the expected characteristics of the service, with specific penalties if those characteristics

are not achieved. These sorts of agreements are effectively contracts, similar to other business contracts that have existed throughout history. Verma [11] published an early and frequently cited paper about service-level agreements specifically on networks that predates most of cloud computing. Later authors, such as Keller and Ludwig [26] and Lamanna et al. [27] describe specific computer-based languages for capturing service-level agreements.

6 Conclusions

Cloud computing provides a number of opportunities and challenges for software engineers. A few opportunities include:

- **Elasticity:** The ability to grow and shrink the amount of resources used by an application as demand changes, only paying for the resources used. This dramatically lowers startup costs to field a new application, and provides unprecedented opportunities to scale up quickly without capital investment.
- **Economies of scale:** Service costs are lower than building your own infrastructure since central providers can hyper-optimize their environments.
- **Extensive, powerful platform services:** Substantial, complex application components are now available as scalable platform services that can be called by your application as needed, reducing the amount of custom code needed to build a new application.
- **Reliability:** By taking advantage of the ability to replicate or migrate applications to different cloud data centers, the failure of any individual hardware component—or even an entire data center—can be handled while avoiding service disruptions.
- **Agility:** The ability to configure and reconfigure cloud applications quickly allows frequent deployments of updates, as well as partial deployments of new features for usability, performance, or reliability testing in the field.

6.1 Key Challenges

Many of the key challenges of using cloud computing were identified in its early days, and continue to represent areas where software designers and architects should pay close attention.

Many of the key challenges of moving to the cloud stem from **decentralization**. The users and providers of cloud services generally come from different organizations. Moving software services to the cloud means that the users must cede some amount of control to provider organizations, and trust them to provide service that is good enough to meet software engineers' business goals. Risks associated with decentralization include:

- **Performance risks:** Cloud providers generally make guarantees about the capacity of their services, and may make guarantees in their SLAs about availability. Performance guarantees are less common. With multi-tenancy, the behavior of one tenant can affect the performance of other tenant's applications. These concerns can be difficult to mitigate, but several strategies exist: find a provider who will make performance guarantees in their SLA, distribute your application across multiple cloud providers (or multiple "zones" of a single provider) and so on. Generally, all these solutions require more resources, so a cost–benefit trade-off must be made.
- **Availability risks:** When hosting a service in the cloud, an outage at the cloud provider means an outage for your hosted service. Every year, there are a few major cloud outages that affect entire services or regions [28]. Mature cloud providers offer multiple "availability zones" (effectively, geographically distributed data centers) to mitigate the risk of an outage at any one zone. However, deploying into multiple availability zones usually means additional costs, and the outage of an entire availability zone can stress alternative zones to the point of breaking.
- **Network risks:** Hosting critical service over a network connection (with distances of hundreds or thousands of miles between consumers and providers) can introduce risks due to varying, and sometimes limited, network bandwidth and latency. Users with particular concerns can, in some cases, contract with cloud and telecommunications providers to set up private or dedicated network circuits to mitigate this risk. Some cloud providers even have a service where customers can ship physical disks to the cloud provider to load or retrieve large amounts of data rather than transferring it over the network [29].
- **Reputation risks:** A security compromise or reliability problem with a cloud provider could negatively impact the reputation of customers that use that provider. Cloud service users should always evaluate their security needs and posture, and layer additional measures on top of the cloud provider's where necessary.

Another set of risks, not specifically related to decentralization, are **complexity risks.** Developing an architecture that takes full advantage of the scalability, elasticity, and agility of the cloud can be daunting. There are dozens of individual technologies to learn and integrate, and these are evolving quickly. Different cloud providers offer these services in slightly different ways, and few standards exist.

Obviously, cloud technologies are not suitable for every problem. They generally assume reliable, constant network connectivity is available, which is not the case in embedded or highly secure applications (although this is changing somewhat; see the discussion on the Internet of Things, below). For applications where security or performance are critical, the risks of trusting a third-party provider may be too great. Even then, developers have the option of creating a private cloud to take advantage of some of the advantages of cloud computing in general—but the cost and complexity of doing so may not be worth the benefit.

Like any major architectural decision, whether and how much to take advantage of the cloud is part of the overall process of software systems engineering. The advantages and disadvantages of different approaches and providers must be carefully considered. Regardless, cloud technologies have given software engineers a plethora of new architectural options and services that can be used to rapidly develop and compose applications and make them available to a global audience, scaling as necessary to meet demand.

6.2 Future Directions

It is difficult to predict how cloud computing will continue to affect software engineering in the future. One clear trend is the emergence of the "Internet of Things" (IoT) [30]. The expectation is that an increasing number of noncomputing devices (home appliances, consumer products, and so on) will gain the ability to connect to the Internet through the addition of low-power, cheap, embedded computers connected through wireless interfaces (WiFi, Bluetooth). Since the computing capabilities of the individual "things" will be limited, they will rely more and more on cloud services for their capability. IoT combines many of the challenges of cloud computing development (many noted above) with the challenges of embedded systems development: How can the software on the endpoint devices be secured? Patched? Updated? What happens to those endpoints if the cloud service goes down or becomes unavailable?

A related trend is the use of cloud resources to perform analysis of huge data sets, a field colloquially known as "Big Data" [31]. As the Internet of Things grows, there will be many more devices that are potential data sources—many IoT devices act as sensors of various kinds, and feed sensed data into cloud services. Big Data analytics can be used to analyze these large data sets, and the cloud provides the resources to elastically provision the amount of computing resources needed to do big data without major capital investment, putting these kinds of capabilities into the hands of individuals and organizations of all sizes. These analyses can be used to optimize user experiences, predict behavior, and for purposes like marketing and advertising. Privacy and confidentiality will be growing challenges in the IoT and Big Data era.

References

1. Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., et al.: Above the clouds: A Berkeley view of cloud computing. Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Rep. UCB/EECS, 28(13) (2009)
2. Mell, P., Grance, T.: The NIST definition of cloud computing. NIST Publication SP 800-145 (2011). https://csrc.nist.gov/publications/detail/sp/800-145/final

3. Creasy, R.J.: The origin of the VM/370 time-sharing system. IBM J. Res. Dev. **25**(5), 483–490 (1981)
4. Marshall, D.: Understanding Full Virtualization, Paravirtualization, and Hardware Assist. VMWare White Paper (2007)
5. Kreutz, D., Ramos, F.M., Verissimo, P.E., Rothenberg, C.E., Azodolmolky, S., Uhlig, S.: Software-defined networking: A comprehensive survey. Proc. IEEE. **103**(1), 14–76 (2015)
6. Bridgwater, A.: What is bare-metal cloud? Computer Weekly Application Developer Network (2013). http://www.computerweekly.com/blog/CW-Developer-Network/What-is-bare-metal-cloud
7. Amazon.com: Using the Query API (2017). http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Using_the_Query_API.html
8. Fielding, R.T., Taylor, R.N.: Principled design of the modern Web architecture. ACM Trans. Internet Technol. **2**(2), 115–150 (2002)
9. Cattell, R.: Scalable SQL and NoSQL data stores. ACM SIGMOD Rec. **39**(4), 12–27 (2011)
10. Free Software Foundation: Why the Affero GPL (2015). https://www.gnu.org/licenses/why-affero-gpl.en.html
11. Verma, D.C.: Supporting Service Level Agreements on IP Networks. MacMillan Technical Publishing, Basingstoke (1999)
12. Soltesz, S., Pötzl, H., Fiuczynski, M.E., Bavier, A., & Peterson, L.:. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In: ACM SIGOPS Operating Systems Review, vol. 41, no. 3, pp. 275–287. ACM (2007)
13. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., et al.: Xen and the art of virtualization. In: ACM SIGOPS Operating Systems Review, vol. 37, no. 5, pp. 164–177. ACM (2003, October)
14. Garfinkel, S., Abelson, H.: Architects of the Information Society: 35 Years of the Laboratory for Computer Science at MIT. MIT Press, Cambridge, MA (1999)
15. Ritchie, O.M., Thompson, K.: The UNIX Time-Sharing System. Bell Syst. Tech. J. **57**(6), 1905–1929 (1978)
16. Foster, I., Kesselman, C. (eds.): The Grid 2: Blueprint for a New Computing Infrastructure. Elsevier (2003)
17. Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. Commun. ACM. ***51***(1), 107–113 (2008)
18. Bialecki, A., Cafarella, M., Cutting, D., O'Malley, O.: Hadoop: A Framework for Running Applications on Large Clusters Built of Commodity Hardware. http://hadoop.apache.org/
19. White, T.: Hadoop: The definitive guide. O'Reilly Media, Sebastopol, CA (2012)
20. Barr, J.: Amazon EC2 Beta. Amazon AWS Blog (2006). https://aws.amazon.com/blogs/aws/amazon_ec2_beta/
21. Wardley, S.: On open source, gameplay and cloud. "Bits or Pieces" blog (2015). Archived at http://web.archive.org/web/20160308014753/http://blog.gardeviance.org/2015/02/on-open-source-gameplay-and-cloud.html
22. Google: App Engine. https://cloud.google.com/appengine/
23. Bianchi, A.: Upstarts: ASPs. INC Magazine (2000). https://www.inc.com/magazine/20000401/18093.html
24. Craddock, D. (2010). A Short History of Hotmail. Archived at https://web.archive.org/web/20100426043450/http://windowsteamblog.com/blogs/windowslive/archive/2010/01/06/a-short-history-of-hotmail.aspx
25. McCarthy, B. (2016). A Brief History of Salesforce.com. http://www.salesforceben.com/brief-history-salesforce-com/
26. Keller, A., Ludwig, H.: The WSLA framework: Specifying and monitoring service level agreements for web services. J. Netw. Syst. Manag. **11**(1), 57–81 (2003)
27. Lamanna, D.D., Skene, J., Emmerich, W.: SLAng: A Language for Service Level Agreements. IEEE Computer Society Press, Los Alamitos, CA (2003)
28. Tsidulko, J.: The 10 Biggest Cloud Outages of 2016. CRN (2016) http://www.crn.com/slide-shows/cloud/300083247/the-10-biggest-cloud-outages-of-2016.htm

29. Barr, J.: AWS Import/Export: Ship Us That Disk! Amazon AWS Blog (2009). https://aws.amazon.com/blogs/aws/send-us-that-data/
30. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of Things (IoT): A vision, architectural elements, and future directions. Futur. Gener. Comput. Syst. **29**(7), 1645–1660 (2013)
31. Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., Byers, A.H.: Big data: The next frontier for innovation, competition, and productivity. McKinsey Global Institute Report (2011). https://bigdatawg.nist.gov/pdf/MGI_big_data_full_report.pdf