FUTURE VISION BIE

One Stop for All Study Materials

& Lab Programs

Future Vision

By K B Hemanth Raj

Scan the QR Code to Visit the Web Page

Or

Visit : https://hemanthrajhemu.github.io

Gain Access to All Study Materials according to VTU,
CSE – Computer Science Engineering,
ISE – Information Science Engineering,
ECE - Electronics and Communication Engineering
& MORE...

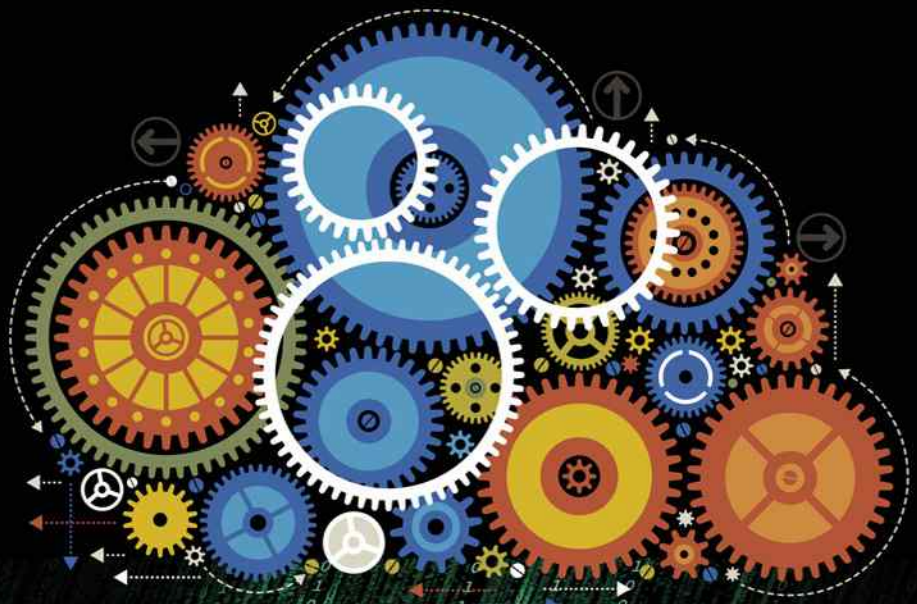
Join Telegram to get Instant Updates: https://bit.ly/VTU_TELEGRAM

Contact: MAIL: futurevisionbie@gmail.com

INSTAGRAM: www.instagram.com/hemanthraj_hemu/

INSTAGRAM: www.instagram.com/futurevisionbie/

WHATSAPP SHARE: https://bit.ly/FVBIESHARE



MASTERING CLOUD COMPUTING

FOUNDATIONS AND APPLICATIONS PROGRAMMING

MORGAN KAUFMANN

Rajkumar Buyya, Christian Vecchiola, S. Thamarai Selvi

CHAPTER

Cloud Computing Architecture

4

The term *cloud computing* is a wide umbrella encompassing many different things; lately it has become a buzzword that is easily misused to revamp existing technologies and ideas for the public. What makes cloud computing so interesting to IT stakeholders and research practitioners? How does it introduce innovation into the field of distributed computing? This chapter addresses all these questions and characterizes the phenomenon. It provides a reference model that serves as a basis for discussion of cloud computing technologies.

4.1 Introduction

Utility-oriented data centers are the first outcome of cloud computing, and they serve as the infrastructure through which the services are implemented and delivered. Any cloud service, whether virtual hardware, development platform, or application software, relies on a distributed infrastructure owned by the provider or rented from a third party. As noted in the previous definition, the characterization of a cloud is quite general: It can be implemented using a datacenter, a collection of clusters, or a heterogeneous distributed system composed of desktop PCs, workstations, and servers. Commonly, clouds are built by relying on one or more datacenters. In most cases hardware resources are virtualized to provide isolation of workloads and to best exploit the infrastructure. According to the specific service delivered to the end user, different layers can be stacked on top of the virtual infrastructure: a virtual machine manager, a development platform, or a specific application middleware.

As noted in earlier chapters, the cloud computing paradigm emerged as a result of the convergence of various existing models, technologies, and concepts that changed the way we deliver and use IT services. A broad definition of the phenomenon could be as follows:

> *Cloud computing is a utility-oriented and Internet-centric way of delivering IT services on demand. These services cover the entire computing stack: from the hardware infrastructure packaged as a set of virtual machines to software services such as development platforms and distributed applications.*

This definition captures the most important and fundamental aspects of cloud computing. We now discuss a reference model that aids in categorization of cloud technologies, applications, and services.

4.2 The cloud reference model

Cloud computing supports any IT service that can be consumed as a utility and delivered through a network, most likely the Internet. Such characterization includes quite different aspects: infrastructure, development platforms, application and services.

4.2.1 Architecture

It is possible to organize all the concrete realizations of cloud computing into a layered view covering the entire stack (see Figure 4.1), from hardware appliances to software systems. Cloud resources are harnessed to offer "computing horsepower" required for providing services. Often, this layer is implemented using a datacenter in which hundreds and thousands of nodes are stacked together. Cloud infrastructure can be heterogeneous in nature because a variety of resources, such as clusters and even networked PCs, can be used to build it. Moreover, database systems and other storage services can also be part of the infrastructure.

The physical infrastructure is managed by the core middleware, the objectives of which are to provide an appropriate runtime environment for applications and to best utilize resources. At the bottom of the stack, virtualization technologies are used to guarantee runtime environment customization, application isolation, sandboxing, and quality of service. Hardware virtualization is most commonly used at this level. Hypervisors manage the pool of resources and expose the distributed infrastructure as a collection of virtual machines. By using virtual machine technology it is possible to finely partition the hardware resources such as CPU and memory and to virtualize specific devices, thus meeting the requirements of users and applications. This solution is generally paired

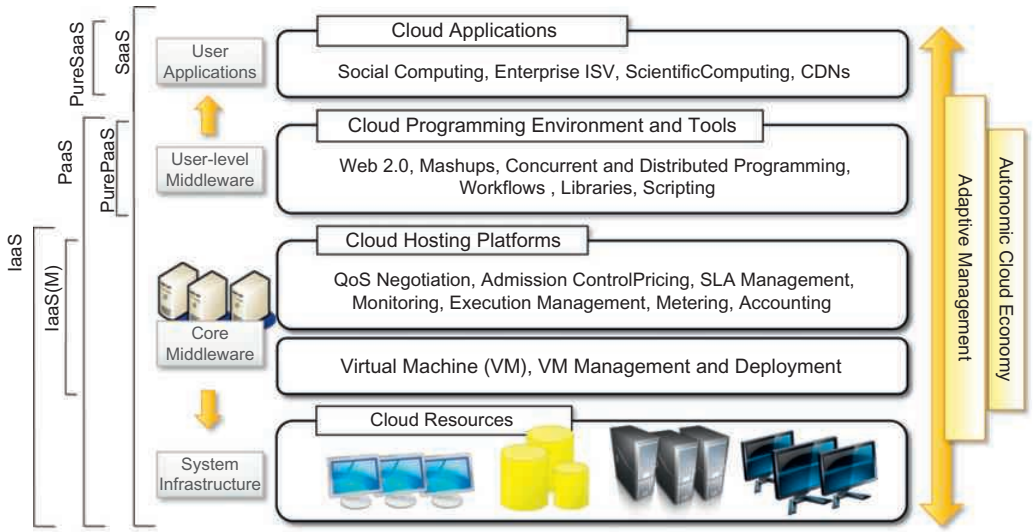


FIGURE 4.1

The cloud computing architecture.

with storage and network virtualization strategies, which allow the infrastructure to be completely virtualized and controlled. According to the specific service offered to end users, other virtualization techniques can be used; for example, programming-level virtualization helps in creating a portable runtime environment where applications can be run and controlled. This scenario generally implies that applications hosted in the cloud be developed with a specific technology or a programming language, such as Java, .NET, or Python. In this case, the user does not have to build its system from bare metal. Infrastructure management is the key function of core middleware, which supports capabilities such as negotiation of the quality of service, admission control, execution management and monitoring, accounting, and billing.

The combination of cloud hosting platforms and resources is generally classified as a *Infrastructure-as-a-Service (IaaS)* solution. We can organize the different examples of IaaS into two categories: Some of them provide both the management layer and the physical infrastructure; others provide only the management layer (*IaaS (M)*). In this second case, the management layer is often integrated with other IaaS solutions that provide physical infrastructure and adds value to them.

IaaS solutions are suitable for designing the system infrastructure but provide limited services to build applications. Such service is provided by cloud programming environments and tools, which form a new layer for offering users a development platform for applications. The range of tools include Web-based interfaces, command-line tools, and frameworks for concurrent and distributed programming. In this scenario, users develop their applications specifically for the cloud by using the API exposed at the user-level middleware. For this reason, this approach is also known as *Platform-as-a-Service (PaaS)* because the service offered to the user is a development platform rather than an infrastructure. PaaS solutions generally include the infrastructure as well, which is bundled as part of the service provided to users. In the case of *Pure PaaS*, only the user-level middleware is offered, and it has to be complemented with a virtual or physical infrastructure.

The top layer of the reference model depicted in Figure 4.1 contains services delivered at the application level. These are mostly referred to as *Software-as-a-Service (SaaS)*. In most cases these are Web-based applications that rely on the cloud to provide service to end users. The horsepower of the cloud provided by IaaS and PaaS solutions allows independent software vendors to deliver their application services over the Internet. Other applications belonging to this layer are those that strongly leverage the Internet for their core functionalities that rely on the cloud to sustain a larger number of users; this is the case of gaming portals and, in general, social networking websites.

As a vision, any service offered in the cloud computing style should be able to adaptively change and expose an autonomic behavior, in particular for its availability and performance. As a reference model, it is then expected to have an adaptive management layer in charge of elastically scaling on demand. SaaS implementations should feature such behavior automatically, whereas PaaS and IaaS generally provide this functionality as a part of the API exposed to users.

The reference model described in Figure 4.1 also introduces the concept of *everything as a Service (XaaS)*. This is one of the most important elements of cloud computing: Cloud services from different providers can be combined to provide a completely integrated solution covering all the computing stack of a system. IaaS providers can offer the bare metal in terms of virtual machines where PaaS solutions are deployed. When there is no need for a PaaS layer, it is possible to directly customize the virtual infrastructure with the software stack needed to run applications. This is the case of virtual Web farms: a distributed system composed of Web servers, database

servers, and load balancers on top of which prepackaged software is installed to run Web applications. This possibility has made cloud computing an interesting option for reducing startups' capital investment in IT, allowing them to quickly commercialize their ideas and grow their infrastructure according to their revenues.

Table 4.1 summarizes the characteristics of the three major categories used to classify cloud computing solutions. In the following section, we briefly discuss these characteristics along with some references to practical implementations.

4.2.2 Infrastructure- and hardware-as-a-service

Infrastructure- and Hardware-as-a-Service (IaaS/HaaS) solutions are the most popular and developed market segment of cloud computing. They deliver customizable infrastructure on demand. The available options within the IaaS offering umbrella range from single servers to entire infrastructures, including network devices, load balancers, and database and Web servers.

The main technology used to deliver and implement these solutions is hardware virtualization: one or more virtual machines opportunely configured and interconnected define the distributed system on top of which applications are installed and deployed. Virtual machines also constitute the atomic components that are deployed and priced according to the specific features of the virtual hardware: memory, number of processors, and disk storage. IaaS/HaaS solutions bring all the benefits of hardware virtualization: workload partitioning, application isolation, sandboxing, and hardware tuning. From the perspective of the service provider, IaaS/HaaS allows better exploiting the IT infrastructure and provides a more secure environment where executing third party applications. From the perspective of the customer it reduces the administration and maintenance cost as well as the capital costs allocated to purchase hardware. At the same time, users can take advantage of the full customization offered by virtualization to deploy their infrastructure in the cloud; in most cases virtual machines come with only the selected operating system installed and the system can be

Table 4.1 Cloud Computing Services Classification

Category	Characteristics	Product Type	Vendors and Products
SaaS	Customers are provided with applications that are accessible anytime and from anywhere.	Web applications and services (Web 2.0)	SalesForce.com (CRM) Clarizen.com (project management) Google Apps
PaaS	Customers are provided with a platform for developing applications hosted in the cloud.	Programming APIs and frameworks Deployment systems	Google AppEngine Microsoft Azure Manjrasoft Aneka Data Synapse
IaaS/HaaS	Customers are provided with virtualized hardware and storage on top of which they can build their infrastructure.	Virtual machine management infrastructure Storage management Network management	Amazon EC2 and S3 GoGrid Nirvanix

configured with all the required packages and applications. Other solutions provide prepackaged system images that already contain the software stack required for the most common uses: Web servers, database servers, or LAMP[1] stacks. Besides the basic virtual machine management capabilities, additional services can be provided, generally including the following: SLA resource-based allocation, workload management, support for infrastructure design through advanced Web interfaces, and the ability to integrate third-party IaaS solutions.

Figure 4.2 provides an overall view of the components forming an Infrastructure-as-a-Service solution. It is possible to distinguish three principal layers: the *physical infrastructure*, the *software management infrastructure*, and the *user interface*. At the top layer the user interface provides access to the services exposed by the software management infrastructure. Such an interface is

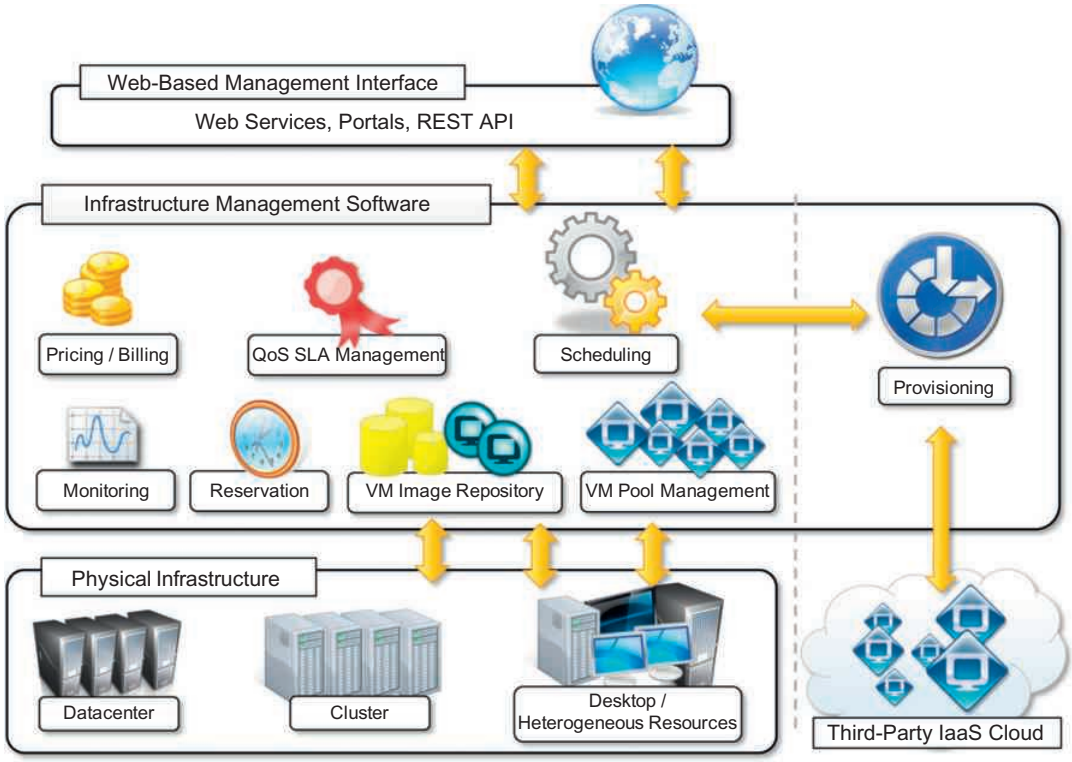


FIGURE 4.2

Infrastructure-as-a-Service reference implementation.

[1] *LAMP* is an acronym for *Linux Apache MySql and PHP* and identifies a specific server configuration running the Linux operating system, featuring Apache as Web server, MySQL as database server, and PHP: Hypertext Preprocessor (PHP) as server-side scripting technology for developing Web applications. LAMP stacks are the most common packaged solutions for quickly deploying Web applications.

generally based on Web 2.0 technologies: Web services, RESTful APIs, and mash-ups. These technologies allow either applications or final users to access the services exposed by the underlying infrastructure. Web 2.0 applications allow developing full-featured management consoles completely hosted in a browser or a Web page. Web services and RESTful APIs allow programs to interact with the service without human intervention, thus providing complete integration within a software system. The core features of an IaaS solution are implemented in the infrastructure management software layer. In particular, management of the virtual machines is the most important function performed by this layer. A central role is played by the scheduler, which is in charge of allocating the execution of virtual machine instances. The scheduler interacts with the other components that perform a variety of tasks:

- The *pricing and billing* component takes care of the cost of executing each virtual machine instance and maintains data that will be used to charge the user.
- The *monitoring* component tracks the execution of each virtual machine instance and maintains data required for reporting and analyzing the performance of the system.
- The *reservation* component stores the information of all the virtual machine instances that have been executed or that will be executed in the future.
- If support for QoS-based execution is provided, a *QoS/SLA management* component will maintain a repository of all the SLAs made with the users; together with the monitoring component, this component is used to ensure that a given virtual machine instance is executed with the desired quality of service.
- The *VM repository* component provides a catalog of virtual machine images that users can use to create virtual instances. Some implementations also allow users to upload their specific virtual machine images.
- A *VM pool manager* component is responsible for keeping track of all the live instances.
- Finally, if the system supports the integration of additional resources belonging to a third-party IaaS provider, a *provisioning* component interacts with the scheduler to provide a virtual machine instance that is external to the local physical infrastructure directly managed by the pool.

The bottom layer is composed of the physical infrastructure, on top of which the management layer operates. As previously discussed, the infrastructure can be of different types; the specific infrastructure used depends on the specific use of the cloud. A service provider will most likely use a massive datacenter containing hundreds or thousands of nodes. A cloud infrastructure developed in house, in a small or medium-sized enterprise or within a university department, will most likely rely on a cluster. At the bottom of the scale it is also possible to consider a heterogeneous environment where different types of resources—PCs, workstations, and clusters—can be aggregated. This case mostly represents an evolution of desktop grids where any available computing resource (such as PCs and workstations that are idle outside of working hours) is harnessed to provide a huge compute power. From an architectural point of view, the physical layer also includes the virtual resources that are rented from external IaaS providers.

In the case of complete IaaS solutions, all three levels are offered as service. This is generally the case with public clouds vendors such as Amazon, GoGrid, Joyent, Rightscale, Terremark, Rackspace, ElasticHosts, and Flexiscale, which own large datacenters and give access to their computing infrastructures using an IaaS approach. Other solutions instead cover only the user interface

and the infrastructure software management layers. They need to provide credentials to access third-party IaaS providers or to own a private infrastructure in which the management software is installed. This is the case with Enomaly, Elastra, Eucalyptus, OpenNebula, and specific IaaS (M) solutions from VMware, IBM, and Microsoft.

The proposed architecture only represents a reference model for IaaS implementations. It has been used to provide general insight into the most common features of this approach for providing cloud computing services and the operations commonly implemented at this level. Different solutions can feature additional services or even not provide support for some of the features discussed here. Finally, the reference architecture applies to IaaS implementations that provide computing resources, especially for the scheduling component. If storage is the main service provided, it is still possible to distinguish these three layers. The role of infrastructure management software is not to keep track and manage the execution of virtual machines but to provide access to large infrastructures and implement storage virtualization solutions on top of the physical layer.

4.2.3 Platform as a service

Platform-as-a-Service (PaaS) solutions provide a development and deployment platform for running applications in the cloud. They constitute the middleware on top of which applications are built. A general overview of the features characterizing the PaaS approach is given in Figure 4.3.

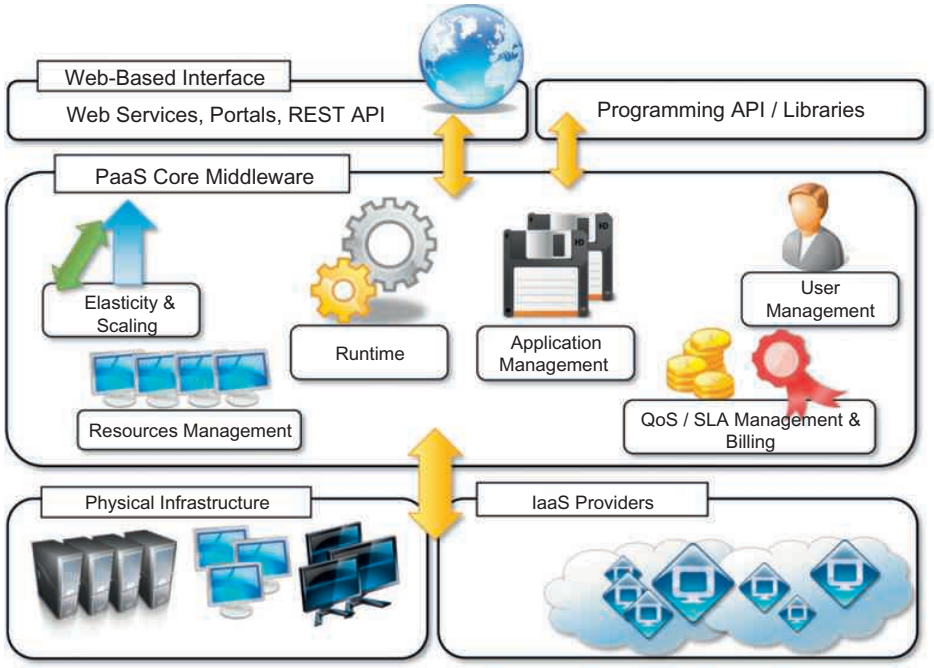


FIGURE 4.3

The Platform-as-a-Service reference model.

Application management is the core functionality of the middleware. PaaS implementations provide applications with a runtime environment and do not expose any service for managing the underlying infrastructure. They automate the process of deploying applications to the infrastructure, configuring application components, provisioning and configuring supporting technologies such as load balancers and databases, and managing system change based on policies set by the user. Developers design their systems in terms of applications and are not concerned with hardware (physical or virtual), operating systems, and other low-level services. The core middleware is in charge of managing the resources and scaling applications on demand or automatically, according to the commitments made with users. From a user point of view, the core middleware exposes interfaces that allow programming and deploying applications on the cloud. These can be in the form of a Web-based interface or in the form of programming APIs and libraries.

The specific development model decided for applications determines the interface exposed to the user. Some implementations provide a completely Web-based interface hosted in the cloud and offering a variety of services. It is possible to find integrated developed environments based on 4GL and visual programming concepts, or rapid prototyping environments where applications are built by assembling mash-ups and user-defined components and successively customized. Other implementations of the PaaS model provide a complete object model for representing an application and provide a programming language-based approach. This approach generally offers more flexibility and opportunities but incurs longer development cycles. Developers generally have the full power of programming languages such as Java, .NET, Python, or Ruby, with some restrictions to provide better scalability and security. In this case the traditional development environments can be used to design and develop applications, which are then deployed on the cloud by using the APIs exposed by the PaaS provider. Specific components can be offered together with the development libraries for better exploiting the services offered by the PaaS environment. Sometimes a local runtime environment that simulates the conditions of the cloud is given to users for testing their applications before deployment. This environment can be restricted in terms of features, and it is generally not optimized for scaling.

PaaS solutions can offer middleware for developing applications together with the infrastructure or simply provide users with the software that is installed on the user premises. In the first case, the PaaS provider also owns large datacenters where applications are executed; in the second case, referred to in this book as *Pure PaaS*, the middleware constitutes the core value of the offering. It is also possible to have vendors that deliver both middleware and infrastructure and ship only the middleware for private installations.

Table 4.2 provides a classification of the most popular PaaS implementations. It is possible to organize the various solutions into three wide categories: *PaaS-I*, *PaaS-II*, and *PaaS-III*. The first category identifies PaaS implementations that completely follow the cloud computing style for application development and deployment. They offer an integrated development environment hosted within the Web browser where applications are designed, developed, composed, and deployed. This is the case of Force.com and Longjump. Both deliver as platforms the combination of middleware and infrastructure. In the second class we can list all those solutions that are focused on providing a scalable infrastructure for Web application, mostly websites. In this case, developers generally use the providers' APIs, which are built on top of industrial runtimes, to develop

Table 4.2 Platform-as-a-Service Offering Classification

Category	Description	Product Type	Vendors and Products
PaaS-I	Runtime environment with Web-hosted application development platform. Rapid application prototyping.	Middleware + Infrastructure Middleware + Infrastructure	Force.com Longjump
PaaS-II	Runtime environment for scaling Web applications. The runtime could be enhanced by additional components that provide scaling capabilities.	Middleware + Infrastructure Middleware Middleware + Infrastructure Middleware + Infrastructure Middleware + Infrastructure Middleware	Google AppEngine AppScale Heroku Engine Yard Joyent Smart Platform GigaSpaces XAP
PaaS-III	Middleware and programming model for developing distributed applications in the cloud.	Middleware + Infrastructure Middleware Middleware Middleware Middleware Middleware	Microsoft Azure DataSynapse Cloud IQ Manjrasof Aneka Apprenda SaaSGrid GigaSpaces DataGrid

applications. Google AppEngine is the most popular product in this category. It provides a scalable runtime based on the Java and Python programming languages, which have been modified for providing a secure runtime environment and enriched with additional APIs and components to support scalability. AppScale, an open-source implementation of Google AppEngine, provides interface-compatible middleware that has to be installed on a physical infrastructure. Joyent Smart Platform provides a similar approach to Google AppEngine. A different approach is taken by Heroku and Engine Yard, which provide scalability support for Ruby- and Ruby on Rails-based Websites. In this case developers design and create their applications with the traditional methods and then deploy them by uploading to the provider's platform.

The third category consists of all those solutions that provide a cloud programming platform for any kind of application, not only Web applications. Among these, the most popular is Microsoft Windows Azure, which provides a comprehensive framework for building service-oriented cloud applications on top of the .NET technology, hosted on Microsoft's datacenters. Other solutions in the same category, such as Manjrasoft Aneka, Apprenda SaaSGrid, Appistry Cloud IQ Platform, DataSynapse, and GigaSpaces DataGrid, provide only middleware with different services. Table 4.2 shows only a few options available in the Platform-as-a-Service market segment.

The PaaS umbrella encompasses a variety of solutions for developing and hosting applications in the cloud. Despite this heterogeneity, it is possible to identify some criteria that are expected to

be found in any implementation. As noted by Sam Charrington, product manager at Appistry.com,[2] there are some essential characteristics that identify a PaaS solution:

- *Runtime framework*. This framework represents the "software stack" of the PaaS model and the most intuitive aspect that comes to people's minds when they refer to PaaS solutions. The runtime framework executes end-user code according to the policies set by the user and the provider.
- *Abstraction*. PaaS solutions are distinguished by the higher level of abstraction that they provide. Whereas in the case of IaaS solutions the focus is on delivering "raw" access to virtual or physical infrastructure, in the case of PaaS the focus is on the applications the cloud must support. This means that PaaS solutions offer a way to deploy and manage applications on the cloud rather than a bunch of virtual machines on top of which the IT infrastructure is built and configured.
- *Automation*. PaaS environments automate the process of deploying applications to the infrastructure, scaling them by provisioning additional resources when needed. This process is performed automatically and according to the SLA made between the customers and the provider. This feature is normally not native in IaaS solutions, which only provide ways to provision more resources.
- *Cloud services*. PaaS offerings provide developers and architects with services and APIs, helping them to simplify the creation and delivery of elastic and highly available cloud applications. These services are the key differentiators among competing PaaS solutions and generally include specific components for developing applications, advanced services for application monitoring, management, and reporting.

Another essential component for a PaaS-based approach is the ability to integrate third-party cloud services offered from other vendors by leveraging service-oriented architecture. Such integration should happen through standard interfaces and protocols. This opportunity makes the development of applications more agile and able to evolve according to the needs of customers and users. Many of the PaaS offerings provide this facility, which is naturally built into the framework they leverage to provide a cloud computing solution.

One of the major concerns of leveraging PaaS solutions for implementing applications is *vendor lock-in*. Differently from IaaS solutions, which deliver bare virtual servers that can be fully customized in terms of the software stack installed, PaaS environments deliver a platform for developing applications, which exposes a well-defined set of APIs and, in most cases, binds the application to the specific runtime of the PaaS provider. Even though a platform-based approach strongly simplifies the development and deployment cycle of applications, it poses the risk of making these applications completely dependent on the provider. Such dependency can become a significant obstacle in retargeting the application to another environment and runtime if the commitments made with the provider cease. The impact of the vendor lock-in on applications obviously varies according to the various solutions. Some of them, such as Force.com, rely on a proprietary runtime framework, which makes the retargeting process very difficult. Others, such as Google AppEngine and Microsoft Azure, rely on industry-standard runtimes but utilize private data storage facilities and

[2]The full detail of this analysis can be found in the Cloud-pulse blog post available at the following address: http://Cloudpulseblog.com/2010/02/the-essential-characteristics-of-paas.

computing infrastructure. In this case it is possible to find alternatives based on PaaS solutions implementing the same interfaces, with perhaps different performance. Others, such as Appistry Cloud IQ Platform, Heroku, and Engine Yard, completely rely on open standards, thus making the migration of applications easier.

Finally, from a financial standpoint, although IaaS solutions allow shifting the capital cost into operational costs through outsourcing, PaaS solutions can cut the cost across development, deployment, and management of applications. It helps management reduce the risk of ever-changing technologies by offloading the cost of upgrading the technology to the PaaS provider. This happens transparently for the consumers of this model, who can concentrate their effort on the core value of their business. The PaaS approach, when bundled with underlying IaaS solutions, helps even small start-up companies quickly offer customers integrated solutions on a hosted platform at a very minimal cost. These opportunities make the PaaS offering a viable option that targets different market segments.

4.2.4 Software as a service

Software-as-a-Service (SaaS) is a software delivery model that provides access to applications through the Internet as a Web-based service. It provides a means to free users from complex hardware and software management by offloading such tasks to third parties, which build applications accessible to multiple users through a Web browser. In this scenario, customers neither need install anything on their premises nor have to pay considerable up-front costs to purchase the software and the required licenses. They simply access the application website, enter their credentials and billing details, and can instantly use the application, which, in most of the cases, can be further customized for their needs. On the provider side, the specific details and features of each customer's application are maintained in the infrastructure and made available on demand.

The SaaS model is appealing for applications serving a wide range of users and that can be adapted to specific needs with little further customization. This requirement characterizes SaaS as a "one-to-many" software delivery model, whereby an application is shared across multiple users. This is the case of CRM[3] and ERP[4] applications that constitute common needs for almost all enterprises, from small to medium-sized and large business. Every enterprise will have the same requirements for the basic features concerning CRM and ERP; different needs can be satisfied with further customization. This scenario facilitates the development of software platforms that provide a general set of features and support specialization and ease of integration of new components. Moreover, it constitutes the perfect candidate for hosted solutions, since the applications delivered to the user are the same, and the applications themselves provide users with the means to shape the

[3]CRM is an acronym for *customer relationship management* and identifies concerns related to interactions with customers and prospect sales. CRM solutions are software systems that simplify the process of managing customers and identifying sales strategies.

[4]ERP, an acronym for *enterprise resource planning*, generally refers to an integrated computer-based system used to manage internal and external resources, including tangible assets, materials, and financial and human resources. ERP software provides an integrated view of the enterprise and facilitates the management of the information flows between business functions and resources.

applications according to user needs. As a result, SaaS applications are naturally multitenant. *Multitenancy*, which is a feature of SaaS compared to traditional packaged software, allows providers to centralize and sustain the effort of managing large hardware infrastructures, maintaining and upgrading applications transparently to the users, and optimizing resources by sharing the costs among the large user base. On the customer side, such costs constitute a minimal fraction of the usage fee paid for the software.

As noted previously (see Section 1.2), the concept of software as a service preceded cloud computing, starting to circulate at the end of the 1990s, when it began to gain marketplace acceptance [31]. The acronym SaaS was then coined in 2001 by the *Software Information & Industry Association (SIIA)* [32] with the following connotation:

> *In the software as a service model, the application, or service, is deployed from a centralized datacenter across a network—Internet, Intranet, LAN, or VPN—providing access and use on a recurring fee basis. Users "rent," "subscribe to," "are assigned," or "are granted access to" the applications from a central provider. Business models vary according to the level to which the software is streamlined, to lower price and increase efficiency, or value-added through customization to further improve digitized business processes.*

The analysis carried out by SIIA was mainly oriented to cover application service providers (ASPs) and all their variations, which capture the concept of software applications consumed as a service in a broader sense. ASPs already had some of the core characteristics of SaaS:

- The product sold to customer is *application access*.
- The application is centrally managed.
- The service delivered is *one-to-many*.
- The service delivered is an integrated solution *delivered on the contract*, which means provided as promised.

Initially ASPs offered hosting solutions for packaged applications, which were served to multiple customers. Successively, other options, such as Web-based integration of third-party application services, started to gain interest and a new range of opportunities open up to independent software vendors and service providers. These opportunities eventually evolved into a more flexible model to deliver applications as a service: the SaaS model. ASPs provided access to packaged software solutions that addressed the needs of a variety of customers. Initially this approach was affordable for service providers, but it later became inconvenient when the cost of customizations and specializations increased. The SaaS approach introduces a more flexible way of delivering application services that are fully customizable by the user by integrating new services, injecting their own components, and designing the application and information workflows. Such a new approach has also been possible with the support of Web 2.0 technologies, which allowed turning the Web browser into a full-featured interface, able even to support application composition and development.

How is cloud computing related to SaaS? According to the classification of services shown in Figure 4.1, the SaaS approach lays on top of the cloud computing stack. It fits into the cloud computing vision expressed by the *XaaS* acronym, Everything-as-a-Service; and with SaaS, applications

are delivered as a service. Initially the SaaS model was of interest only for lead users and early adopters. The benefits delivered at that stage were the following:

- Software cost reduction and total cost of ownership (TCO) were paramount
- Service-level improvements
- Rapid implementation
- Standalone and configurable applications
- Rudimentary application and data integration
- Subscription and pay-as-you-go (PAYG) pricing

With the advent of cloud computing there has been an increasing acceptance of SaaS as a viable software delivery model. This led to transition into SaaS 2.0 [40], which does not introduce a new technology but transforms the way in which SaaS is used.

In particular, SaaS 2.0 is focused on providing a more robust infrastructure and application platforms driven by SLAs. Rather than being characterized as a more rapid implementation and deployment environment, SaaS 2.0 will focus on the rapid achievement of business objectives. This is why such evolution does not introduce any new technology: The existing technologies are composed together in order to achieve business goals efficiently. Fundamental to this perspective is the ability to leverage existing solutions and integrate value-added business services. The existing SaaS infrastructures not only allow the development and customization of applications, but they also facilitate the integration of services that are exposed by other parties. SaaS applications are then the result of the interconnection and the synergy of different applications and components that together provide customers with added value. This approach dramatically changes the software ecosystem of the SaaS market, which is no longer monopolized by a few vendors but is now a fully interconnected network of service providers, clustered around some "big hubs" that deliver the application to the customer. In this scenario, each single component integrated into the SaaS application becomes responsible to the user for ensuring the attached SLA and at the same time could be priced differently. Customers can then choose how to specialize their applications by deciding which components and services they want to integrate.

Software-as-a-Service applications can serve different needs. CRM, ERP, and social networking applications are definitely the most popular ones. SalesForce.com is probably the most successful and popular example of a CRM service. It provides a wide range of services for applications: customer relationship and human resource management, enterprise resource planning, and many other features. SalesForce.com builds on top of the Force.com platform, which provides a fully featured environment for building applications. It offers either a programming language or a visual environment to arrange components together for building applications. In addition to the basic features provided, the integration with third-party-made applications enriches SalesForce.com's value. In particular, through AppExchange customers can publish, search, and integrate new services and features into their existing applications. This makes SalesForce.com applications completely extensible and customizable. Similar solutions are offered by NetSuite and RightNow. NetSuite is an integrated software business suite featuring financials, CRM, inventory, and ecommerce functionalities integrated all together. RightNow is customer experience-centered SaaS application that integrates together different features, from chat to Web communities, to support the common activity of an enterprise.

Another important class of popular SaaS applications comprises social networking applications such as Facebook and professional networking sites such as LinkedIn. Other than providing the basic features of networking, they allow incorporating and extending their capabilities by integrating third-party applications. These can be developed as plug-ins for the hosting platform, as happens for Facebook, and made available to users, who can select which applications they want to add to their profile. As a result, the integrated applications get full access to the network of contacts and users' profile data. The nature of these applications can be of different types: office automation components, games, or integration with other existing services.

Office automation applications are also an important representative for SaaS applications: Google Documents and Zoho Office are examples of Web-based applications that aim to address all user needs for documents, spreadsheets, and presentation management. They offer a Web-based interface for creating, managing, and modifying documents that can be easily shared among users and made accessible from anywhere.

It is important to note the role of SaaS solution enablers, which provide an environment in which to integrate third-party services and share information with others. A quite successful example is Box.net, an SaaS application providing users with a Web space and profile that can be enriched and extended with third-party applications such as office automation, integration with CRM-based solutions, social Websites, and photo editing.

4.3 Types of clouds

Clouds constitute the primary outcome of cloud computing. They are a type of parallel and distributed system harnessing physical and virtual computers presented as a unified computing resource. Clouds build the infrastructure on top of which services are implemented and delivered to customers. Such infrastructures can be of different types and provide useful information about the nature and the services offered by the cloud. A more useful classification is given according to the administrative domain of a cloud: It identifies the boundaries within which cloud computing services are implemented, provides hints on the underlying infrastructure adopted to support such services, and qualifies them. It is then possible to differentiate four different types of cloud:

- *Public clouds*. The cloud is open to the wider public.
- *Private clouds*. The cloud is implemented within the private premises of an institution and generally made accessible to the members of the institution or a subset of them.
- *Hybrid or heterogeneous clouds*. The cloud is a combination of the two previous solutions and most likely identifies a private cloud that has been augmented with resources or services hosted in a public cloud.
- *Community clouds*. The cloud is characterized by a multi-administrative domain involving different deployment models (public, private, and hybrid), and it is specifically designed to address the needs of a specific industry.

Almost all the implementations of clouds can be classified in this categorization. In the following sections, we provide brief characterizations of these clouds.

4.3.1 Public clouds

Public clouds constitute the first expression of cloud computing. They are a realization of the canonical view of cloud computing in which the services offered are made available to anyone, from anywhere, and at any time through the Internet. From a structural point of view they are a distributed system, most likely composed of one or more datacenters connected together, on top of which the specific services offered by the cloud are implemented. Any customer can easily sign in with the cloud provider, enter her credential and billing details, and use the services offered.

Historically, public clouds were the first class of cloud that were implemented and offered. They offer solutions for minimizing IT infrastructure costs and serve as a viable option for handling peak loads on the local infrastructure. They have become an interesting option for small enterprises, which are able to start their businesses without large up-front investments by completely relying on public infrastructure for their IT needs. What made attractive public clouds compared to the reshaping of the private premises and the purchase of hardware and software was the ability to grow or shrink according to the needs of the related business. By renting the infrastructure or subscribing to application services, customers were able to dynamically upsize or downsize their IT according to the demands of their business. Currently, public clouds are used both to completely replace the IT infrastructure of enterprises and to extend it when it is required.

A fundamental characteristic of public clouds is multitenancy. A public cloud is meant to serve a multitude of users, not a single customer. Any customer requires a virtual computing environment that is separated, and most likely isolated, from other users. This is a fundamental requirement to provide effective monitoring of user activities and guarantee the desired performance and the other QoS attributes negotiated with users. QoS management is a very important aspect of public clouds. Hence, a significant portion of the software infrastructure is devoted to monitoring the cloud resources, to bill them according to the contract made with the user, and to keep a complete history of cloud usage for each customer. These features are fundamental to public clouds because they help providers offer services to users with full accountability.

A public cloud can offer any kind of service: infrastructure, platform, or applications. For example, Amazon EC2 is a public cloud that provides infrastructure as a service; Google AppEngine is a public cloud that provides an application development platform as a service; and SalesForce.com is a public cloud that provides software as a service. What makes public clouds peculiar is the way they are consumed: They are available to everyone and are generally architected to support a large quantity of users. What characterizes them is their natural ability to scale on demand and sustain peak loads.

From an architectural point of view there is no restriction concerning the type of distributed system implemented to support public clouds. Most likely, one or more datacenters constitute the physical infrastructure on top of which the services are implemented and delivered. Public clouds can be composed of geographically dispersed datacenters to share the load of users and better serve them according to their locations. For example, Amazon Web Services has datacenters installed in the United States, Europe, Singapore, and Australia; they allow their customers to choose between three different regions: *us-west-1*, *us-east-1*, or *eu-west-1*. Such regions are priced differently and are further divided into availability zones, which map to specific datacenters. According to the specific class of services delivered by the cloud, a different software stack is installed to manage the infrastructure: virtual machine managers, distributed middleware, or distributed applications.

4.3.2 Private clouds

Public clouds are appealing and provide a viable option to cut IT costs and reduce capital expenses, but they are not applicable in all scenarios. For example, a very common critique to the use of cloud computing in its canonical implementation is the *loss of control*. In the case of public clouds, the provider is in control of the infrastructure and, eventually, of the customers' core logic and sensitive data. Even though there could be regulatory procedure in place that guarantees fair management and respect of the customer's privacy, this condition can still be perceived as a threat or as an unacceptable risk that some organizations are not willing to take. In particular, institutions such as government and military agencies will not consider public clouds as an option for processing or storing their sensitive data. The risk of a breach in the security infrastructure of the provider could expose such information to others; this could simply be considered unacceptable.

In other cases, the loss of control of where your virtual IT infrastructure resides could open the way to other problematic situations. More precisely, the geographical location of a datacenter generally determines the regulations that are applied to management of digital information. As a result, according to the specific location of data, some sensitive information can be made accessible to government agencies or even considered outside the law if processed with specific cryptographic techniques. For example, the USA PATRIOT Act[5] provides its government and other agencies with virtually limitless powers to access information, including that belonging to any company that stores information in the U.S. territory. Finally, existing enterprises that have large computing infrastructures or large installed bases of software do not simply want to switch to public clouds, but they use the existing IT resources and optimize their revenue. All these aspects make the use of a public computing infrastructure not always possible. Yet the general idea supported by the cloud computing vision can still be attractive. More specifically, having an infrastructure able to deliver IT services on demand can still be a winning solution, even when implemented within the private premises of an institution. This idea led to the diffusion of private clouds, which are similar to public clouds, but their resource-provisioning model is limited within the boundaries of an organization.

Private clouds are virtual distributed systems that rely on a private infrastructure and provide internal users with dynamic provisioning of computing resources. Instead of a pay-as-you-go model as in public clouds, there could be other schemes in place, taking into account the usage of the cloud and proportionally billing the different departments or sections of an enterprise. Private clouds have the advantage of keeping the core business operations in-house by relying on the existing IT infrastructure and reducing the burden of maintaining it once the cloud has been set up. In this scenario, security concerns are less critical, since sensitive information does not flow out of the private infrastructure. Moreover, existing IT resources can be better utilized because the private cloud can provide services to a different range of users. Another interesting opportunity that comes with private clouds is the possibility of testing applications and systems at a comparatively lower

[5]The USA PATRIOT Act is a statute enacted by the U.S. government that increases the ability of law enforcement agencies to search telephone, email, medical, financial, and other records and eases restrictions on foreign intelligence gathering within the United States. The full text of the act is available at the Website of the Library of the Congress at the following address: http://thomas.loc.gov/cgi-bin/bdquery/z?d107:hr03162: (accessed April 20, 2010).

price rather than public clouds before deploying them on the public virtual infrastructure. A Forrester report [34] on the benefits of delivering in-house cloud computing solutions for enterprises highlighted some of the key advantages of using a private cloud computing infrastructure:

- *Customer information protection*. Despite assurances by the public cloud leaders about security, few provide satisfactory disclosure or have long enough histories with their cloud offerings to provide warranties about the specific level of security put in place on their systems. In-house security is easier to maintain and rely on.
- *Infrastructure ensuring SLAs*. Quality of service implies specific operations such as appropriate clustering and failover, data replication, system monitoring and maintenance, and disaster recovery, and other uptime services can be commensurate to the application needs. Although public cloud vendors provide some of these features, not all of them are available as needed.
- *Compliance with standard procedures and operations*. If organizations are subject to third-party compliance standards, specific procedures have to be put in place when deploying and executing applications. This could be not possible in the case of the virtual public infrastructure.

All these aspects make the use of cloud-based infrastructures in private premises an interesting option.

From an architectural point of view, private clouds can be implemented on more heterogeneous hardware: They generally rely on the existing IT infrastructure already deployed on the private premises. This could be a datacenter, a cluster, an enterprise desktop grid, or a combination of them. The physical layer is complemented with infrastructure management software (i.e., IaaS (M); see Section 4.2.2) or a PaaS solution, according to the service delivered to the users of the cloud.

Different options can be adopted to implement private clouds. Figure 4.4 provides a comprehensive view of the solutions together with some reference to the most popular software used to deploy private clouds. At the bottom layer of the software stack, virtual machine technologies such as Xen [35], KVM [36], and VMware serve as the foundations of the cloud. Virtual machine management technologies such as VMware vCloud, Eucalyptus [37], and OpenNebula [38] can be used to control the virtual infrastructure and provide an IaaS solution. VMware vCloud is a proprietary solution, but Eucalyptus provides full compatibility with Amazon Web Services interfaces and supports different virtual machine technologies such as Xen, KVM, and VMware. Like Eucalyptus, OpenNebula is an open-source solution for virtual infrastructure management that supports KVM, Xen, and VMware, which has been designed to easily integrate third-party IaaS providers. Its modular architecture allows extending the software with additional features such as the capability of reserving virtual machine instances by using Haizea [39] as scheduler.

Solutions that rely on the previous virtual machine managers and provide added value are OpenPEX [40] and InterGrid [41]. OpenPEX is Web-based system that allows the reservation of virtual machine instances and is designed to support different back ends (at the moment only the support for Xen is implemented). InterGrid provides added value on top of OpenNebula and Amazon EC2 by allowing the reservation of virtual machine instances and managing multi-administrative domain clouds. PaaS solutions can provide an additional layer and deliver a high-level service for private clouds. Among the options available for private deployment of clouds we can consider DataSynapse, Zimory Pools, Elastra, and Aneka. DataSynapse is a global provider of application virtualization software. By relying on the VMware virtualization technology,

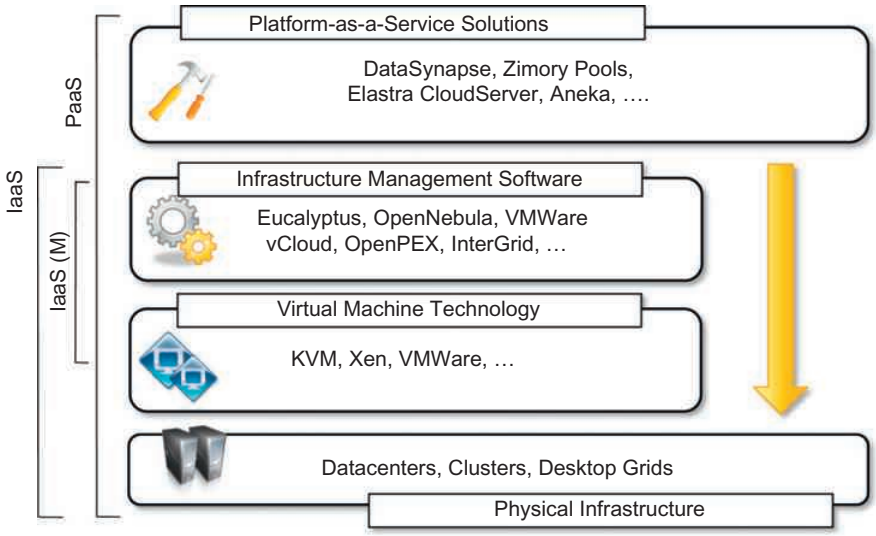


FIGURE 4.4

Private clouds hardware and software stack.

DataSynapse provides a flexible environment for building private clouds on top of datacenters. Elastra Cloud Server is a platform for easily configuring and deploying distributed application infrastructures on clouds. Zimory provides a software infrastructure layer that automates the use of resource pools based on Xen, KVM, and VMware virtualization technologies. It allows creating an internal cloud composed of sparse private and public resources and provides facilities for migrating applications within the existing infrastructure. Aneka is a software development platform that can be used to deploy a cloud infrastructure on top of heterogeneous hardware: datacenters, clusters, and desktop grids. It provides a pluggable service-oriented architecture that's mainly devoted to supporting the execution of distributed applications with different programming models: bag of tasks, MapReduce, and others.

Private clouds can provide in-house solutions for cloud computing, but if compared to public clouds they exhibit more limited capability to scale elastically on demand.

4.3.3 Hybrid clouds

Public clouds are large software and hardware infrastructures that have a capability that is huge enough to serve the needs of multiple users, but they suffer from security threats and administrative pitfalls. Although the option of completely relying on a public virtual infrastructure is appealing for companies that did not incur IT capital costs and have just started considering their IT needs (i.e., start-ups), in most cases the private cloud option prevails because of the existing IT infrastructure.

Private clouds are the perfect solution when it is necessary to keep the processing of information within an enterprise's premises or it is necessary to use the existing hardware and software infrastructure. One of the major drawbacks of private deployments is the inability to scale on demand and to efficiently address peak loads. In this case, it is important to leverage capabilities of public clouds as needed. Hence, a hybrid solution could be an interesting opportunity for taking advantage of the best of the private and public worlds. This led to the development and diffusion of hybrid clouds.

Hybrid clouds allow enterprises to exploit existing IT infrastructures, maintain sensitive information within the premises, and naturally grow and shrink by provisioning external resources and releasing them when they're no longer needed. Security concerns are then only limited to the public portion of the cloud that can be used to perform operations with less stringent constraints but that are still part of the system workload. Figure 4.5 provides a general overview of a hybrid cloud: It is a heterogeneous distributed system resulting from a private cloud that integrates additional services or resources from one or more public clouds. For this reason they are also called *heterogeneous clouds*. As depicted in the diagram, dynamic provisioning is a fundamental component in this scenario. Hybrid clouds address scalability issues by leveraging external resources for exceeding

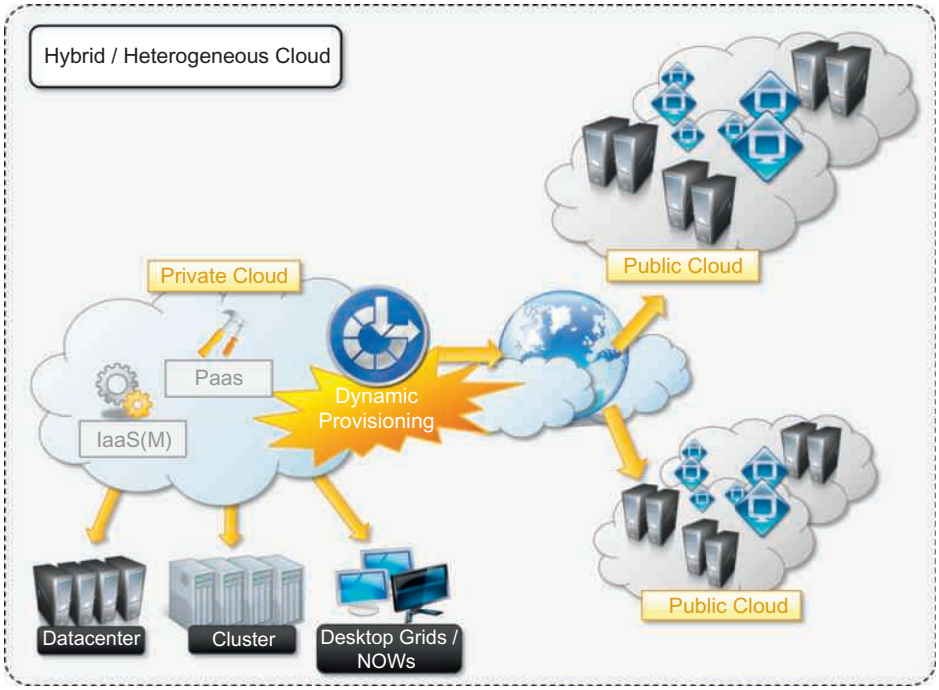


FIGURE 4.5

Hybrid/heterogeneous cloud overview.

capacity demand. These resources or services are temporarily leased for the time required and then released. This practice is also known as *cloudbursting*.[6]

Whereas the concept of hybrid cloud is general, it mostly applies to IT infrastructure rather than software services. Service-oriented computing already introduces the concept of integration of paid software services with existing application deployed in the private premises. In an IaaS scenario, *dynamic provisioning* refers to the ability to acquire on demand virtual machines in order to increase the capability of the resulting distributed system and then release them. Infrastructure management software and PaaS solutions are the building blocks for deploying and managing hybrid clouds. In particular, with respect to private clouds, dynamic provisioning introduces a more complex scheduling algorithm and policies, the goal of which is also to optimize the budget spent to rent public resources.

Infrastructure management software such as OpenNebula already exposes the capability of integrating resources from public clouds such as Amazon EC2. In this case the virtual machine obtained from the public infrastructure is managed as all the other virtual machine instances maintained locally. What is missing is then an advanced scheduling engine that's able to differentiate these resources and provide smart allocations by taking into account the budget available to extend the existing infrastructure. In the case of OpenNebula, advanced schedulers such as Haizea can be integrated to provide cost-based scheduling. A different approach is taken by InterGrid. This is essentially a distributed scheduling engine that manages the allocation of virtual machines in a collection of peer networks. Such networks can be represented by a local cluster, a gateway to a public cloud, or a combination of the two. Once a request is submitted to one of the InterGrid gateways, it is served by possibly allocating virtual instances in all the peered networks, and the allocation of requests is performed by taking into account the user budget and the peering arrangements between networks.

Dynamic provisioning is most commonly implemented in PaaS solutions that support hybrid clouds. As previously discussed, one of the fundamental components of PaaS middleware is the mapping of distributed applications onto the cloud infrastructure. In this scenario, the role of dynamic provisioning becomes fundamental to ensuring the execution of applications under the QoS agreed on with the user. For example, Aneka provides a provisioning service that leverages different IaaS providers for scaling the existing cloud infrastructure [42]. The provisioning service cooperates with the scheduler, which is in charge of guaranteeing a specific QoS for applications. In particular, each user application has a budget attached, and the scheduler uses that budget to optimize the execution of the application by renting virtual nodes if needed. Other PaaS implementations support the deployment of hybrid clouds and provide dynamic provisioning capabilities. Among those discussed for the implementation and management of private clouds we can cite Elastra CloudServer and Zimory Pools.

[6]According to the Cloud Computing Wiki, the term *cloudburst* has a double meaning; it also refers to the "failure of a cloud computing environment due to the inability to handle a spike in demand" (http://sites.google.com/site/Cloudcomputingwiki/Home/Cloud-computing-vocabulary). In this book, we always refer to the dynamic provisioning of resources from public clouds when mentioning this term.

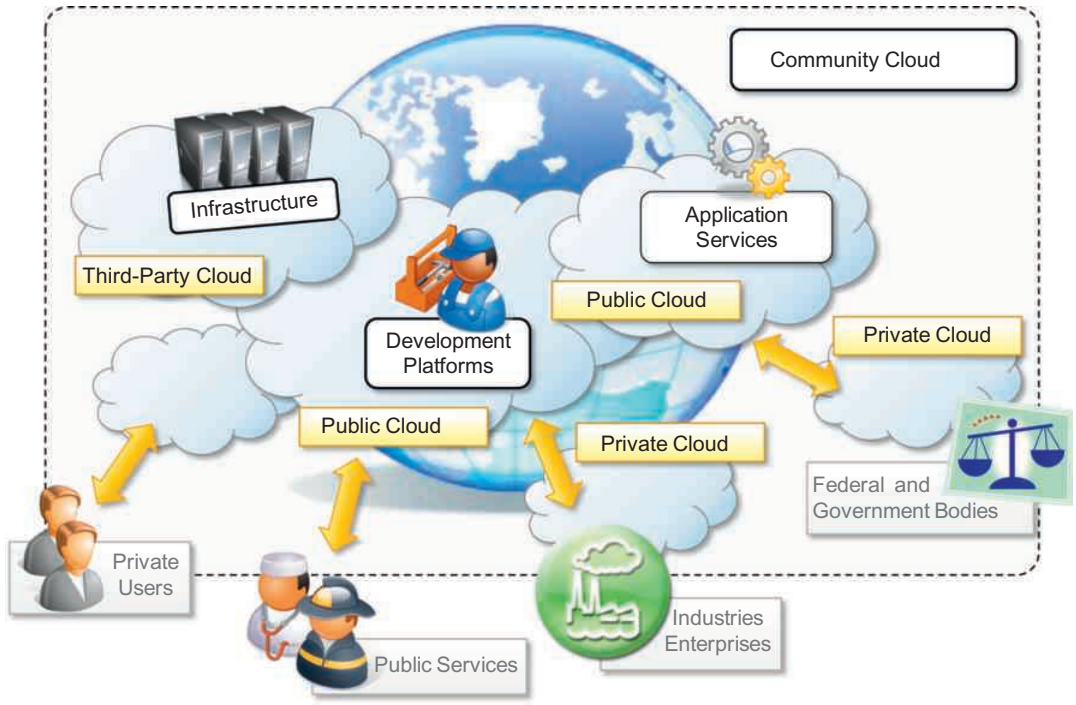


FIGURE 4.6

A community cloud.

4.3.4 Community clouds

Community clouds are distributed systems created by integrating the services of different clouds to address the specific needs of an industry, a community, or a business sector. The National Institute of Standards and Technologies (NIST) [43] characterizes community clouds as follows:

> *The infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise.*

Figure 4.6 provides a general view of the usage scenario of community clouds, together with reference architecture. The users of a specific community cloud fall into a well-identified community, sharing the same concerns or needs; they can be government bodies, industries, or even simple users, but all of them focus on the same issues for their interaction with the cloud. This is a different scenario than public clouds, which serve a multitude of users with different needs. Community clouds are also different from private clouds, where the services are generally delivered within the institution that owns the cloud.

From an architectural point of view, a community cloud is most likely implemented over multiple administrative domains. This means that different organizations such as government bodies,

private enterprises, research organizations, and even public virtual infrastructure providers contribute with their resources to build the cloud infrastructure.

Candidate sectors for community clouds are as follows:

- *Media industry*. In the media industry, companies are looking for low-cost, agile, and simple solutions to improve the efficiency of content production. Most media productions involve an extended ecosystem of partners. In particular, the creation of digital content is the outcome of a collaborative process that includes movement of large data, massive compute-intensive rendering tasks, and complex workflow executions. Community clouds can provide a shared environment where services can facilitate business-to-business collaboration and offer the horsepower in terms of aggregate bandwidth, CPU, and storage required to efficiently support media production.
- *Healthcare industry*. In the healthcare industry, there are different scenarios in which community clouds could be of use. In particular, community clouds can provide a global platform on which to share information and knowledge without revealing sensitive data maintained within the private infrastructure. The naturally hybrid deployment model of community clouds can easily support the storing of patient-related data in a private cloud while using the shared infrastructure for noncritical services and automating processes within hospitals.
- *Energy and other core industries*. In these sectors, community clouds can bundle the comprehensive set of solutions that together vertically address management, deployment, and orchestration of services and operations. Since these industries involve different providers, vendors, and organizations, a community cloud can provide the right type of infrastructure to create an open and fair market.
- *Public sector*. Legal and political restrictions in the public sector can limit the adoption of public cloud offerings. Moreover, governmental processes involve several institutions and agencies and are aimed at providing strategic solutions at local, national, and international administrative levels. They involve business-to-administration, citizen-to-administration, and possibly business-to-business processes. Some examples include invoice approval, infrastructure planning, and public hearings. A community cloud can constitute the optimal venue to provide a distributed environment in which to create a communication platform for performing such operations.
- *Scientific research*. Science clouds are an interesting example of community clouds. In this case, the common interest driving different organizations sharing a large distributed infrastructure is scientific computing.

The term *community cloud* can also identify a more specific type of cloud that arises from concern over the controls of vendors in cloud computing and that aspire to combine the principles of *digital ecosystems*[7] [44] with the case study of cloud computing. A community cloud is formed by harnessing the underutilized resources of user machines [45] and providing an infrastructure in

[7]*Digital ecosystems* are distributed, adaptive, and open sociotechnical systems with properties of self-organization, scalability, and sustainability inspired by natural ecosystems. The primary aim of digital ecosystems is to sustain the regional development of small and medium-sized enterprises (SMEs).

which each can be at the same time a consumer, a producer, or a coordinator of the services offered by the cloud. The benefits of these community clouds are the following:

- *Openness*. By removing the dependency on cloud vendors, community clouds are open systems in which fair competition between different solutions can happen.
- *Community*. Being based on a collective that provides resources and services, the infrastructure turns out to be more scalable because the system can grow simply by expanding its user base.
- *Graceful failures*. Since there is no single provider or vendor in control of the infrastructure, there is no single point of failure.
- *Convenience and control*. Within a community cloud there is no conflict between convenience and control because the cloud is shared and owned by the community, which makes all the decisions through a collective democratic process.
- *Environmental sustainability*. The community cloud is supposed to have a smaller carbon footprint because it harnesses underutilized resources. Moreover, these clouds tend to be more organic by growing and shrinking in a symbiotic relationship to support the demand of the community, which in turn sustains it.

This is an alternative vision of a community cloud, focusing more on the social aspect of the clouds that are formed as an aggregation of resources of community members. The idea of a heterogeneous infrastructure built to serve the needs of a community of people is also reflected in the previous definition, but in that case the attention is focused on the commonality of interests that aggregates the users of the cloud into a community. In both cases, the concept of community is fundamental.

4.4 Economics of the cloud

The main drivers of cloud computing are economy of scale and simplicity of software delivery and its operation. In fact, the biggest benefit of this phenomenon is financial: the *pay-as-you-go* model offered by cloud providers. In particular, cloud computing allows:

- Reducing the capital costs associated to the IT infrastructure
- Eliminating the depreciation or lifetime costs associated with IT capital assets
- Replacing software licensing with subscriptions
- Cutting the maintenance and administrative costs of IT resources

A *capital cost* is the cost occurred in purchasing an asset that is useful in the production of goods or the rendering of services. Capital costs are one-time expenses that are generally paid up front and that will contribute over the long term to generate profit. The IT infrastructure and the software are capital assets because enterprises require them to conduct their business. At present it does not matter whether the principal business of an enterprise is related to IT, because the business will definitely have an IT department that is used to automate many of the activities that are performed within the enterprise: payroll, customer relationship management, enterprise resource planning, tracking and inventory of products, and others. Hence, IT resources constitute a capital cost for any kind of enterprise. It is good practice to try to keep capital costs low because they introduce

expenses that will generate profit over time; more than that, since they are associated with material things they are subject to *depreciation* over time, which in the end reduces the profit of the enterprise because such costs are directly subtracted from the enterprise revenues. In the case of IT capital costs, the depreciation costs are represented by the loss of value of the hardware over time and the aging of software products that need to be replaced because new features are required.

Before cloud computing diffused within the enterprise, the budget spent on IT infrastructure and software constituted a significant expense for medium-sized and large enterprises. Many enterprises own a small or medium-sized datacenter that introduces several operational costs in terms of maintenance, electricity, and cooling. Additional operational costs are occurred in maintaining an IT department and an IT support center. Moreover, other costs are triggered by the purchase of potentially expensive software. With cloud computing these costs are significantly reduced or simply disappear according to its penetration. One of the advantages introduced by the cloud computing model is that it shifts the capital costs previously allocated to the purchase of hardware and software into operational costs inducted by renting the infrastructure and paying subscriptions for the use of software. These costs can be better controlled according to the business needs and prosperity of the enterprise. Cloud computing also introduces reductions in administrative and maintenance costs. That is, there is no or limited need for having administrative staff take care of the management of the cloud infrastructure. At the same time, the cost of IT support staff is also reduced. When it comes to depreciation costs, they simply disappear for the enterprise, since in a scenario where all the IT needs are served by the cloud there are no IT capital assets that depreciate over time.

The amount of cost savings that cloud computing can introduce within an enterprise is related to the specific scenario in which cloud services are used and how they contribute to generate a profit for the enterprise. In the case of a small startup, it is possible to completely leverage the cloud for many aspects, such as:

- IT infrastructure
- Software development
- CRM and ERP

In this case it is possible to completely eliminate capital costs because there are no initial IT assets. The situation is completely different in the case of enterprises that already have a considerable amount of IT assets. In this case, cloud computing, especially IaaS-based solutions, can help manage unplanned capital costs that are generated by the needs of the enterprise in the short term. In this case, by leveraging cloud computing, these costs can be turned into operational costs that last as long as there is a need for them. For example, IT infrastructure leasing helps more efficiently manage peak loads without inducing capital expenses. As soon as the increased load does not justify the use of additional resources, these can be released and the costs associated with them disappear. This is the most adopted model of cloud computing because many enterprises already have IT facilities. Another option is to make a slow transition toward cloud-based solutions while the capital IT assets get depreciated and need to be replaced. Between these two cases there is a wide variety of scenarios in which cloud computing could be of help in generating profits for enterprises.

Another important aspect is the elimination of some indirect costs that are generated by IT assets, such as software licensing and support and carbon footprint emissions. With cloud computing, an enterprise uses software applications on a subscription basis, and there is no need for any licensing fee because the software providing the service remains the property of the provider. Leveraging IaaS solutions allows room for datacenter consolidation that in the end could result in a smaller carbon footprint. In some countries such as Australia, the carbon footprint emissions are taxable, so by reducing or completely eliminating such emissions, enterprises can pay less tax.

In terms of the pricing models introduced by cloud computing, we can distinguish three different strategies that are adopted by the providers:

- *Tiered pricing*. In this model, cloud services are offered in several tiers, each of which offers a fixed computing specification and SLA at a specific price per unit of time. This model is used by Amazon for pricing the EC2 service, which makes available different server configurations in terms of computing capacity (CPU type and speed, memory) that have different costs per hour.
- *Per-unit pricing*. This model is more suitable to cases where the principal source of revenue for the cloud provider is determined in terms of units of specific services, such as data transfer and memory allocation. In this scenario customers can configure their systems more efficiently according to the application needs. This model is used, for example, by GoGrid, which makes customers pay according to RAM/hour units for the servers deployed in the GoGrid cloud.
- *Subscription-based pricing*. This is the model used mostly by SaaS providers in which users pay a periodic subscription fee for use of the software or the specific component services that are integrated in their applications.

All of these costs are based on a pay-as-you-go model, which constitutes a more flexible solution for supporting the delivery on demand of IT services. This is what actually makes possible the conversion of IT capital costs into operational costs, since the cost of buying hardware turns into a cost for leasing it and the cost generated by the purchase of software turns into a subscription fee paid for using it.

4.5 Open challenges

Still in its infancy, cloud computing presents many challenges for industry and academia. There is a significant amount of work in academia focused on defining the challenges brought by this phenomenon [46–49]. In this section, we highlight the most important ones: the definition and the formalization of cloud computing, the interoperation between different clouds, the creation of standards, security, scalability, fault tolerance, and organizational aspects.

4.5.1 Cloud definition

As discussed earlier, there have been several attempts made to define cloud computing and to provide a classification of all the services and technologies identified as such. One of the most comprehensive formalizations is noted in the NIST working definition of cloud computing [43]. It **characterizes** cloud computing as on-demand self-service, broad network access, resource-pooling,

rapid elasticity, and measured service; **classifies** services as SaaS, PaaS, and IaaS; and **categorizes** deployment models as public, private, community, and hybrid clouds. The view is in line with our discussion and shared by many IT practitioners and academics.

Despite the general agreement on the NIST definition, there are alternative taxonomies for cloud services. David Linthicum, founder of BlueMountains Labs, provides a more detailed classification,[8] which comprehends 10 different classes and better suits the vision of cloud computing within the enterprise. A different approach has been taken at the University of California, Santa Barbara (UCSB) [50], which departs from the XaaS concept and tries to define an ontology for cloud computing. In their work the concept of a cloud is dissected into five main layers: applications, software environments, software infrastructure, software kernel, and hardware. Each layer addresses the needs of a different class of users within the cloud computing community and most likely builds on the underlying layers. According to the authors, this work constitutes the first effort to provide a more robust interaction model between the different cloud entities on both the functional level and the semantic level.

These characterizations and taxonomies reflect what is meant by cloud computing at the present time, but being in its infancy the phenomenon is constantly evolving, and the same will happen to the attempts to capture the real nature of cloud computing. It is interesting to note that the principal characterization used in this book as a reference for introducing and explaining cloud computing is considered a working definition, which by nature identifies something that continuously changes over time by becoming refined.

4.5.2 Cloud interoperability and standards

Cloud computing is a service-based model for delivering IT infrastructure and applications like utilities such as power, water, and electricity. To fully realize this goal, introducing standards and allowing interoperability between solutions offered by different vendors are objectives of fundamental importance. Vendor lock-in constitutes one of the major strategic barriers against the seamless adoption of cloud computing at all stages. In particular there is major fear on the part of enterprises in which IT constitutes the significant part of their revenues. Vendor lock-in can prevent a customer from switching to another competitor's solution, or when this is possible, it happens at considerable conversion cost and requires significant amounts of time. This can occur either because the customer wants to find a more suitable solution for customer needs or because the vendor is no longer able to provide the required service. The presence of standards that are actually implemented and adopted in the cloud computing community could give room for interoperability and then lessen the risks resulting from vendor lock-in.

The current state of standards and interoperability in cloud computing resembles the early Internet era, when there was no common agreement on the protocols and technologies used and each organization had its own network. Yet the first steps toward a standardization process have been made, and a few organizations, such as the Cloud Computing Interoperability Forum (CCIF),[9]

[8]David Linthicum, Cloud Computing Ontology Framework; http://Cloudcomputing.sys-con.com/node/811519.
[9]www.Cloudforum.org.

the Open Cloud Consortium,[10] and the DMTF Cloud Standards Incubator,[11] are leading the path. Another interesting initiative is the Open Cloud Manifesto,[12] which embodies the point of view of various stakeholders on the benefits of open standards in the field.

The standardization efforts are mostly concerned with the lower level of the cloud computing architecture, which is the most popular and developed. In particular, in the IaaS market, the use of a proprietary virtual machine format constitutes the major reasons for the vendor lock-in, and efforts to provide virtual machine image compatibility between IaaS vendors can possibly improve the level of interoperability among them. The Open Virtualization Format (OVF) [51] is an attempt to provide a common format for storing the information and metadata describing a virtual machine image. Even though the OVF provides a full specification for packaging and distributing virtual machine images in completely platform-independent fashion, it is supported by few vendors that use it to import static virtual machine images. The challenge is providing standards for supporting the migration of running instances, thus allowing the real ability of switching from one infrastructure vendor to another in a completely transparent manner.

Another direction in which standards try to move is devising a general reference architecture for cloud computing systems and providing a standard interface through which one can interact with them. At the moment the compatibility between different solutions is quite restricted, and the lack of a common set of APIs make the interaction with cloud-based solutions vendor specific. In the IaaS market, Amazon Web Services plays a leading role, and other IaaS solutions, mostly open source, provide AWS-compatible APIs, thus constituting themselves as valid alternatives. Even in this case, there is no consistent trend in devising some common APIs for interfacing with IaaS (and, in general, XaaS), and this constitutes one of the areas in which a considerable improvement can be made in the future.

4.5.3 Scalability and fault tolerance

The ability to scale on demand constitutes one of the most attractive features of cloud computing. Clouds allow scaling beyond the limits of the existing in-house IT resources, whether they are infrastructure (compute and storage) or applications services. To implement such a capability, the cloud middleware has to be designed with the principle of scalability along different dimensions in mind—for example, performance, size, and load. The cloud middleware manages a huge number of resource and users, which rely on the cloud to obtain the horsepower that they cannot obtain within the premises without bearing considerable administrative and maintenance costs. These costs are a reality for whomever develops, manages, and maintains the cloud middleware and offers the service to customers. In this scenario, the ability to tolerate failure becomes fundamental, sometimes even more important than providing an extremely efficient and optimized system. Hence, the challenge in this case is designing highly scalable and fault-tolerant systems that are easy to manage and at the same time provide competitive performance.

[10] www.opencloudconsortium.org.
[11] www.dmtf.org/about/cloud-incubator.
[12] www.opencloudmanifesto.org.

4.5.4 Security, trust, and privacy

Security, trust, and privacy issues are major obstacles for massive adoption of cloud computing. The traditional cryptographic technologies are used to prevent data tampering and access to sensitive information. The massive use of virtualization technologies exposes the existing system to new threats, which previously were not considered applicable. For example, it might be possible that applications hosted in the cloud can process sensitive information; such information can be stored within a cloud storage facility using the most advanced technology in cryptography to protect data and then be considered safe from any attempt to access it without the required permissions. Although these data are processed in memory, they must necessarily be decrypted by the legitimate application, but since the application is hosted in a managed virtual environment it becomes accessible to the virtual machine manager that by program is designed to access the memory pages of such an application. In this case, what is experienced is a lack of control over the environment in which the application is executed, which is made possible by leveraging the cloud. It then happens that a new way of using existing technologies creates new opportunities for additional threats to the security of applications. The lack of control over their own data and processes also poses severe problems for the trust we give to the cloud service provider and the level of privacy we want to have for our data.

On one side we need to decide whether to trust the provider itself; on the other side, specific regulations can simply prevail over the agreement the provider is willing to establish with us concerning the privacy of the information managed on our behalf. Moreover, cloud services delivered to the end user can be the result of a complex stack of services that are obtained by third parties via the primary cloud service provider. In this case there is a chain of responsibilities in terms of service delivery that can introduce more vulnerability for the secure management of data, the enforcement of privacy rules, and the trust given to the service provider. In particular, when a violation of privacy or illegal access to sensitive information is detected, it could become difficult to identify who is liable for such violations. The challenges in this area are, then, mostly concerned with devising secure and trustable systems from different perspectives: technical, social, and legal.

4.5.5 Organizational aspects

Cloud computing introduces a significant change in the way IT services are consumed and managed. More precisely, storage, compute power, network infrastructure, and applications are delivered as metered services over the Internet. This introduces a billing model that is new within typical enterprise IT departments, which requires a certain level of cultural and organizational process maturity. In particular, a wide acceptance of cloud computing will require a significant change to business processes and organizational boundaries. Some interesting questions arise in considering the role of the IT department in this new scenario. In particular, the following questions have to be considered:

- What is the new role of the IT department in an enterprise that completely or significantly relies on the cloud?
- How will the compliance department perform its activity when there is a considerable lack of control over application workflows?

- What are the implications (political, legal, etc.) for organizations that lose control over some aspects of their services?
- What will be the perception of the end users of such services?

From an organizational point of view, the lack of control over the management of data and processes poses not only security threats but also new problems that previously did not exist. Traditionally, when there was a problem with computer systems, organizations developed strategies and solutions to cope with them, often by relying on local expertise and knowledge. One of the major advantages of moving IT infrastructure and services to the cloud is to reduce or completely remove the costs related to maintenance and support. As a result, users of such infrastructure and services lose a reference to deal with for IT troubleshooting. At the same time, the existing IT staff is required to have a different kind of competency and, in general, fewer skills, thus reducing their value. These are the challenges from an organizational point of view that must be faced and that will significantly change the relationships within the enterprise itself among the various groups of people working together.

SUMMARY

In this chapter we discussed the fundamental characteristics of cloud computing and introduced reference architecture for classifying and organizing cloud services. To best sum up the content of this chapter, we can recall the NIST working definition of cloud computing, which outlines the fundamental aspects of this phenomenon as follows:

- *Five essential characteristics*. In-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service.
- *Three service models*. Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS).
- *Four deployment models*. Public clouds, private clouds, community clouds, and hybrid clouds.

The major driving force for rapid adoption of cloud computing are the economics and the simplicity of software delivery and operation. Cloud computing presents considerable opportunity to increase the profits of enterprises by reducing capital costs of IT assets and transforming them into operational costs. For these reasons we have also discussed the economic and cost models introduced with cloud computing.

Although cloud computing has been rapidly adopted in industry, there are several open research challenges in areas such as management of cloud computing systems, their security, and social and organizational issues. There is significant room for advancement in software infrastructure and models supporting cloud computing.

Review questions

1. What does the acronym *XaaS* stand for?
2. What are the fundamental components introduced in the cloud reference model?

3. What does Infrastructure-as-a-Service refer to?
4. Which are the basic components of an IaaS-based solution for cloud computing?
5. Provide some examples of IaaS implementations.
6. What are the main characteristics of a Platform-as-a-Service solution?
7. Describe the different categories of options available in a PaaS market.
8. What does the acronym SaaS mean? How does it relate to cloud computing?
9. Give the name of some popular Software-as-a-Service solutions.
10. Classify the various types of clouds.
11. Give an example of the public cloud.
12. Which is the most common scenario for a private cloud?
13. What kinds of needs are addressed by heterogeneous clouds?
14. Describe the fundamental features of the economic and business model behind cloud computing.
15. How does cloud computing help to reduce the time to market for applications and to cut down capital expenses?
16. List some of the challenges in cloud computing.

PART

Cloud Application Programming and the Aneka Platform

2

This page intentionally left blank

CHAPTER

Aneka
Cloud Application Platform

5

Aneka is Manjrasoft Pty. Ltd.'s solution for developing, deploying, and managing cloud applications. Aneka consists of a scalable cloud middleware that can be deployed on top of heterogeneous computing resources. It offers an extensible collection of services coordinating the execution of applications, helping administrators monitor the status of the cloud, and providing integration with existing cloud technologies. One of Aneka's key advantages is its extensible set of application programming interfaces (APIs) associated with different types of programming models—such as Task, Thread, and MapReduce—used for developing distributed applications, integrating new capabilities into the cloud, and supporting different types of cloud deployment models: public, private, and hybrid (see Figure 5.1). These features differentiate Aneka from infrastructure management software and characterize it as a platform for developing, deploying, and managing execution of applications on various types of clouds.

This chapter provides a complete overview of the framework by first describing the architecture of the system. It introduces Aneka's components and the fundamental services that make up the Aneka Cloud and discusses some common deployment scenarios.

5.1 Framework overview

Aneka is a software platform for developing cloud computing applications. It allows harnessing of disparate computing resources and managing them into a unique virtual domain—the Aneka Cloud—in which applications are executed. According to the Cloud Computing Reference Model presented in Chapter 1, Aneka is a *pure PaaS* solution for cloud computing. Aneka is a cloud middleware product that can be deployed on a heterogeneous set of resources: a network of computers, a multicore server, datacenters, virtual cloud infrastructures, or a mixture of these. The framework provides both middleware for managing and scaling distributed applications and an extensible set of APIs for developing them.

Figure 5.2 provides a complete overview of the components of the Aneka framework. The core infrastructure of the system provides a uniform layer that allows the framework to be deployed over different platforms and operating systems. The physical and virtual resources representing the bare metal of the cloud are managed by the Aneka container, which is installed on each node and constitutes the basic building block of the middleware. A collection of interconnected containers constitute the Aneka Cloud: a single domain in which services are made available to users and

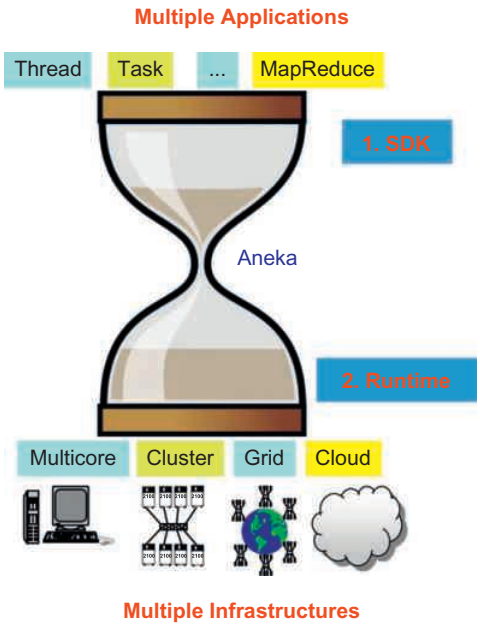


FIGURE 5.1

Aneka's capabilities at a glance.

developers. The container features three different classes of services: *Fabric Services*, *Foundation Services*, and *Execution Services*. These take care of infrastructure management, supporting services for the Aneka Cloud, and application management and execution, respectively. These services are made available to developers and administrators by means of the application management and development layer, which includes interfaces and APIs for developing cloud applications and the management tools and interfaces for controlling Aneka Clouds.

Aneka implements a service-oriented architecture (SOA), and services are the fundamental components of an Aneka Cloud. Services operate at container level and, except for the platform abstraction layer, they provide developers, users, and administrators with all features offered by the framework. Services also constitute the extension and customization point of Aneka Clouds: The infrastructure allows for the integration of new services or replacement of the existing ones with a different implementation. The framework includes the basic services for infrastructure and node management, application execution, accounting, and system monitoring; existing services can be extended and new features can be added to the cloud by dynamically plugging new ones into the container. Such extensible and flexible infrastructure enables Aneka Clouds to support different programming and execution models for applications. A programming model represents a collection of abstractions that developers can use to express distributed applications; the runtime support for a programming model is constituted by a collection of execution and foundation services interacting together to carry out application execution. Thus, the implementation of a new model requires the development of the specific programming abstractions used by application developers and the

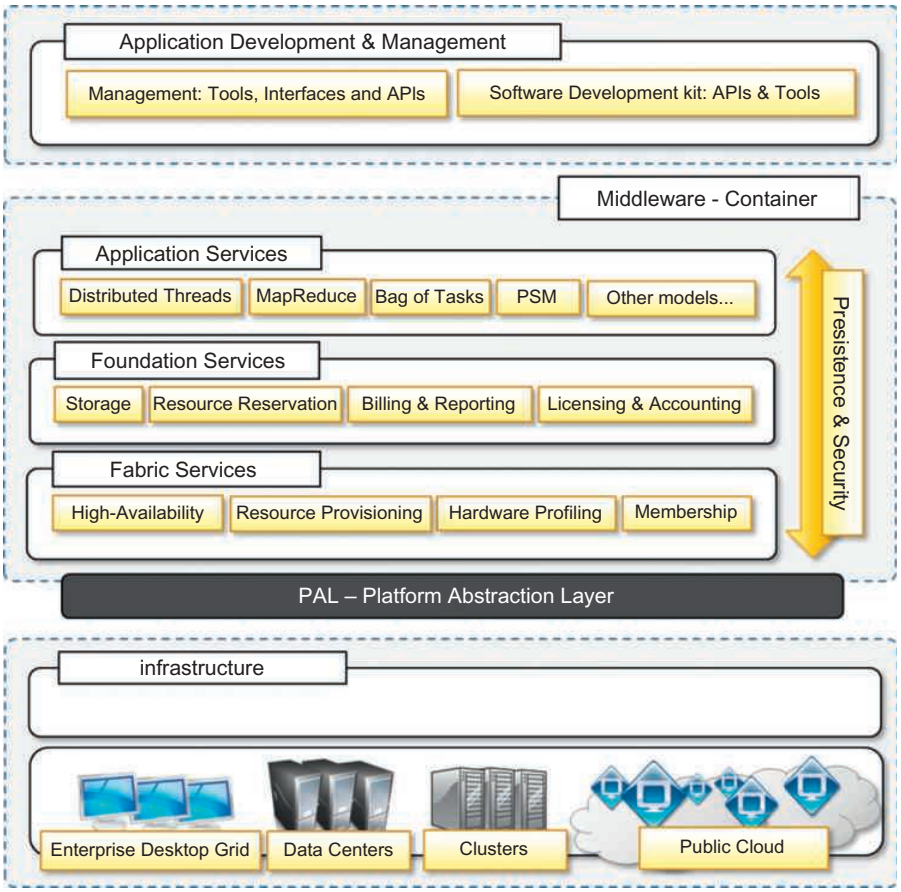


FIGURE 5.2

Aneka framework overview.

services, providing runtime support for them. Programming models are just one aspect of application management and execution. Within an Aneka Cloud environment, there are different aspects involved in providing a scalable and elastic infrastructure and distributed runtime for applications. These services involve:

- *Elasticity and scaling*. By means of the dynamic provisioning service, Aneka supports dynamically upsizing and downsizing of the infrastructure available for applications.
- *Runtime management*. The runtime machinery is responsible for keeping the infrastructure up and running and serves as a hosting environment for services. It is primarily represented by the container and a collection of services that manage service membership and lookup, infrastructure maintenance, and profiling.
- *Resource management*. Aneka is an elastic infrastructure in which resources are added and removed dynamically according to application needs and user requirements. To provide

QoS-based execution, the system not only allows dynamic provisioning but also provides capabilities for reserving nodes for exclusive use by specific applications.

- *Application management.* A specific subset of services is devoted to managing applications. These services include scheduling, execution, monitoring, and storage management.
- *User management.* Aneka is a multitenant distributed environment in which multiple applications, potentially belonging to different users, are executed. The framework provides an extensible user system via which it is possible to define users, groups, and permissions. The services devoted to user management build up the security infrastructure of the system and constitute a fundamental element for accounting management.
- *QoS/SLA management and billing.* Within a cloud environment, application execution is metered and billed. Aneka provides a collection of services that coordinate together to take into account the usage of resources by each application and to bill the owning user accordingly.

All these services are available to specific interfaces and APIs on top of which the software development kit (SDK) and management kit are built. The SDK mainly relates to application development and modeling; it provides developers with APIs to develop applications with the existing programming models and an object model for creating new models. The management kit is mostly focused on interacting with the runtime services for managing the infrastructure, users, and applications. The management kit gives a complete view of Aneka Clouds and allows monitoring Aneka's status, whereas the SDK is more focused on the single application and provides means to control its execution from a single user. Both components are meant to provide an easy-to-use interface via which to interact and manage containers that are the core component of the Aneka framework.

5.2 Anatomy of the Aneka container

The Aneka container constitutes the building blocks of Aneka Clouds and represents the runtime machinery available to services and applications. The container, the unit of deployment in Aneka Clouds, is a lightweight software layer designed to host services and interact with the underlying operating system and hardware. The main role of the container is to provide a lightweight environment in which to deploy services and some basic capabilities such as communication channels through which it interacts with other nodes in the Aneka Cloud. Almost all operations performed within Aneka are carried out by the services managed by the container. The services installed in the Aneka container can be classified into three major categories:

- Fabric Services
- Foundation Services
- Application Services

The services stack resides on top of the *Platform Abstraction Layer (PAL)*, representing the interface to the underlying operating system and hardware. It provides a uniform view of the software and hardware environment in which the container is running. Persistence and security traverse all the services stack to provide a secure and reliable infrastructure. In the following sections we discuss the components of these layers in more detail.

5.2.1 From the ground up: the platform abstraction layer

The core infrastructure of the system is based on the .NET technology and allows the Aneka container to be portable over different platforms and operating systems. Any platform featuring an ECMA-334 [52] and ECMA-335 [53] compatible environment can host and run an instance of the Aneka container.

The *Common Language Infrastructure (CLI)*, which is the specification introduced in the ECMA-334 standard, defines a common runtime environment and application model for executing programs but does not provide any interface to access the hardware or to collect performance data from the hosting operating system. Moreover, each operating system has a different file system organization and stores that information differently. The *Platform Abstraction Layer (PAL)* addresses this heterogeneity and provides the container with a uniform interface for accessing the relevant hardware and operating system information, thus allowing the rest of the container to run unmodified on any supported platform.

The PAL is responsible for detecting the supported hosting environment and providing the corresponding implementation to interact with it to support the activity of the container. The PAL provides the following features:

- Uniform and platform-independent implementation interface for accessing the hosting platform
- Uniform access to extended and additional properties of the hosting platform
- Uniform and platform-independent access to remote nodes
- Uniform and platform-independent management interfaces

The PAL is a small layer of software that comprises a detection engine, which automatically configures the container at boot time, with the platform-specific component to access the above information and an implementation of the abstraction layer for the Windows, Linux, and Mac OS X operating systems.

The collectible data that are exposed by the PAL are the following:

- Number of cores, frequency, and CPU usage
- Memory size and usage
- Aggregate available disk space
- Network addresses and devices attached to the node

Moreover, additional custom information can be retrieved by querying the properties of the hardware. The PAL interface provides means for custom implementations to pull additional information by using name-value pairs that can host any kind of information about the hosting platform. For example, these properties can contain additional information about the processor, such as the model and family, or additional data about the process running the container.

5.2.2 Fabric services

Fabric Services define the lowest level of the software stack representing the Aneka Container. They provide access to the resource-provisioning subsystem and to the monitoring facilities implemented in Aneka. Resource-provisioning services are in charge of dynamically providing new nodes on demand by relying on virtualization technologies, while monitoring services allow for hardware profiling and implement a basic monitoring infrastructure that can be used by all the services installed in the container.

5.2.2.1 Profiling and monitoring

Profiling and monitoring services are mostly exposed through the *Heartbeat*, *Monitoring*, and *Reporting Services*. The first makes available the information that is collected through the PAL; the other two implement a generic infrastructure for monitoring the activity of any service in the Aneka Cloud.

The Heartbeat Service periodically collects the dynamic performance information about the node and publishes this information to the membership service in the Aneka Cloud. These data are collected by the index node of the Cloud, which makes them available for services such as reservations and scheduling in order to optimize the use of a heterogeneous infrastructure. As already discussed, basic information about memory, disk space, CPU, and operating system is collected. Moreover, additional data are pulled into the "alive" message, such as information about the installed software in the system and any other useful information. More precisely, the infrastructure has been designed to carry over any type of data that can be expressed by means of text-valued properties. As previously noted, the information published by the Heartbeat Service is mostly concerned with the properties of the node. A specific component, called *Node Resolver*, is in charge of collecting these data and making them available to the Heartbeat Service. Aneka provides different implementations for such component in order to cover a wide variety of hosting environments. A variety of operating systems are supported with different implementations of the PAL, and different node resolvers allow Aneka to capture other types of data that do not strictly depend on the hosting operating system. For example, the retrieval of the public IP of the node is different in the case of physical machines or virtual instances hosted in the infrastructure of an IaaS provider such as EC2 or GoGrid. In virtual deployment, a different node resolver is used so that all other components of the system can work transparently.

The set of built-in services for monitoring and profiling is completed by a generic monitoring infrastructure, which allows any custom service to report its activity. This infrastructure is composed of the Reporting and Monitoring Services. The *Reporting Service* manages the store for monitored data and makes them accessible to other services or external applications for analysis purposes. On each node, an instance of the *Monitoring Service* acts as a gateway to the *Reporting Service* and forwards to it all the monitored data that has been collected on the node. Any service that wants to publish monitoring data can leverage the local monitoring service without knowing the details of the entire infrastructure. Currently several built-in services provide information through this channel:

- The *Membership Catalogue* tracks the performance information of nodes.
- The *Execution Service* monitors several time intervals for the execution of jobs.
- The *Scheduling Service* tracks the state transitions of jobs.
- The *Storage Service* monitors and makes available information about data transfer, such as upload and download times, filenames, and sizes.
- The *Resource Provisioning Service* tracks the provisioning and lifetime information of virtual nodes.

All this information can be stored on a relational database management system (RDBMS) or a flat file and can be further analyzed by specific applications. For example, the Management Console provides a view on such data for administrative purposes.

5.2.2.2 Resource management

Resource management is another fundamental feature of Aneka Clouds. It comprises several tasks: resource membership, resource reservation, and resource provisioning. Aneka provides a collection of services that are in charge of managing resources. These are the *Index Service* (or *Membership Catalogue*), *Reservation Service*, and *Resource Provisioning Service*.

The *Membership Catalogue* is Aneka's fundamental component for resource management; it keeps track of the basic node information for all the nodes that are connected or disconnected. The Membership Catalogue implements the basic services of a directory service, allowing the search for services using attributes such as names and nodes. During container startup, each instance publishes its information to the Membership Catalogue and updates it constantly during its lifetime. Services and external applications can query the membership catalogue to discover the available services and interact with them. To speed up and enhance the performance of queries, the membership catalogue is organized as a distributed database: All the queries that pertain to information maintained locally are resolved locally; otherwise the query is forwarded to the main index node, which has a global knowledge of the entire Cloud. The Membership Catalogue is also the collector of the dynamic performance data of each node, which are then sent to the local monitoring service to be persisted in the long term.

Indexing and categorizing resources are fundamental to resource management. On top of the basic indexing service, provisioning completes the set of features that are available for resource management within Aneka. Deployment of container instances and their configuration are performed by the infrastructure management layer and are not part of the Fabric Services.

Dynamic resource provisioning allows the integration and management of virtual resources leased from IaaS providers into the Aneka Cloud. This service changes the structure of the Aneka Cloud by allowing it to scale up and down according to different needs: handling node failures, ensuring the quality of service for applications, or maintaining a constant performance and throughput of the Cloud. Aneka defines a very flexible infrastructure for resource provisioning whereby it is possible to change the logic that triggers provisioning, support several back-ends, and change the runtime strategy with which a specific back-end is selected for provisioning. The resource-provisioning infrastructure built into Aneka is mainly concentrated in the *Resource Provisioning Service*, which includes all the operations that are needed for provisioning virtual instances. The implementation of the service is based on the idea of *resource pools*. A resource pool abstracts the interaction with a specific IaaS provider by exposing a common interface so that all the pools can be managed uniformly. A resource pool does not necessarily map to an IaaS provider but can be used to expose as dynamic resources a private cloud managed by a Xen Hypervisor or a collection of physical resources that are only used sporadically. The system uses an open protocol, allowing for the use of metadata to provide additional information for describing resource pools and to customize provisioning requests. This infrastructure simplifies the implementation of additional features and the support of different implementations that can be transparently integrated into the existing system.

Resource provisioning is a feature designed to support QoS requirements-driven execution of applications. Therefore, it mostly serves requests coming from the Reservation Service or the Scheduling Services. Despite this, external applications can directly leverage Aneka's resource-provisioning capabilities by dynamically retrieving a client to the service and interacting with the infrastructure to it. This extends the resource-provisioning scenarios that can be handled by Aneka, which can also be used as a virtual machine manager.

5.2.3 Foundation services

Fabric Services are fundamental services of the Aneka Cloud and define the basic infrastructure management features of the system. *Foundation Services* are related to the logical management of the distributed system built on top of the infrastructure and provide supporting services for the execution of distributed applications. All the supported programming models can integrate with and leverage these services to provide advanced and comprehensive application management. These services cover:

- Storage management for applications
- Accounting, billing, and resource pricing
- Resource reservation

Foundation Services provide a uniform approach to managing distributed applications and allow developers to concentrate only on the logic that distinguishes a specific programming model from the others. Together with the Fabric Services, Foundation Services constitute the core of the Aneka middleware. These services are mostly consumed by the execution services and Management Consoles. External applications can leverage the exposed capabilities for providing advanced application management.

5.2.3.1 Storage management

Data management is an important aspect of any distributed system, even in computing clouds. Applications operate on data, which are mostly persisted and moved in the format of files. Hence, any infrastructure that supports the execution of distributed applications needs to provide facilities for file/data transfer management and persistent storage. Aneka offers two different facilities for storage management: a centralized file storage, which is mostly used for the execution of compute-intensive applications, and a distributed file system, which is more suitable for the execution of data-intensive applications. The requirements for the two types of applications are rather different. Compute-intensive applications mostly require powerful processors and do not have high demands in terms of storage, which in many cases is used to store small files that are easily transferred from one node to another. In this scenario, a centralized storage node, or a pool of storage nodes, can constitute an appropriate solution. In contrast, data-intensive applications are characterized by large data files (gigabytes or terabytes), and the processing power required by tasks does not constitute a performance bottleneck. In this scenario, a distributed file system harnessing the storage space of all the nodes belonging to the cloud might be a better and more scalable solution.

Centralized storage is implemented through and managed by Aneka's *Storage Service*. The service constitutes Aneka's data-staging facilities. It provides distributed applications with the basic file transfer facility and abstracts the use of a specific protocol to end users and other components of the system, which are dynamically configured at runtime according to the facilities installed in the cloud. The option that is currently installed by default is normal File Transfer Protocol (FTP).

To support different protocols, the system introduces the concept of a *file channel* that identifies a pair of components: a file channel controller and a file channel handler. The *file channel controller* constitutes the server component of the channel, where files are stored and made available; the *file channel handler* represents the client component, which is used by user applications or other components of the system to upload, download, or browse files. The storage service uses the

configured file channel factory to first create the server component that will manage the storage and then create the client component on demand. User applications that require support for file transfer are automatically configured with the appropriate file channel handler and transparently upload input files or download output files during application execution. In the same way, worker nodes are configured by the infrastructure to retrieve the required files for the execution of the jobs and to upload their results.

An interesting property of the file channel abstraction is the ability to chain two different channels to move files by using two different protocols. Each file in Aneka contains metadata that helps the infrastructure select the appropriate channel for moving the file. For example, an output file whose final location is an S3 bucket can be moved from the worker node to the Storage Service using the internal FTP protocol and then can be staged out on S3 by the FTP channel controller managed by the service. The Storage Service supports the execution of task-based programming such as the *Task* and the *Thread Model* as well as *Parameter Sweep*-based applications.

Storage support for data-intensive applications is provided by means of a distributed file system. The reference model for the distributed file system is the Google File System [54], which features a highly scalable infrastructure based on commodity hardware. The architecture of the file system is based on a master node, which contains a global map of the file system and keeps track of the status of all the storage nodes, and a pool of chunk servers, which provide distributed storage space in which to store files. Files are logically organized into a directory structure but are persisted on the file system using a flat namespace based on a unique ID. Each file is organized as a collection of *chunks* that are all of the same size. File chunks are assigned unique IDs and stored on different servers, eventually replicated to provide high availability and failure tolerance. The model proposed by the Google File System provides optimized support for a specific class of applications that expose the following characteristics:

- Files are huge by traditional standards (multi-gigabytes).
- Files are modified by appending new data rather than rewriting existing data.
- There are two kinds of major workloads: large streaming reads and small random reads.
- It is more important to have a sustained bandwidth than a low latency.

Moreover, given the huge number of commodity machines that the file system harnesses together, failure (process or hardware failure) is the norm rather than an exception. These characteristics strongly influenced the design of the storage, which provides the best performance for applications specifically designed to operate on data as described. Currently, the only programming model that makes use of the distributed file system is *MapReduce* [55], which has been the primary reason for the Google File System implementation. Aneka provides a simple distributed file system (DFS), which relies on the file system services of the Windows operating system.

5.2.3.2 Accounting, billing, and resource pricing

Accounting Services keep track of the status of applications in the Aneka Cloud. The collected information provides a detailed breakdown of the distributed infrastructure usage and is vital for the proper management of resources.

The information collected for accounting is primarily related to infrastructure usage and application execution. A complete history of application execution and storage as well as other resource

utilization parameters is captured and minded by the Accounting Services. This information constitutes the foundation on which users are charged in Aneka.

Billing is another important feature of accounting. Aneka is a multitenant cloud programming platform in which the execution of applications can involve provisioning additional resources from commercial IaaS providers. Aneka Billing Service provides detailed information about each user's usage of resources, with the associated costs. Each resource can be priced differently according to the set of services that are available on the corresponding Aneka container or the installed software in the node. The accounting model provides an integrated view of budget spent for each application, a summary view of the costs associated to a specific user, and the detailed information about the execution cost of each job.

The accounting capabilities are concentrated within the *Accounting Service* and the *Reporting Service*. The former keeps track of the information that is related to application execution, such as the distribution of jobs among the available resources, the timing of each of job, and the associated cost. The latter makes available the information collected from the monitoring services for accounting purposes: storage utilization and CPU performance. This information is primarily consumed by the Management Console.

5.2.3.3 Resource reservation

Aneka's *Resource Reservation* supports the execution of distributed applications and allows for reserving resources for exclusive use by specific applications. Resource reservation is built out of two different kinds of services: Resource Reservation and the Allocation Service. Resource Reservation keeps track of all the reserved time slots in the Aneka Cloud and provides a unified view of the system. The *Allocation Service* is installed on each node that features execution services and manages the database of information regarding the allocated slots on the local node. Applications that need to complete within a given deadline can make a reservation request for a specific number of nodes in a given timeframe. If it is possible to satisfy the request, the Reservation Service will return a reservation identifier as proof of the resource booking. During application execution, such an identifier is used to select the nodes that have been reserved, and they will be used to execute the application. On each reserved node, the execution services will check with the Allocation Service that each job has valid permissions to occupy the execution timeline by verifying the reservation identifier. Even though this is the general reference model for the reservation infrastructure, Aneka allows for different implementations of the service, which mostly vary in the protocol that is used to reserve resources or the parameters that can be specified while making a reservation request. Different protocol and strategies are integrated in a completely transparent manner, and Aneka provides extensible APIs for supporting advanced services. At the moment, the framework supports three different implementations:

- *Basic Reservation*. Features the basic capability to reserve execution slots on nodes and implements the *alternate offers* protocol, which provides alternative options in case the initial reservation requests cannot be satisfied.
- *Libra Reservation.* Represents a variation of the previous implementation that features the ability to price nodes differently according to their hardware capabilities.
- *Relay Reservation.* Constitutes a very thin implementation that allows a resource broker to reserve nodes in Aneka Clouds and control the logic with which these nodes are reserved. This

implementation is useful in integration scenarios in which Aneka operates in an intercloud environment.

Resource reservation is fundamental to ensuring the quality of service that is negotiated for applications. It allows Aneka to have a predictable environment in which applications can complete within the deadline or not be executed at all. The assumptions made by the reservation service for accepting reservation requests are based on the static allocation of such requests to the existing physical (or virtual) infrastructure available at the time of the requests and by taking into account the current and future load. This solution is sensitive to node failures that could make Aneka unable to fulfill the service-level agreement (SLA) made with users. Specific implementations of the service tend to delay the allocation of nodes to reservation requests as late as possible in order to cope with temporary failures or limited outages, but in the case of serious outages in which the remaining available nodes are not able to cover the demand, this strategy is not enough. In this case, resource provisioning can provide an effective solution: Additional nodes can be provisioned from external resource providers in order to cover the outage and meet the SLA defined for applications. The current implementation of the resource reservation infrastructure leverages the provisioning capabilities of the fabric layer when the current availability in the system is not able to address the reservation requests already confirmed. Such behavior solves the problems of both insufficient resources and temporary failures.

5.2.4 Application services

Application Services manage the execution of applications and constitute a layer that differentiates according to the specific programming model used for developing distributed applications on top of Aneka. The types and the number of services that compose this layer for each of the programming models may vary according to the specific needs or features of the selected model. It is possible to identify two major types of activities that are common across all the supported models: scheduling and execution. Aneka defines a reference model for implementing the runtime support for programming models that abstracts these two activities in corresponding services: the *Scheduling Service* and the *Execution Service*. Moreover, it also defines base implementations that can be extended in order to integrate new models.

5.2.4.1 Scheduling

Scheduling Services are in charge of planning the execution of distributed applications on top of Aneka and governing the allocation of jobs composing an application to nodes. They also constitute the integration point with several other Foundation and Fabric Services, such as the Resource Provisioning Service, the Reservation Service, the Accounting Service, and the Reporting Service. Common tasks that are performed by the scheduling component are the following:

- Job to node mapping
- Rescheduling of failed jobs
- Job status monitoring
- Application status monitoring

Aneka does not provide a centralized scheduling engine, but each programming model features its own scheduling service that needs to work in synergy with the existing services of the middleware. As already mentioned, these services mostly belong to the fabric and the foundation layers of the architecture shown in Figure 5.2. The possibility of having different scheduling engines for different models gives great freedom in implementing scheduling and resource allocation strategies but, at the same time, requires a careful design of use of shared resources. In this scenario, common situations that have to be appropriately managed are the following: multiple jobs sent to the same node at the same time; jobs without reservations sent to reserved nodes; and jobs sent to nodes where the required services are not installed. Aneka's Foundation Services provide sufficient information to avoid these cases, but the runtime infrastructure does not feature specific policies to detect these conditions and provide corrective action. The current design philosophy in Aneka is to keep the scheduling engines completely separate from each other and to leverage existing services when needed. As a result, it is possible to enforce that only one job per programming model is run on each node at any given time, but the execution of applications is not mutually exclusive unless Resource Reservation is used.

5.2.4.2 Execution

Execution Services control the execution of single jobs that compose applications. They are in charge of setting up the runtime environment hosting the execution of jobs. As happens for the scheduling services, each programming model has its own requirements, but it is possible to identify some common operations that apply across all the range of supported models:

- Unpacking the jobs received from the scheduler
- Retrieval of input files required for job execution
- Sandboxed execution of jobs
- Submission of output files at the end of execution
- Execution failure management (i.e., capturing sufficient contextual information useful to identify the nature of the failure)
- Performance monitoring
- Packing jobs and sending them back to the scheduler

Execution Services constitute a more self-contained unit with respect to the corresponding scheduling services. They handle less information and are required to integrate themselves only with the Storage Service and the local Allocation and Monitoring Services. Aneka provides a reference implementation of execution services that has built-in integration with all these services, and currently two of the supported programming models specialize on the reference implementation.

Application Services constitute the runtime support of the programming model in the Aneka Cloud. Currently there are several supported models:

- *Task Model*. This model provides the support for the independent "bag of tasks" applications and many computing tasks. In this model, an application is modeled as a collection of tasks that are independent from each other and whose execution can be sequenced in any order.
- *Thread Model*. This model provides an extension to the classical multithreaded programming to a distributed infrastructure and uses the abstraction of *Thread* to wrap a method that is executed remotely.

- *MapReduce Model*. This is an implementation of MapReduce as proposed by Google on top of Aneka.
- *Parameter Sweep Model*. This model is a specialization of the Task Model for applications that can be described by a template task whose instances are created by generating different combinations of parameters, which identify a specific point into the domain of interest.

Other programming models have been developed for internal use and are at an experimental stage. These are the Dataflow Model [56], the Message-Passing Interface, and the Actor Model [57].

5.3 Building Aneka clouds

Aneka is primarily a platform for developing distributed applications for clouds. As a software platform it requires infrastructure on which to be deployed; this infrastructure needs to be managed. Infrastructure management tools are specifically designed for this task, and building clouds is one of the primary tasks of administrators. Aneka supports various deployment models for public, private, and hybrid clouds.

5.3.1 Infrastructure organization

Figure 5.3 provides an overview of Aneka Clouds from an infrastructure point of view. The scenario is a reference model for all the different deployments Aneka supports. A central role is played by the Administrative Console, which performs all the required management operations. A fundamental element for Aneka Cloud deployment is constituted by *repositories*. A repository provides storage for all the libraries required to lay out and install the basic Aneka platform. These libraries constitute the software image for the node manager and the container programs. Repositories can make libraries available through a variety of communication channels, such as HTTP, FTP, common file sharing, and so on. The Management Console can manage multiple repositories and select the one that best suits the specific deployment. The infrastructure is deployed by harnessing a collection of nodes and installing on them the Aneka node manager, also called the *Aneka daemon*. The daemon constitutes the remote management service used to deploy and control container instances. The collection of resulting containers identifies the Aneka Cloud.

From an infrastructure point of view, the management of physical or virtual nodes is performed uniformly as long as it is possible to have an Internet connection and remote administrative access to the node. A different scenario is constituted by the dynamic provisioning of virtual instances; these are generally created by prepackaged images already containing an installation of Aneka, which only need to be configured to join a specific Aneka Cloud. It is also possible to simply install the container or install the Aneka daemon, and the selection of the proper solution mostly depends on the lifetime of virtual resources.

5.3.2 Logical organization

The logical organization of Aneka Clouds can be very diverse, since it strongly depends on the configuration selected for each of the container instances belonging to the Cloud. The most common

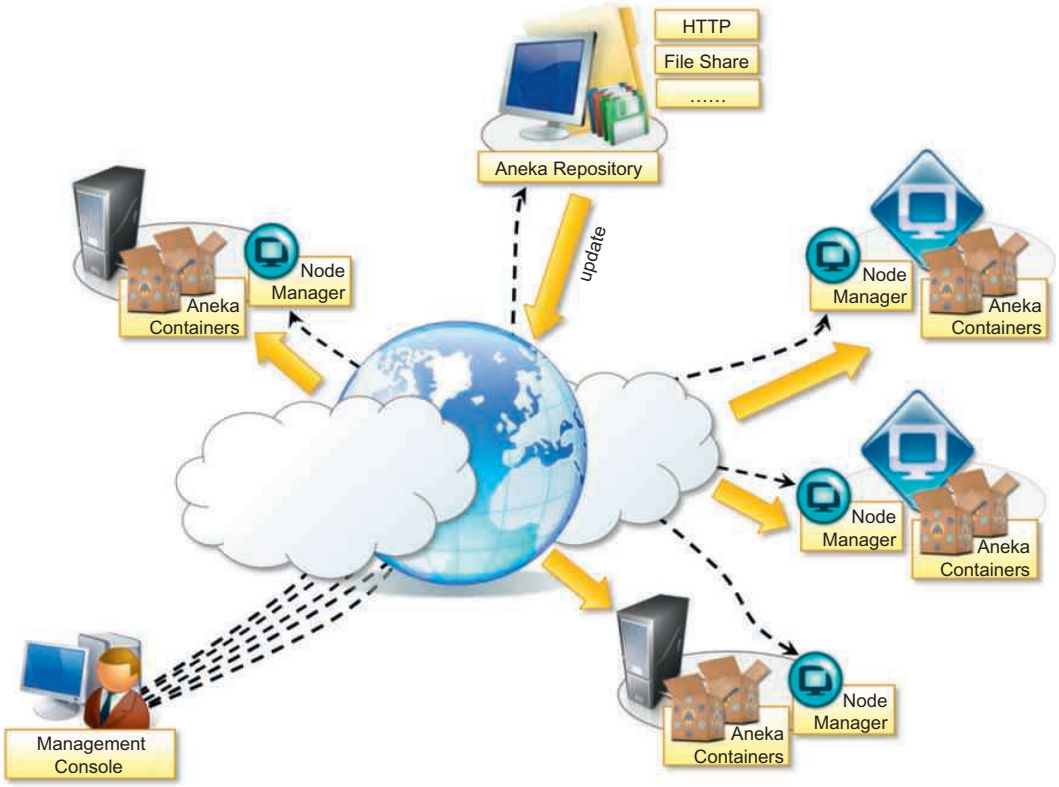


FIGURE 5.3

Aneka cloud infrastructure overview.

scenario is to use a master-worker configuration with separate nodes for storage, as shown in Figure 5.4.

The master node features all the services that are most likely to be present in one single copy and that provide the intelligence of the Aneka Cloud. What specifically characterizes a node as a master node is the presence of the *Index Service* (or Membership Catalogue) configured in master mode; all the other services, except for those that are mandatory, might be present or located in other nodes. A common configuration of the master node is as follows:

- Index Service (master copy)
- Heartbeat Service
- Logging Service
- Reservation Service
- Resource Provisioning Service
- Accounting Service
- Reporting and Monitoring Service
- Scheduling Services for the supported programming models

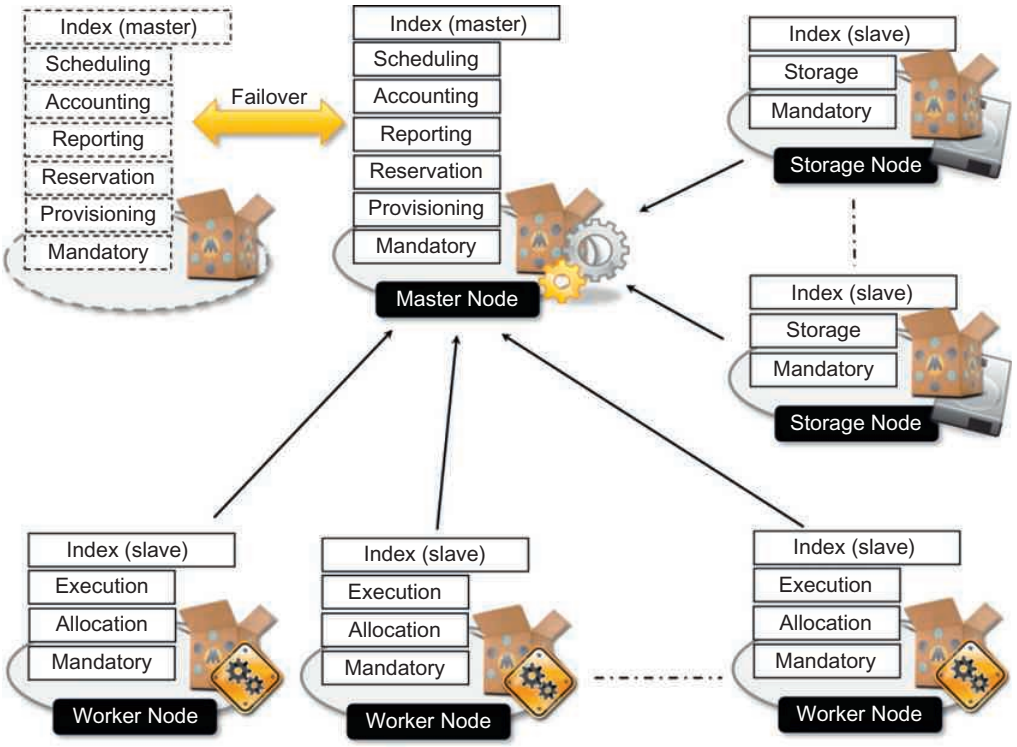


FIGURE 5.4

Logical organization of an Aneka cloud.

The master node also provides connection to an RDBMS facility where the state of several services is maintained. For the same reason, all the scheduling services are maintained in the master node. They share the application store that is normally persisted on the RDBMS in order to provide a fault-tolerant infrastructure. The master configuration can then be replicated in several nodes to provide a highly available infrastructure based on the failover mechanism.

The worker nodes constitute the workforce of the Aneka Cloud and are generally configured for the execution of applications. They feature the mandatory services and the specific execution services of each of the supported programming models in the Cloud. A very common configuration is the following:

- Index Service
- Heartbeat Service
- Logging Service
- Allocation Service
- Monitoring Service
- Execution Services for the supported programming models

A different option is to partition the pool of worker nodes with a different selection of execution services in order to balance the load between programming models and reserve some nodes for a specific class of applications.

Storage nodes are optimized to provide storage support to applications. They feature, among the mandatory and usual services, the presence of the Storage Service. The number of storage nodes strictly depends on the predicted workload and storage consumption of applications. Storage nodes mostly reside on machines that have considerable disk space to accommodate a large quantity of files. The common configuration of a storage node is the following:

- Index Service
- Heartbeat Service
- Logging Service
- Monitoring Service
- Storage Service

In specific cases, when the data transfer requirements are not demanding, there might be only one storage node. In some cases, for very small deployments, there is no need to have a separate storage node, and the Storage Service is installed and hosted on the master node.

All nodes are registered with the master node and transparently refer to any failover partner in the case of a high-availability configuration.

5.3.3 Private cloud deployment mode

A private deployment mode is mostly constituted by local physical resources and infrastructure management software providing access to a local pool of nodes, which might be virtualized. In this scenario Aneka Clouds are created by harnessing a heterogeneous pool of resources such has desktop machines, clusters, or workstations. These resources can be partitioned into different groups, and Aneka can be configured to leverage these resources according to application needs. Moreover, leveraging the Resource Provisioning Service, it is possible to integrate virtual nodes provisioned from a local resource pool managed by systems such as XenServer, Eucalyptus, and OpenStack.

Figure 5.5 shows a common deployment for a private Aneka Cloud. This deployment is acceptable for a scenario in which the workload of the system is predictable and a local virtual machine manager can easily address excess capacity demand. Most of the Aneka nodes are constituted of physical nodes with a long lifetime and a static configuration and generally do not need to be reconfigured often. The different nature of the machines harnessed in a private environment allows for specific policies on resource management and usage that can be accomplished by means of the Reservation Service. For example, desktop machines that are used during the day for office automation can be exploited outside the standard working hours to execute distributed applications. Workstation clusters might have some specific legacy software that is required for supporting the execution of applications and should be preferred for the execution of applications with special requirements.

5.3.4 Public cloud deployment mode

Public Cloud deployment mode features the installation of Aneka master and worker nodes over a completely virtualized infrastructure that is hosted on the infrastructure of one or more resource

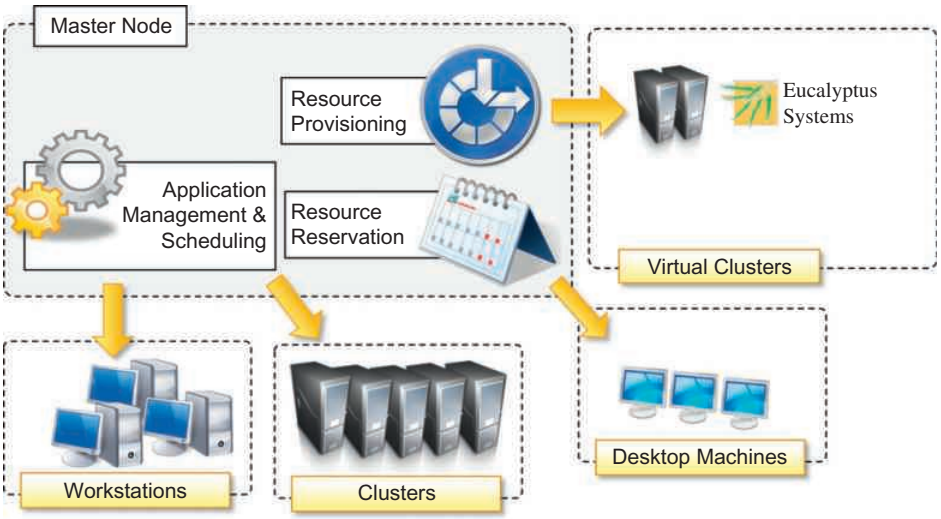


FIGURE 5.5

Private cloud deployment.

providers such as Amazon EC2 or GoGrid. In this case it is possible to have a static deployment where the nodes are provisioned beforehand and used as though they were real machines. This deployment merely replicates a classic Aneka installation on a physical infrastructure without any dynamic provisioning capability. More interesting is the use of the elastic features of IaaS providers and the creation of a Cloud that is completely dynamic. Figure 5.6 provides an overview of this scenario.

The deployment is generally contained within the infrastructure boundaries of a single IaaS provider. The reasons for this are to minimize the data transfer between different providers, which is generally priced at a higher cost, and to have better network performance. In this scenario it is possible to deploy an Aneka Cloud composed of only one node and to completely leverage dynamic provisioning to elastically scale the infrastructure on demand. A fundamental role is played by the Resource Provisioning Service, which can be configured with different images and templates to instantiate. Other important services that have to be included in the master node are the Accounting and Reporting Services. These provide details about resource utilization by users and applications and are fundamental in a multitenant Cloud where users are billed according to their consumption of Cloud capabilities.

Dynamic instances provisioned on demand will mostly be configured as worker nodes, and, in the specific case of Amazon EC2, different images featuring a different hardware setup can be made available to instantiate worker containers. Applications with specific requirements for computing capacity or memory can provide additional information to the scheduler that will trigger the appropriate provisioning request. Application execution is not the only use of dynamic instances; any service requiring elastic scaling can leverage dynamic provisioning. Another example is the Storage Service. In multitenant Clouds, multiple applications can leverage the support for storage; in this

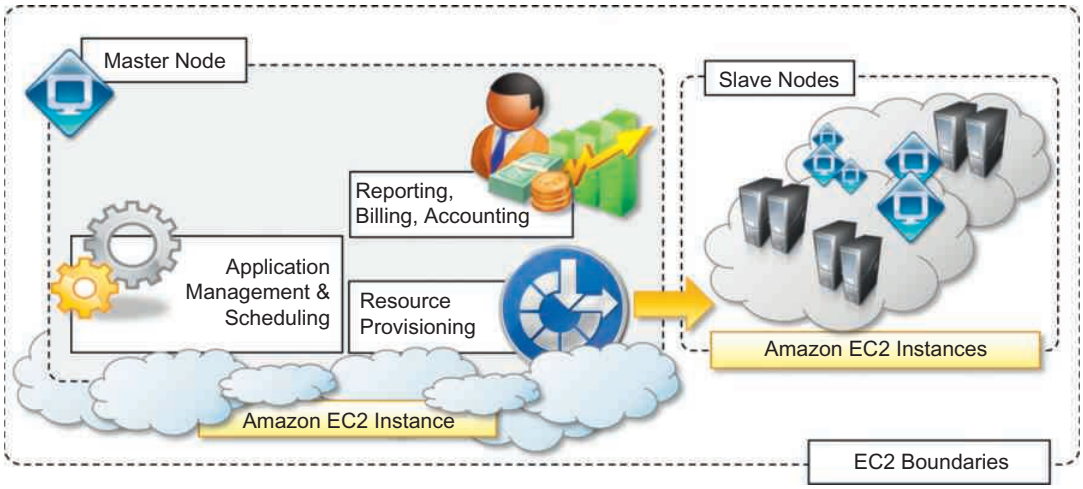


FIGURE 5.6

Public Aneka cloud deployment.

scenario it is then possible to introduce bottlenecks or simply reach the quota limits allocated for storage on the node. Dynamic provisioning can easily solve this issue as it does for increasing the computing capability of an Aneka Cloud.

Deployments using different providers are unlikely to happen because of the data transfer costs among providers, but they might be a possible scenario for federated Aneka Clouds [58]. In this scenario resources can be shared or leased among providers under specific agreements and more convenient prices. In this case the specific policies installed in the Resource Provisioning Service can discriminate among different resource providers, mapping different IaaS providers to provide the best solution to a provisioning request.

5.3.5 Hybrid cloud deployment mode

The hybrid deployment model constitutes the most common deployment of Aneka. In many cases, there is an existing computing infrastructure that can be leveraged to address the computing needs of applications. This infrastructure will constitute the static deployment of Aneka that can be elastically scaled on demand when additional resources are required. An overview of this deployment is presented in Figure 5.7.

This scenario constitutes the most complete deployment for Aneka that is able to leverage all the capabilities of the framework:

- Dynamic Resource Provisioning
- Resource Reservation
- Workload Partitioning
- Accounting, Monitoring, and Reporting

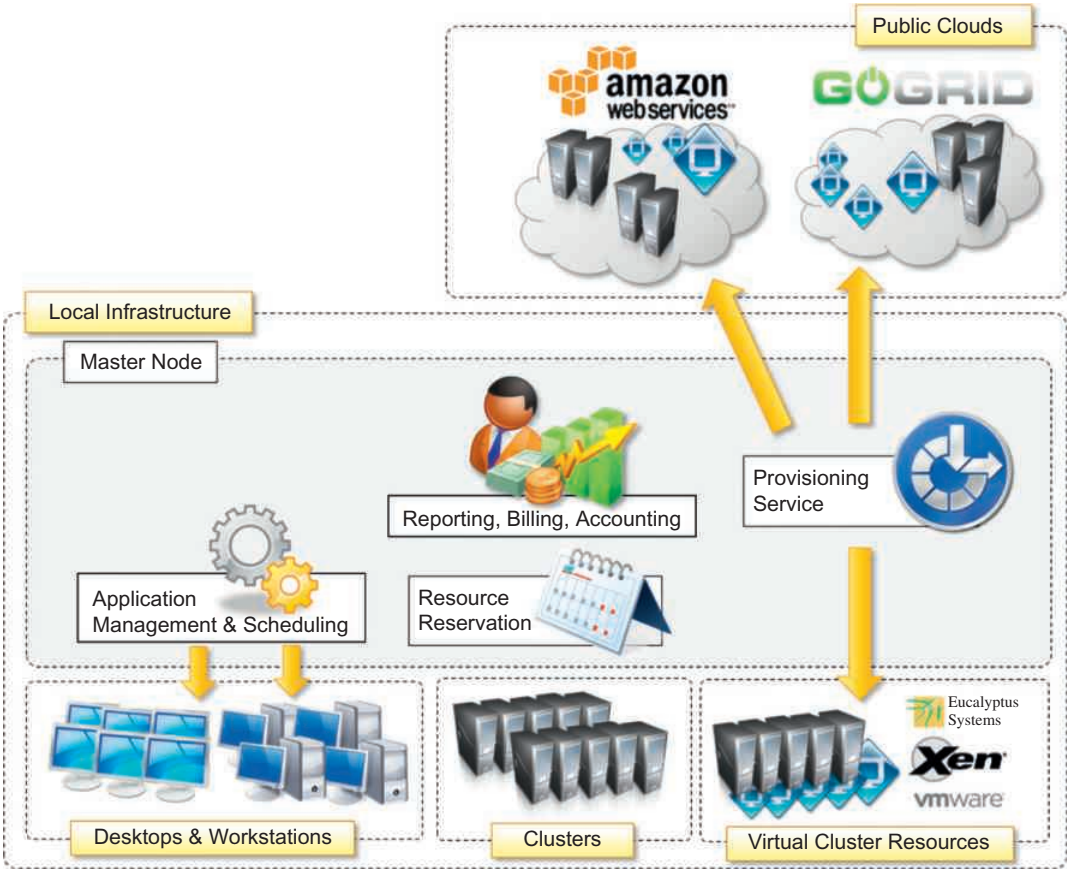


FIGURE 5.7

Hybrid cloud deployment.

Moreover, if the local premises offer some virtual machine management capabilities, it is possible to provide a very efficient use of resources, thus minimizing the expenditure for application execution.

In a hybrid scenario, heterogeneous resources can be used for different purposes. As we discussed in the case of a private cloud deployment, desktop machines can be reserved for low priority workload outside the common working hours. The majority of the applications will be executed on workstations and clusters, which are the nodes that are constantly connected to the Aneka Cloud. Any additional computing capability demand can be primarily addressed by the local virtualization facilities, and if more computing power is required, it is possible to leverage external IaaS providers.

Different from the Aneka Public Cloud deployment is the case in which it makes more sense to leverage a variety of resource providers to provision virtual resources. Since part of the infrastructure is local, a cost in data transfer to the external IaaS infrastructure cannot be avoided. It is then important to select the most suitable option to address application needs. The Resource Provisioning

Service implemented in Aneka exposes the capability of leveraging several resource pools at the same time and configuring specific policies to select the most appropriate pool for satisfying a provisioning request. These features simplify the development of custom policies that can better serve the needs of a specific hybrid deployment.

5.4 Cloud programming and management

Aneka's primary purpose is to provide a scalable middleware product in which to execute distributed applications. Application development and management constitute the two major features that are exposed to developers and system administrators. To simplify these activities, Aneka provides developers with a comprehensive and extensible set of APIs and administrators with powerful and intuitive management tools. The APIs for development are mostly concentrated in the Aneka SDK; management tools are exposed through the Management Console.

5.4.1 Aneka SDK

Aneka provides APIs for developing applications on top of existing programming models, implementing new programming models, and developing new services to integrate into the Aneka Cloud. The development of applications mostly focuses on the use of existing features and leveraging the services of the middleware, while the implementation of new programming models or new services enriches the features of Aneka. The SDK provides support for both programming models and services by means of the *Application Model* and the *Service Model*. The former covers the development of applications and new programming models; the latter defines the general infrastructure for service development.

5.4.1.1 Application model

Aneka provides support for distributed execution in the Cloud with the abstraction of programming models. A programming model identifies both the abstraction used by the developers and the runtime support for the execution of programs on top of Aneka. The *Application Model* represents the minimum set of APIs that is common to all the programming models for representing and programming distributed applications on top of Aneka. This model is further specialized according to the needs and the particular features of each of the programming models.

An overview of the components that define the Aneka Application Model is shown in Figure 5.8. Each distributed application running on top of Aneka is an instance of the *ApplicationBase* $<M>$ class, where *M* identifies the specific type of application manager used to control the application. Application classes constitute the developers' view of a distributed application on Aneka Clouds, whereas application managers are internal components that interact with Aneka Clouds in order to monitor and control the execution of the application. Application managers are also the first element of specialization of the model and vary according to the specific programming model used.

Whichever the specific model used, a distributed application can be conceived as a set of tasks for which the collective execution defines the execution of the application on the Cloud. Aneka further specializes applications into two main categories: (1) applications whose tasks are generated

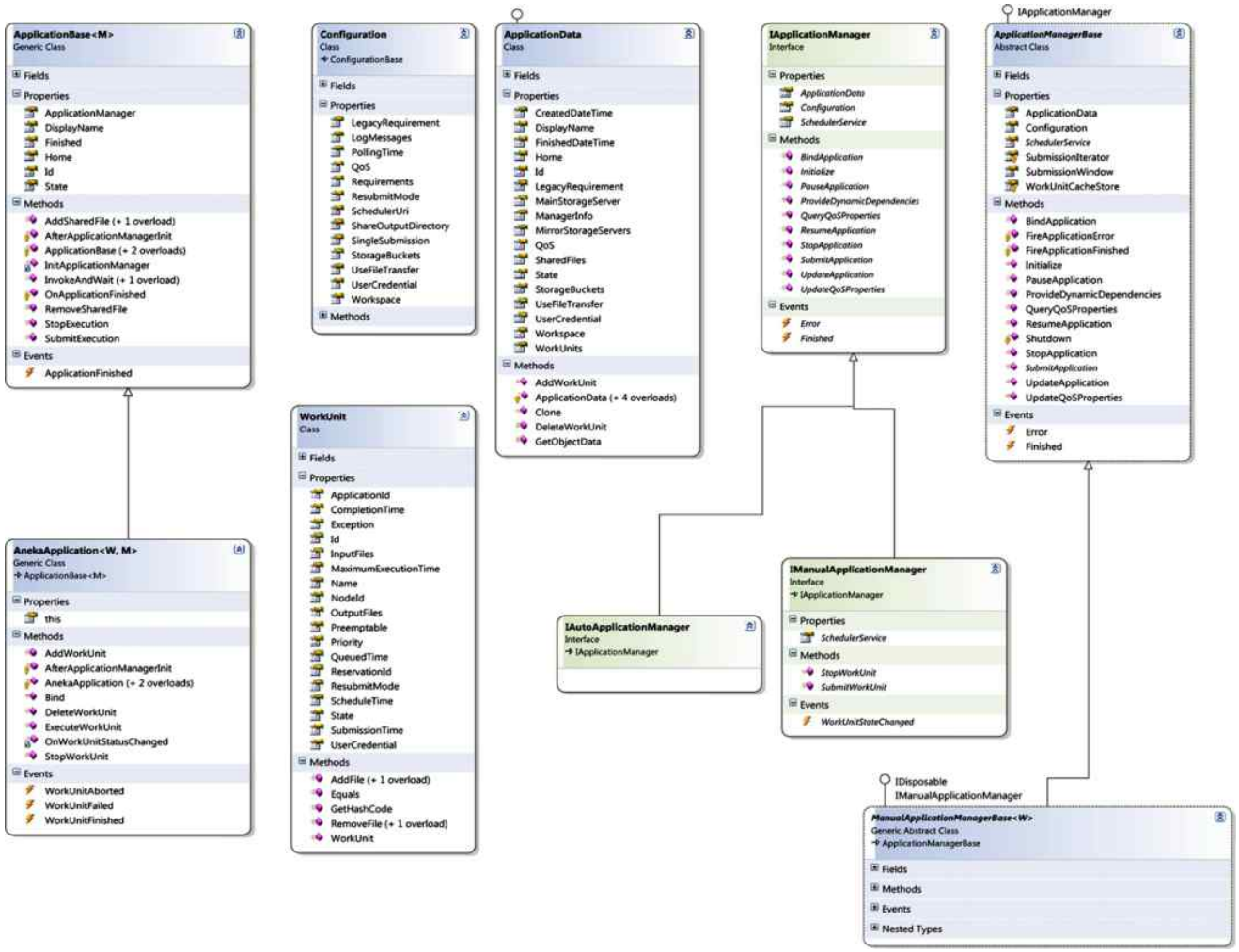


FIGURE 5.8

The Aneka application model.

by the user and (2) applications whose tasks are generated by the runtime infrastructure. These two categories generally correspond to different application base classes and different implementations of the application manager.

The first category is the most common and it is used as a reference for several programming models supported by Aneka: the *Task Model*, the *Thread Model*, and the *Parameter Sweep Model*. Applications that fall into this category are composed of a collection of units of work submitted by the user and represented by the *WorkUnit* class. Each unit of work can have input and output files, the transfer of which is transparently managed by the runtime. The specific type of *WorkUnit* class used to represent the unit of work depends on the programming model used (*AnekaTask* for the *Task Model* and *AnekaThread* for the *Thread Model*). All the applications that fall into this category inherit or are instances *of AnekaApplication* $< W,M >$, where W is the specific type of *WorkUnit* class used, and M is the type of application manager used to implement the *IManualApplicationManager* interface.

The second category covers the case of *MapReduce* and all those other scenarios in which the units of work are generated by the runtime infrastructure rather than the user. In this case there is no common unit-of-work class used, and the specific classes used by application developers strictly depend on the requirements of the programming model used. For example, in the case of the *MapReduce* programming model, developers express their distributed applications in terms of two functions, *map* and *reduce*; hence, the *MapReduceApplication* class provides an interface for specifying the *Mapper* $< K,V >$ and *Reducer* $< K,V >$ types and the input files required by the application. Other programming models might have different requirements and expose different interfaces. For this reason there are no common base types for this category except for *ApplicationBase* $< M >$, where M implements *IAutoApplicationManager*.

A set of additional classes completes the object model. Among these classes, the most notable are the *Configuration* class, which is used to specify the settings required to initialize the application and customize its behavior, and the *ApplicationData* class, which contains the runtime information of the application.

Table 5.1 summarizes the features that are available in the Aneka Application Model and the way they reflect into the supported programming model. The model has been designed to be extensible, and these classes can be used as a starting point to implement a new programming model. This can be done by augmenting the features (or specializing) an existing implementation of a

Table 5.1 Aneka's Application Model Features

Category	Description	Base Application Type	Work Units?	Programming Models
Manual	Units of work are generated by the user and submitted through the application.	*AnekaApplication* $< W,M >$ *IManualApplicationManager* $< W >$ *ManualApplicationManager* $< W >$	Yes	Task Model Thread Model Parameter Sweep Model
Auto	Units of work are generated by the runtime infrastructure and managed internally.	*ApplicationBase* $< M >$ *IAutoApplicationManager*	No	*MapReduce* Model

programming model or by using the base classes to define new models and abstractions. For example, the Parameter Sweep Model is a specialization of the Task Model, and it has been implemented in the context of management of applications on Aneka. It is achieved by providing a different interface to end users who just need to define a template task and the parameters that customize it.

5.4.1.2 Service model

The Aneka *Service Model* defines the basic requirements to implement a service that can be hosted in an Aneka Cloud. The container defines the runtime environment in which services are hosted. Each service that is hosted in the container must be compliant with the *IService* interface, which exposes the following methods and properties:

- Name and status
- Control operations such as *Start*, *Stop*, *Pause*, and *Continue* methods
- Message handling by means of the *HandleMessage* method

Specific services can also provide clients if they are meant to directly interact with end users. Examples of such services might be Resource Provisioning and Resource Reservation Services, which ship their own clients for allowing resource provisioning and reservation. Apart from control operations, which are used by the container to set up and shut down the service during the container life cycle, the core logic of a service resides in its message-processing functionalities that are contained in the *HandleMessage* method. Each operation that is requested to a service is triggered by a specific message, and results are communicated back to the caller by means of messages.

Figure 5.9 describes the reference life cycle of each service instance in the Aneka container. The shaded balloons indicate transient states; the white balloons indicate steady states. A service instance can initially be in the *Unknown* or *Initialized* state, a condition that refers to the creation of the service instance by invoking its constructor during the configuration of the container. Once the container is started, it will iteratively call the *Start* method on each service method. As a result the service instance is expected to be in a *Starting* state until the startup process is completed, after which it will exhibit the *Running* state. This is the condition in which the service will last as long as the container is active and running. This is the only state in which the service is able to process messages. If an exception occurs while starting the service, it is expected that the service will fall back to the *Unknown* state, thus signaling an error.

When a service is running it is possible to pause its activity by calling the *Pause* method and resume it by calling *Continue*. As described in the figure, the service moves first into the *Pausing* state, thus reaching the *Paused* state. From this state, it moves into the *Resuming* state while restoring its activity to return to the *Running* state. Not all the services need to support the pause/continue operations, and the current implementation of the framework does not feature any service with these capabilities.

When the container shuts down, the *Stop* method is iteratively called on each service running, and services move first into the transient *Stopping* state to reach the final *Stopped* state, where all resources that were initially allocated have been released.

Aneka provides a default base class for simplifying service implementation and a set of guidelines that service developers should follow to design and implement services that are compliant with Aneka. In particular, the guidelines define a *ServiceBase* class that can be further extended to

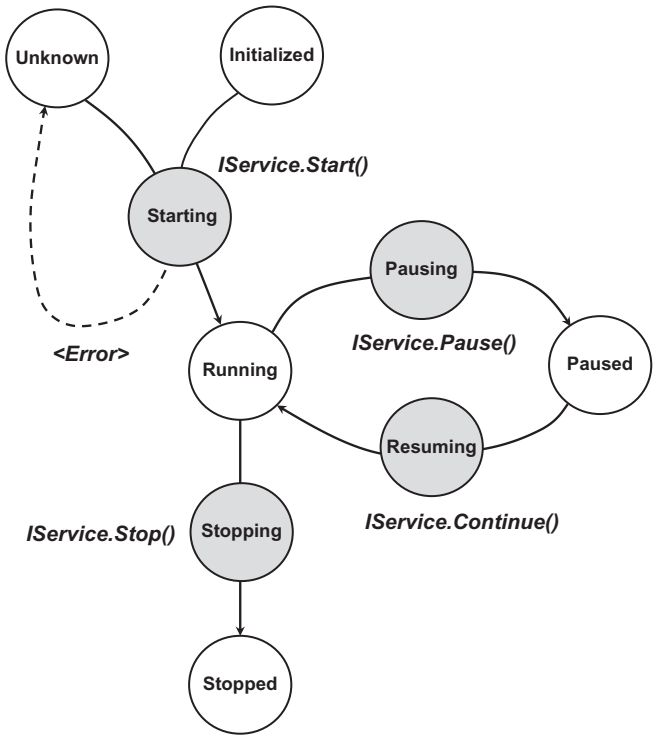


FIGURE 5.9

Service life cycle.

provide a proper implementation. This class is the base class of several services in the framework and provides some built-in features:

- Implementation of the basic properties exposed by *IService*
- Implementation of the control operations with logging capabilities and state control
- Built-in infrastructure for delivering a service specific client
- Support for service monitoring

Developers are provided with template methods for specializing the behavior of control operations, implementing their own message-processing logic, and providing a service-specific client.

Aneka uses a strongly typed message-passing communication model, whereby each service defines its own messages, which are in turn the only ones that the service is able to process. As a result, developers who implement new services in Aneka need also to define the type of messages that the services will use to communicate with services and clients. Each message type inherits from the base class *Message* defining common properties such as:

- Source node and target node
- Source service and target service
- Security credentials

Additional properties are added to carry the specific information for each type. Messages are generally used inside the Aneka infrastructure. In case the service exposes features directly used by applications, they may expose a service client that provides an object-oriented interface to the operations exposed by the service. Aneka features a ready-to-use infrastructure for dynamically injecting service clients into applications by querying the middleware. Services inheriting from the *ServiceBase* class already support such a feature and only need to define an interface and a specific implementation for the service client. Service clients are useful to integrate Aneka services into existing applications that do not necessarily need support for the execution of distributed applications or require access to additional services.

Aneka also provides advanced capabilities for service configuration. Developers can define editors and configuration classes that allow Aneka's management tools to integrate the configuration of services within the common workflow required by the container configuration.

5.4.2 Management tools

Aneka is a pure PaaS implementation and requires virtual or physical hardware to be deployed. Hence, infrastructure management, together with facilities for installing logical clouds on such infrastructure, is a fundamental feature of Aneka's management layer. This layer also includes capabilities for managing services and applications running in the Aneka Cloud.

5.4.2.1 Infrastructure management

Aneka leverages virtual and physical hardware in order to deploy Aneka Clouds. Virtual hardware is generally managed by means of the Resource Provisioning Service, which acquires resources on demand according to the need of applications, while physical hardware is directly managed by the Administrative Console by leveraging the Aneka management API of the PAL. The management features are mostly concerned with the provisioning of physical hardware and the remote installation of Aneka on the hardware.

5.4.2.2 Platform management

Infrastructure management provides the basic layer on top of which Aneka Clouds are deployed. The creation of Clouds is orchestrated by deploying a collection of services on the physical infrastructure that allows the installation and the management of containers. A collection of connected containers defines the platform on top of which applications are executed. The features available for platform management are mostly concerned with the logical organization and structure of Aneka Clouds. It is possible to partition the available hardware into several Clouds variably configured for different purposes. Services implement the core features of Aneka Clouds and the management layer exposes operations for some of them, such as Cloud monitoring, resource provisioning and reservation, user management, and application profiling.

5.4.2.3 Application management

Applications identify the user contribution to the Cloud. The management APIs provide administrators with monitoring and profiling features that help them track the usage of resources and relate them to users and applications. This is an important feature in a cloud computing scenario in which

users are billed for their resource usage. Aneka exposes capabilities for giving summary and detailed information about application execution and resource utilization.

All these features are made accessible through the Aneka Cloud Management Studio, which constitutes the main Administrative Console for the Cloud.

SUMMARY

In this chapter we introduced Aneka, a platform for application programming in the cloud. Aneka is a pure PaaS implementation of the Cloud Computing Reference Model and constitutes a middleware product that enables the creation of computing clouds on top of heterogeneous hardware: desktop machines, clusters, and public virtual resources.

One of the key aspects of Aneka's framework is its configurable runtime environment, which allows for the creation of a service-based middleware where applications are executed. A fundamental element of the infrastructure is the container, which represents the deployment unit of Aneka Clouds. The container hosts a collection of services that define the capabilities of the middleware. Fundamental services in the Aneka middleware are:

- Fabric Services for monitoring, resource provisioning, hardware profiling, and membership
- Foundation Services for storage, resource reservation, billing, accounting, and reporting
- Application Services for scheduling and execution

From an application programming point of view, Aneka provides the capability of supporting different programming models, thus allowing developers to express distributed applications with different abstractions. The framework currently supports three different models: independent "bag of tasks" applications, multithreaded applications, and *MapReduce*.

The infrastructure is extensible, and Aneka provides both an application model and a service model that can be easily extended to integrate new services and programming models.

Review questions

1. Describe in a few words the main characteristics of Aneka.
2. What is the Aneka container and what is its use?
3. Which types of services are hosted inside the Aneka container?
4. Describe Aneka's resource-provisioning capabilities.
5. Describe the storage architecture implemented in Aneka.
6. What is a programming model?
7. List the programming models supported by Aneka.
8. Which are the components that compose the Aneka infrastructure?
9. Discuss the logical organization of an Aneka Cloud.
10. Which services are hosted in a worker node?
11. Discuss the private deployment of Aneka Clouds.
12. Discuss the public deployment of Aneka Clouds.

13. Discuss the role of dynamic provisioning in hybrid deployments.
14. Which facilities does Aneka provide for development?
15. Discuss the major features of the Aneka Application Model.
16. Discuss the major features of the Aneka Service Model.
17. Describe the features of the Aneka management tools in terms of infrastructure, platform, and applications.

This page intentionally left blank