UNIT III: **Cloud Platform Architecture:**
Cloud Computing and service Models, Public Cloud Platforms, Service Oriented Architecture, Programming on Amazon AWS and Microsoft Azure

1. CLOUD COMPUTING AND SERVICE MODELS:

Users can access and deploy cloud applications from anywhere in the world at very competitive costs. Virtualized cloud platforms are often built on top of large data centers. In other words, clouds aim to power the next generation of data centers by architecting them as virtual resources over automated hardware, databases, user interfaces, and application environments. In this sense, clouds grow out of the desire to build better data centers through automated resource provisioning.

1. Public, Private, and Hybrid Clouds:

The concept of cloud computing has evolved from cluster, grid, and utility computing. Cluster and grid computing leverage the use of many computers in parallel to solve problems of any size. Utility and Software as a Service (SaaS) provide computing resources as a service with the notion of pay per use.

Cloud computing leverages dynamic resources to deliver large numbers of services to end users. Cloud computing is a high-throughput computing (HTC) paradigm whereby the infrastructure provides the services through a large data center or server farms. The cloud computing model enables users to share access to resources from anywhere at any time through their connected devices. Furthermore, machine virtualization has enhanced resource utilization, increased application flexibility, and reduced the total cost of using virtualized data-center resources. The main idea is to move desktop computing to a service-oriented platform using server clusters and huge databases at data centers. Cloud computing leverages its low cost and simplicity to both providers and users.

Centralized versus Distributed Computing:

Some people argue that cloud computing is centralized computing at data centers. Others claim that cloud computing is the practice of distributed parallel computing over data-center resources. All computations in cloud applications are distributed to servers in a data center. These are mainly virtual machines (VMs) in virtual clusters created out of data-center resources. In this sense, cloud platforms are systems distributed through virtualization.

As **Figure 4.1** shows, both public clouds and private clouds are developed in the Internet. As many clouds are generated by commercial providers or by enterprises in a distributed manner, they will be interconnected over the Internet to achieve scalable and efficient computing services. Commercial cloud providers such as Amazon, Google, and Microsoft created their platforms to be distributed geographically. This distribution is partially attributed to fault tolerance, response latency reduction, and even legal reasons. Intranet-based private clouds are linked to public clouds to get additional resources.

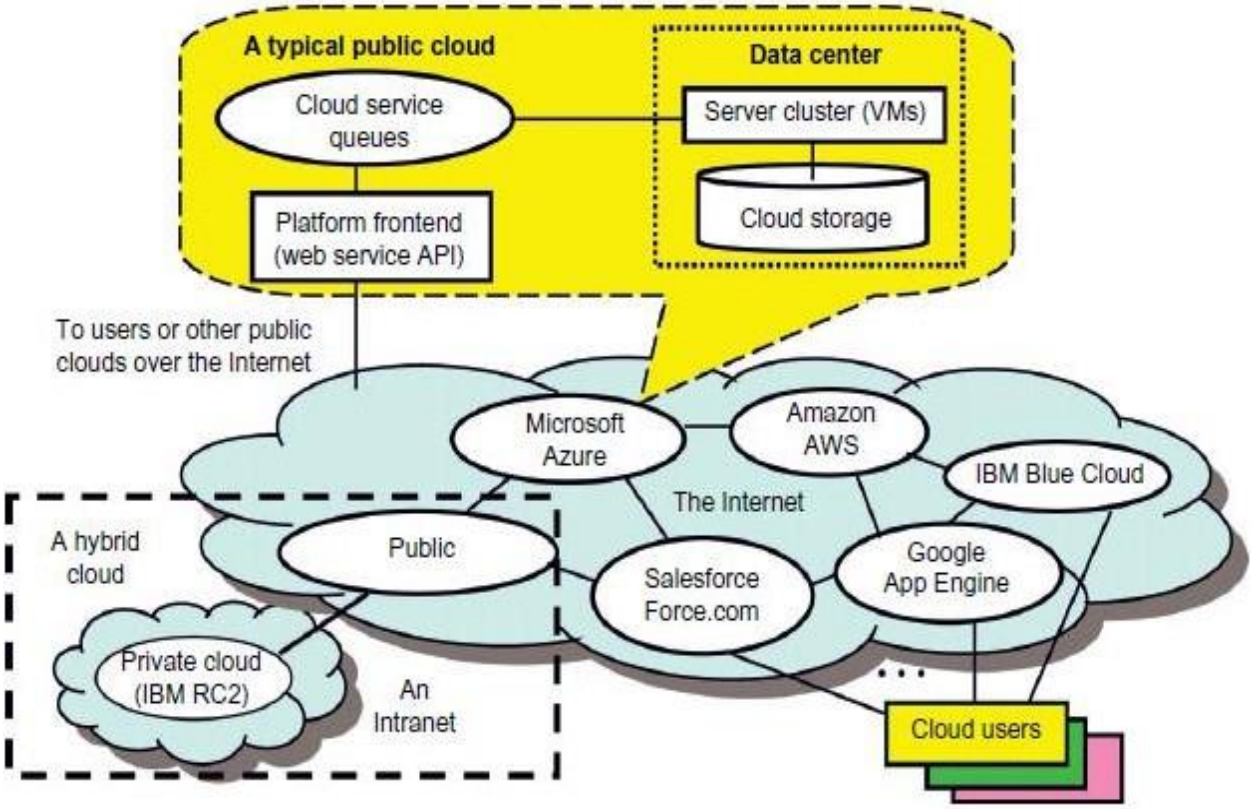


FIGURE 4.1

Public, private, and hybrid clouds illustrated by functional architecture and connectivity of representative clouds available by 2011.

Public Clouds:

A public cloud is built over the Internet and can be accessed by any user who has paid for the service. Public clouds are owned by service providers and are accessible through a subscription. The callout box in top of Figure 4.1 shows the architecture of a typical public cloud. Many public clouds are available, including Google App Engine (GAE), Amazon Web Services (AWS), Microsoft Azure, IBM Blue Cloud, and Salesforce.com's Force.com. The providers of the aforementioned clouds are commercial providers that offer a publicly accessible remote interface for creating and managing VM instances within their proprietary infrastructure. A public cloud delivers a selected set of business processes.

Private Clouds:

A private cloud is built within the domain of an intranet owned by a single organization. Therefore, it is client owned and managed, and its access is limited to the own clients and their partners. Private clouds give local users a flexible and alert private infrastructure to run service workloads within their administrative domains. A private cloud is supposed to deliver more efficient and convenient cloud services. It may impact the cloud standardization, while retaining greater customization and organizational control.

Hybrid Clouds:

A hybrid cloud is built with both public and private clouds, as shown at the lower-left corner of Figure 4.1. Private clouds can also support a hybrid cloud model by supplementing local infrastructure with computing

capacity from an external public cloud. A hybrid cloud provides access to clients, the partner network, and third parties. In summary, public clouds promote standardization, preserve capital investment, and offer application flexibility. Private clouds attempt to achieve customization and offer higher efficiency, resiliency, security, and privacy. Hybrid clouds operate in the middle, with many compromises in terms of resource sharing.

Data-Center Networking Structure:

The core of a cloud is the server cluster (or VM cluster). Cluster nodes are used as compute nodes. A few control nodes are used to manage and monitor cloud activities. The scheduling of user jobs requires that you assign work to virtual clusters created for users. The gateway nodes provide the access points of the service from the outside world. These gateway nodes can be also used for security control of the entire cloud platform. In physical clusters and traditional grids, users expect static demand of resources. Clouds are designed to handle fluctuating workloads, and thus demand variable resources dynamically. Private clouds will satisfy this demand if properly designed and managed. Data centers and supercomputers have some similarities as well as fundamental differences. In the case of data centers, scaling is a fundamental require-ment. Data-center server clusters are typically built with large number of servers, ranging from thousands to millions of servers (nodes). Data centers and supercomputers also differ in networking requirements, as illustrated in Figure 4.2 which shows a multilayer structure for accessing the Internet. The server racks are at the bottom Layer 2, and they are connected through fast switches (S) as the hardware core. The data center is connected to the Internet at Layer 3 with many access routers (ARs) and border routers (BRs). These cloud models demand different levels of performance, data protection, and security enforcement. In this case, different SLAs may be applied to satisfy both providers and paid users. Cloud computing exploits many existing technologies.

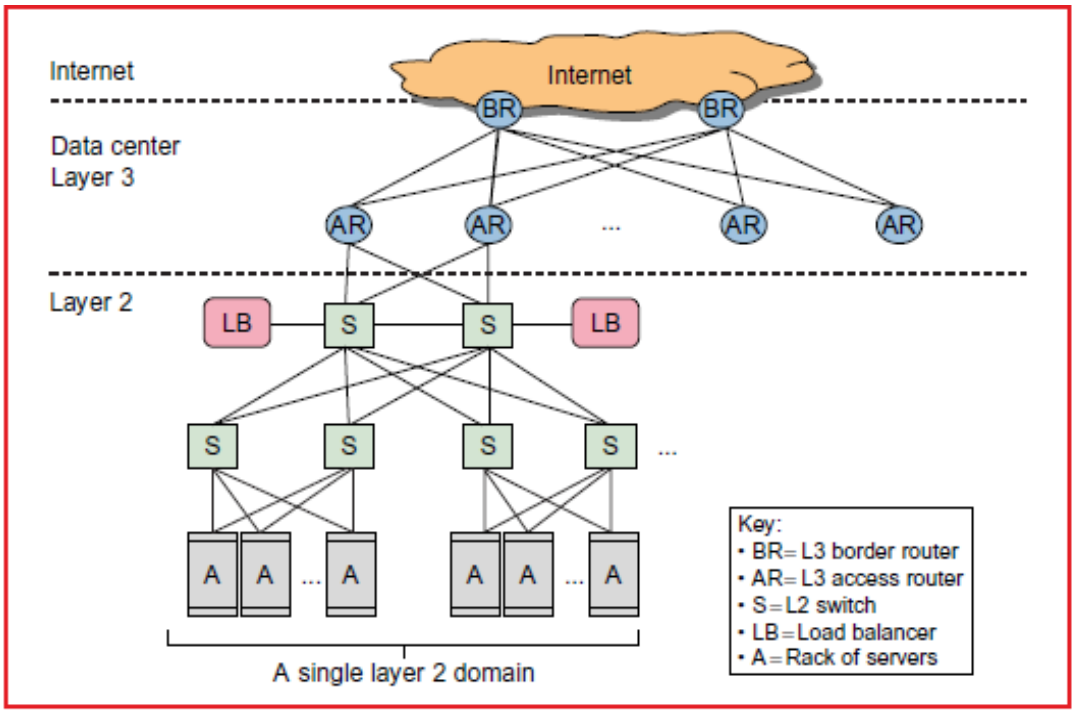


FIGURE 4.2

Standard data-center networking for the cloud to access the Internet.

Cloud Development Trends:

Although most clouds built in 2010 are large public clouds, the authors believe private clouds will grow much faster than public clouds in the future. Private clouds are easier to secure and more trustworthy within a company or organization. Once private clouds become mature and better secured, they could be open or converted to public clouds. Therefore, the boundary between public and private clouds could be blurred in the future. Most likely, most future clouds will be hybrid in nature.

2. Cloud Ecosystem and Enabling Technologies:

Cloud computing platforms differ from conventional computing platforms in many aspects. The traditional computing model is specified below by the process on the left, which involves buying the hardware, acquiring the necessary system software, installing the system, testing the configuration and executing the application code and management of resources. The cloud computing paradigm is shown on the right. This computing model follows a pay-as-you-go model. Therefore the cost is significantly reduced, because we simply rent computer resources without buying the computer in advance. All hardware and software resources are leased from the cloud provider without capital investment on the part of the users. Only the execution phase costs some money.

Classical Computing	Cloud Computing
(Repeat the following cycle every 18 months)	(Pay as you go per each service provided)
Buy and own	**Subscribe**
Hardware, system software, applications to meet peak needs	- - - -
Install, configure, test, verify, evaluate, manage	**Use** (Save about 80-95% of the total cost)
- - - -	- - - -
Use	(Finally)
- - - -	**$ - Pay for what you use**
Pay $$$$$ (High cost)	based on the QoS

Cloud Design Objectives:

The following list highlights six design objectives for cloud computing:

• Shifting computing from desktops to data centers Computer processing, storage, and software delivery is shifted away from desktops and local servers and toward data centers over the Internet.

• Service provisioning and cloud economics Providers supply cloud services by signing SLAs with consumers and end users. The services must be efficient in terms of computing, storage, and power consumption. Pricing is based on a pay-as-you-go policy.

• Scalability in performance The cloud platforms and software and infrastructure services must be able to scale in performance as the number of users increases.

• Data privacy protection Can you trust data centers to handle your private data and records? This concern

must be addressed to make clouds successful as trusted services.

• High quality of cloud services The QoS of cloud computing must be standardized to make clouds interoperable among multiple providers.

• New standards and interfaces This refers to solving the data lock-in problem associated with data centers or cloud providers. Universally accepted APIs and access protocols are needed to provide high portability and flexibility of virtualized applications.

Cost Model:

In traditional IT computing, users must acquire their own computer and peripheral equipment as capital expenses. In addition, they have to face operational expenditures in operating and maintaining the computer systems, including personnel and service costs.

Figure 4.3(a) shows the addition of variable operational costs on top of fixed capital investments in traditional IT. Note that the fixed cost is the main cost, and that it could be reduced slightly as the number of users increases. However, the operational costs may increase sharply with a larger number of users. Therefore, the total cost escalates quickly with massive numbers of users. On the other hand, cloud computing applies a pay-per-use business model, in which user jobs are outsourced to data centers. To use the cloud, one has no up-front cost in hardware acquisitions. Only variable costs are experienced by cloud users, as demonstrated in Figure 4.3(b).

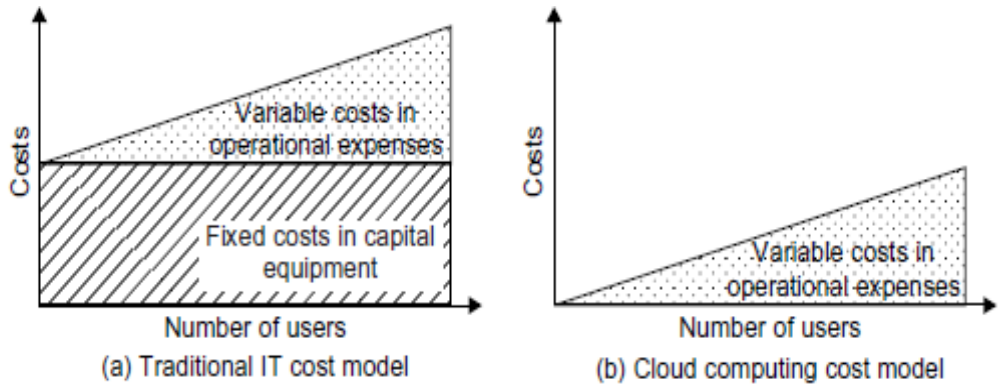


FIGURE 4.3

Computing economics between traditional IT users and cloud users.

Cloud Ecosystems:

With the emergence of various Internet clouds, an ecosystem of providers, users, and technologies has appeared. This ecosystem has evolved around public clouds. Strong interest is growing in open source cloud computing tools that let organizations build their own IaaS clouds using their internal infrastructures. Private and hybrid clouds are not exclusive, since public clouds are involved in both cloud types. A private/hybrid cloud allows remote access to its resources over the Internet using remote web service interfaces such as that

used in Amazon EC2.

They suggested four levels of ecosystem development in a private cloud. At the user end, consumers demand a flexible platform. At the cloud management level, the cloud manager provides virtualized resources over an IaaS platform. At the virtual infrastructure (VI) management level, the manager allocates VMs over multiple server clusters. Finally, at the VM management level, the VM managers handle VMs installed on individual host machines. An ecosystem of cloud tools attempts to span both cloud management and VI management. Integrating these two layers is complicated by the lack of open and standard interfaces between them.

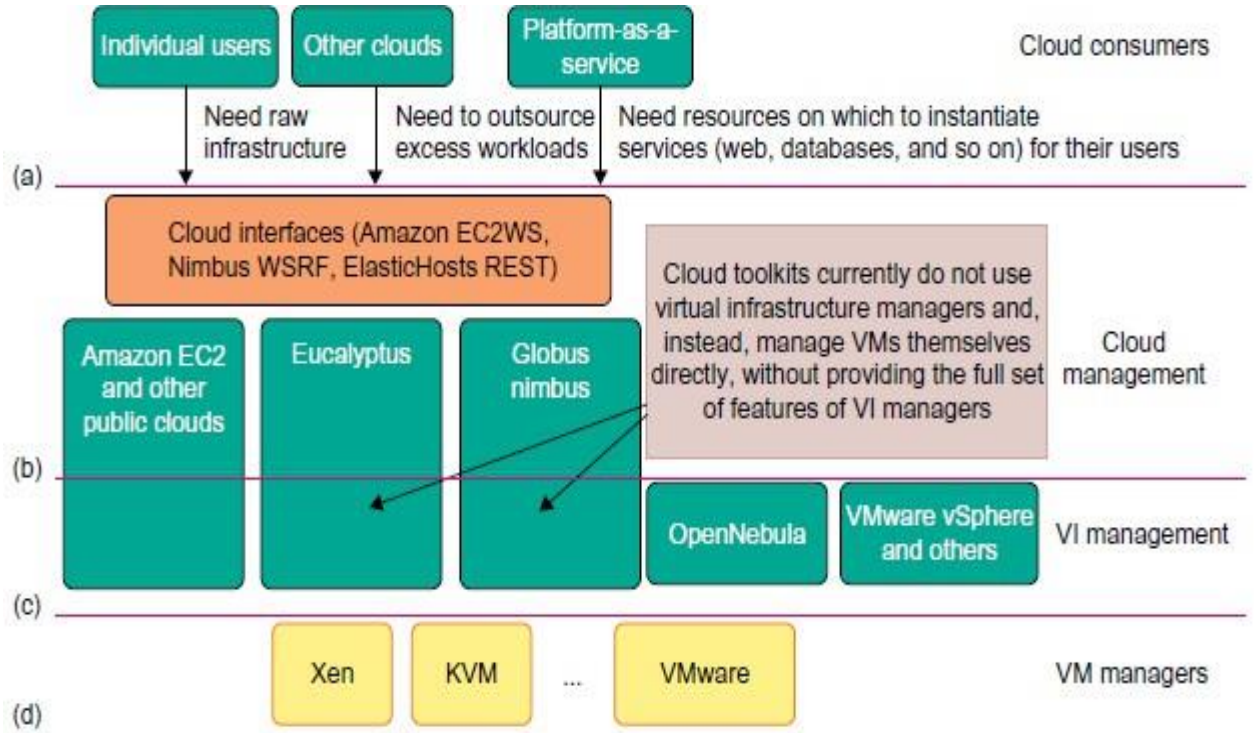


FIGURE 4.4

Cloud ecosystem for building private clouds: (a) Consumers demand a flexible platform; (b) Cloud manager provides virtualized resources over an IaaS platform; (c) VI manager allocates VMs; (d) VM managers handle VMs installed on servers.

An increasing number of startup companies are now basing their IT strategies on cloud resources, spending little or no capital to manage their own IT infrastructures. We desire a flexible and open architecture that enables organizations to build private/hybrid clouds. VI management is aimed at this goal.

3.Infrastructure-as-a-Service (IaaS):

Cloud computing delivers infrastructure, platform, and software (application) as services, which are made available as subscription-based services in a pay-as-you-go model to consumers. The services provided over the cloud can be generally categorized into three different service models: namely IaaS, Platform as a Service (PaaS), and Software as a Service (SaaS). These form the three pillars on top of which cloud computing solutions are delivered to end users. All three models allow users to access services over the Internet, relying entirely on the infrastructures of cloud service providers. These models are offered based on various SLAs

between providers and users. In a broad sense, the SLA for cloud computing is addressed in terms of service availability, performance, and data protection and security.

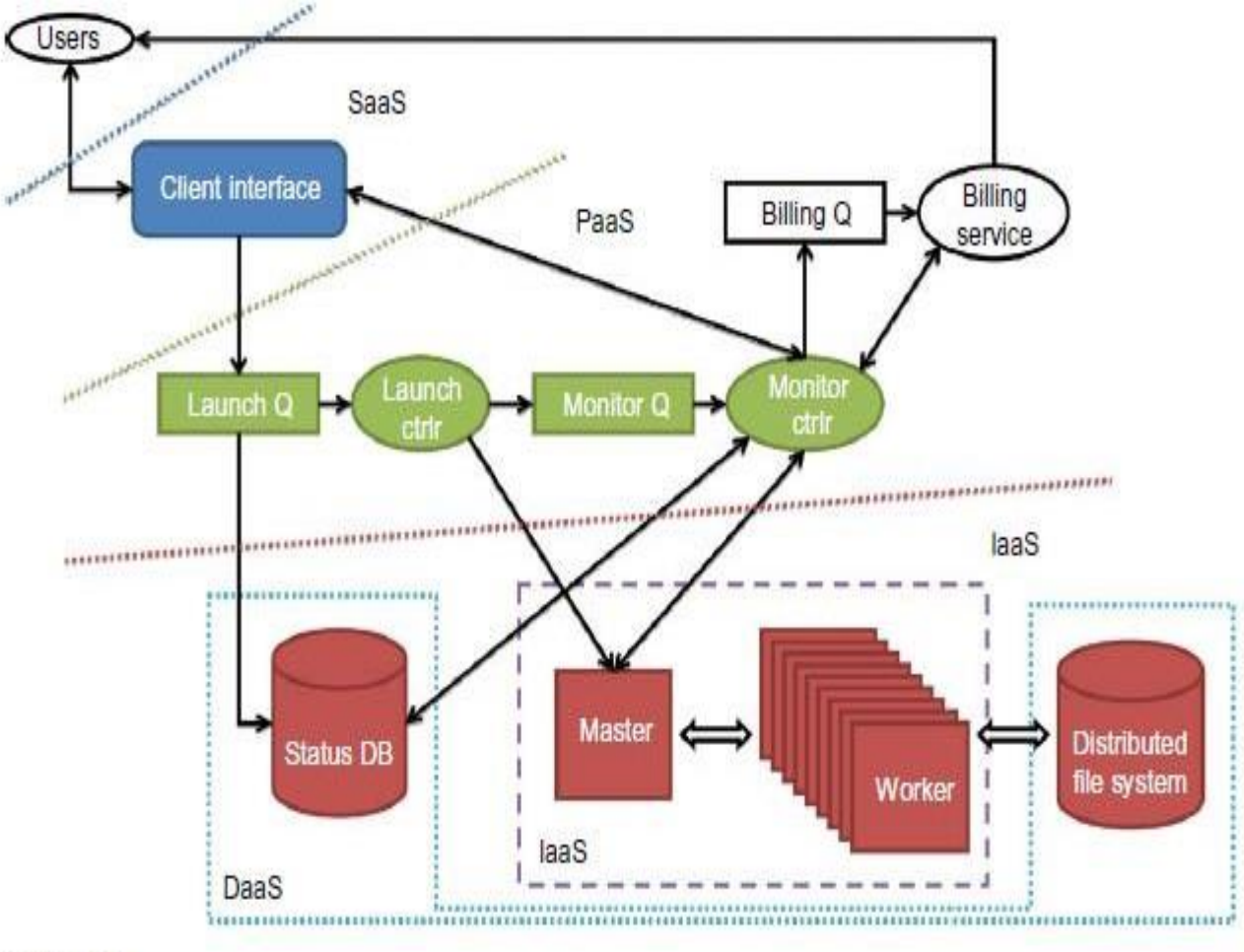


FIGURE 4.5

The IaaS, PaaS, and SaaS cloud service models at different service levels..

Figure 4.5 illustrates three cloud models at different service levels of the cloud. SaaS is applied at the application end using special interfaces by users or clients. At the PaaS layer, the cloud platform must perform billing services and handle job queuing, launching, and monitoring services. At the bottom layer of the IaaS services, databases, compute instances, the file system, and storage must be provisioned to satisfy user demands.

Infrastructure as a Service:

This model allows users to use virtualized IT resources for computing, storage, and networking. In short, the service is performed by rented cloud infrastructure. The user can deploy and run his applications over his chosen OS environment. The user does not manage or control the underlying cloud infrastructure, but has control over the OS, storage, deployed applications, and possibly select networking components. This IaaS model encompasses storage as a service, compute instances as a service, and communication as a service.

3. Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS):

In this section, we will introduce the PaaS and SaaS models for cloud computing. SaaS is often built on top of the PaaS, which is in turn built on top of the IaaS.

Platform as a Service (PaaS):

To be able to develop, deploy, and manage the execution of applications using provisioned resources demands a cloud platform with the proper software environment. Such a platform includes operating system and runtime library support. This has triggered the creation of the PaaS model to enable users to develop and deploy their user applications.

Table 4.2 highlights cloud platform services offered by five PaaS services.

Table 4.2 Five Public Cloud Offerings of PaaS [10,18]

Cloud Name	Languages and Developer Tools	Programming Models Supported by Provider	Target Applications and Storage Option
Google App Engine	Python, Java, and Eclipse-based IDE	MapReduce, web programming on demand	Web applications and BigTable storage
Salesforce.com's Force.com	Apex, Eclipse-based IDE, web-based Wizard	Workflow, Excel-like formula, Web programming on demand	Business applications such as CRM
Microsoft Azure	.NET, Azure tools for MS Visual Studio	Unrestricted model	Enterprise and web applications
Amazon Elastic MapReduce	Hive, Pig, Cascading, Java, Ruby, Perl, Python, PHP, R, C++	MapReduce	Data processing and e-commerce
Aneka	.NET, stand-alone SDK	Threads, task, MapReduce	.NET enterprise applications, HPC

The platform cloud is an integrated computer system consisting of both hardware and software infra-structure. The user application can be developed on this virtualized cloud platform using some programming languages and software tools supported by the provider (e.g., Java, Python, .NET). The user does not manage the underlying cloud infrastructure. The cloud provider supports user application development and testing on a well-defined service platform. This PaaS model enables a collaborated software development platform for users from different parts of the world. This model also encourages third parties to provide software management, integration, and service monitoring solutions.

Software as a Service (SaaS):

This refers to browser-initiated application software over thousands of cloud customers. Services and tools offered by PaaS are utilized in construction of applications and management of their deployment on resources offered by IaaS providers. The SaaS model provides software applications as a service. As a result, on the customer side, there is no upfront investment in servers or software licensing. On the provider side, costs are kept rather low, compared with conventional hosting of user applications. Customer data is stored in the cloud that is either vendor proprietary or publicly hosted to support PaaS and IaaS.

Mashup of Cloud Services:

- At the time of this writing, public clouds are in use by a growing number of users. Due to the lack of trust in leaking sensitive data in the business world, more and more enterprises, organizations, and communities are developing private clouds that demand deep customization. An enterprise cloud is used by multiple users within an organization. Each user may build some strategic applications on the cloud, and demands customized partitioning of the data, logic, and database in the metadata representation. Cloud mashups have resulted from the need to use multiple clouds simultaneously or in sequence. Some public repository provides thousands of service APIs and mashups for web commerce services. Popular APIs are provided by Google Maps, Twitter, YouTube, Amazon e-Commerce, Salesforce.com, etc.

4.PUBLIC CLOUD PLATFORMS: GAE, AWS, AND AZURE:

4. Public Clouds and Service Offerings:

5. Cloud services are demanded by computing and IT administrators, software vendors, and end users. Figure 4.19 introduce five levels of cloud players. At the top level, individual users and organizational users demand very different services. The application providers at the SaaS level serve mainly individual users. Most business organizations are serviced by IaaS and PaaS providers. The infrastructure services (IaaS) provide compute, storage, and communication resources to both applications and organizational users. The cloud environment is defined by the PaaS or platform providers. Note that the platform providers support both infrastructure services and organizational users directly.

Cloud services rely on new advances in machine virtualization, SOA, grid infrastructure management, and power efficiency. Consumers purchase such services in the form of IaaS, PaaS, or SaaS as described earlier. Also, many cloud entrepreneurs are selling value-added utility services to massive numbers of users. The cloud industry leverages the growing demand by many enterprises and business users to outsource their computing and storage jobs to professional providers. The provider service charges are often much lower than the cost for users to replace their obsolete servers frequently.

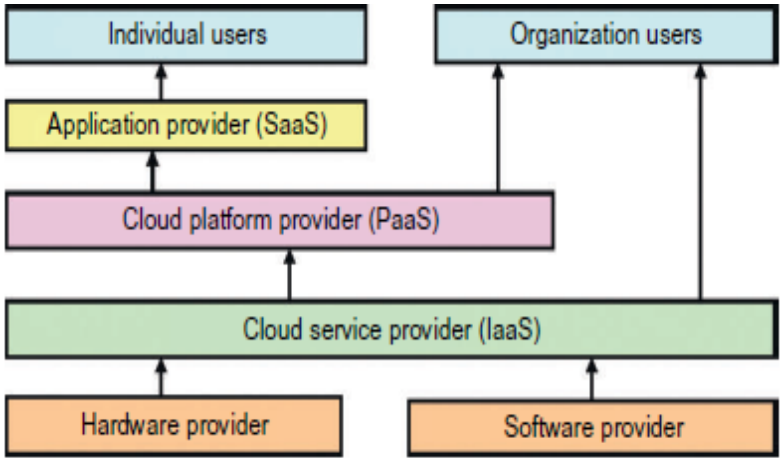


FIGURE 4.19

Roles of individual and organizational users and their interaction with cloud providers under various cloud service models.

As Table 4.5 shows, all IaaS, PaaS, and SaaS models allow users to access services over the Internet, relying entirely on the infrastructures of the cloud service providers. These models are offered based on various SLAs between the providers and the users.

Table 4.5 Five Major Cloud Platforms and Their Service Offerings [36]

Model	IBM	Amazon	Google	Microsoft	Salesforce
PaaS	BlueCloud, WCA, RC2		App Engine (GAE)	Windows Azure	Force.com
IaaS	Ensembles	AWS		Windows Azure	
SaaS	Lotus Live		Gmail, Docs	.NET service, Dynamic CRM	Online CRM, Gifttag
Virtualization		OS and Xen	Application Container	OS level/ Hypel-V	
Service Offerings	SOA, B2, TSAM, RAD, Web 2.0	EC2, S3, SQS, SimpleDB	GFS, Chubby, BigTable, MapReduce	Live, SQL Hotmail	Apex, visual force, record security
Security Features	WebSphere2 and PowerVM tuned for protection	PKI, VPN, EBS to recover from failure	Chubby locks for security enforcement	Replicated data, rule-based access control	Admin./record security, uses metadata API
User Interfaces		EC2 command-line tools	Web-based admin. console	Windows Azure portal	
Web API	Yes	Yes	Yes	Yes	Yes
Programming Support	AMI		Python	.NET Framework	

Note: *WCA: WebSphere CloudBurst Appliance; RC2: Research Compute Cloud; RAD: Rational Application Developer; SOA: Service-Oriented Architecture; TSAM: Tivoli Service Automation Manager; EC2: Elastic Compute Cloud; S3: Simple Storage Service; SQS: Simple Queue Service; GAE: Google App Engine; AWS: Amazon Web Services; SQL: Structured Query Language; EBS: Elastic Block Store; CRM: Consumer Relationship Management.*

6. Google App Engine (GAE):

Google has the world's largest search engine facilities. The company has extensive experience in massive data processing that has led to new insights into data-center design and novel programming models that scale to incredible sizes. The Google platform is based on its search engine expertise with MapReduce, this which is applicable to many other areas.

Google Cloud Infrastructure:

Google has pioneered cloud development by leveraging the large number of data centers it operates. For example, Google pioneered cloud services in Gmail, Google Docs, and Google Earth, among other applications. These applications can support a large number of users simultaneously with HA. Notable technology achievements include the Google File System (GFS), MapReduce, BigTable, and Chubby. In 2008, Google announced the GAE web application platform which is becoming a common platform for many small cloud service providers. This platform specializes in supporting scalable web applications. GAE enables users to run their applications on a large number of data centers associated with Google's search engine operations.

GAE Architecture:

Figure 4.20 shows the major building blocks of the **Google cloud platform** which has been used to deliver the cloud services highlighted earlier. GFS is used for storing large amounts of data. MapReduce is for use in application program development. Chubby is used for distributed application lock services. BigTable offers a storage service for accessing structured data.

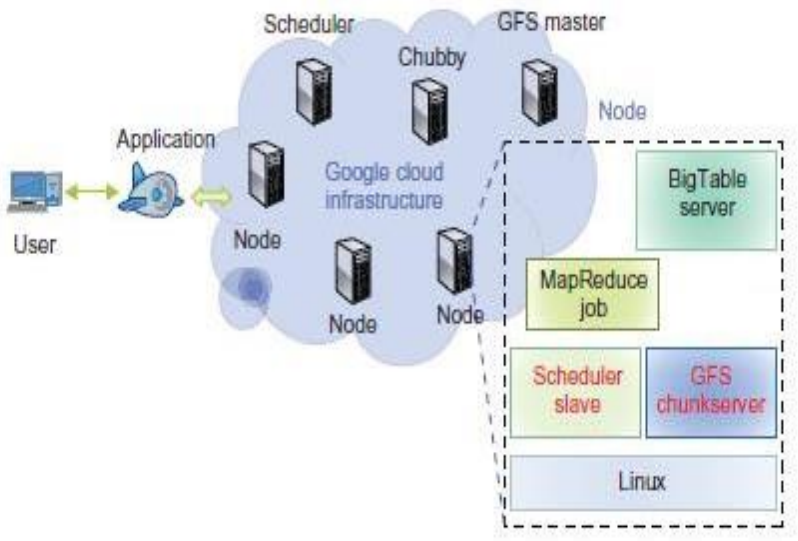


FIGURE 4.20

Google cloud platform and major building blocks, the blocks shown are large clusters of low-cost servers.

Users can interact with Google applications via the web interface provided by each application. Third-party application providers can use GAE to build cloud applications for providing services. The applications all run in data centers under tight management by Google engineers. Inside each data center, there are thousands of servers forming different clusters. The above figure shows the overall architecture of the Google cloud infrastructure. A typical cluster configuration can run the Google File System, MapReduce jobs, and BigTable servers for structure data. Extra services such as Chubby for distributed locks can also run in the clusters. GAE runs the user program on Google's infrastructure. As it is a platform running third-party programs, application developers now do not need to worry about the maintenance of servers. GAE can be thought of as the combination of several software components. The frontend is an application framework which is similar to other web application frameworks such as ASP, J2EE, and JSP. At the time of this writing, GAE supports Python and Java programming environments. The applications can run similar to web application containers. The frontend can be used as the dynamic web serving infrastructure which can provide the full support of common technologies.

Functional Modules of GAE:

The GAE platform comprises the following five major components. The GAE is not an infrastructure platform, but rather an application development platform for users.

a. The datastore offers object-oriented, distributed, structured data storage services based on BigTable techniques. The datastore secures data management operations.

b. The application runtime environment offers a platform for scalable web programming and execution. It supports two development languages: Python and Java.

c. The software development kit (SDK) is used for local application development. The SDK allows users to execute test runs of local applications and upload application code.

d. The administration console is used for easy management of user application development cycles, instead of for physical resource management.

e. The GAE web service infrastructure provides special interfaces to guarantee flexible use and management of storage and network resources by GAE.

GAE Applications:

Well-known GAE applications include the Google Search Engine, Google Docs, Google Earth, and Gmail. These applications can support large numbers of users simultaneously. Users can interact with Google applications via the web interface provided by each application. Third-party application providers can use GAE to build cloud applications for providing services. The applications are all run in the Google data centers. Inside each data center, there might be thousands of server nodes to form different clusters.

7. Amazon Web Services (AWS):

VMs can be used to share computing resources both flexibly and safely. Amazon has been a leader in providing public cloud services. Amazon applies the IaaS model in providing its services. Figure 4.21 shows the AWS architecture. EC2 provides the virtualized platforms to the host VMs where the cloud application can run. S3 (Simple Storage Service) provides the object-oriented storage service for users.

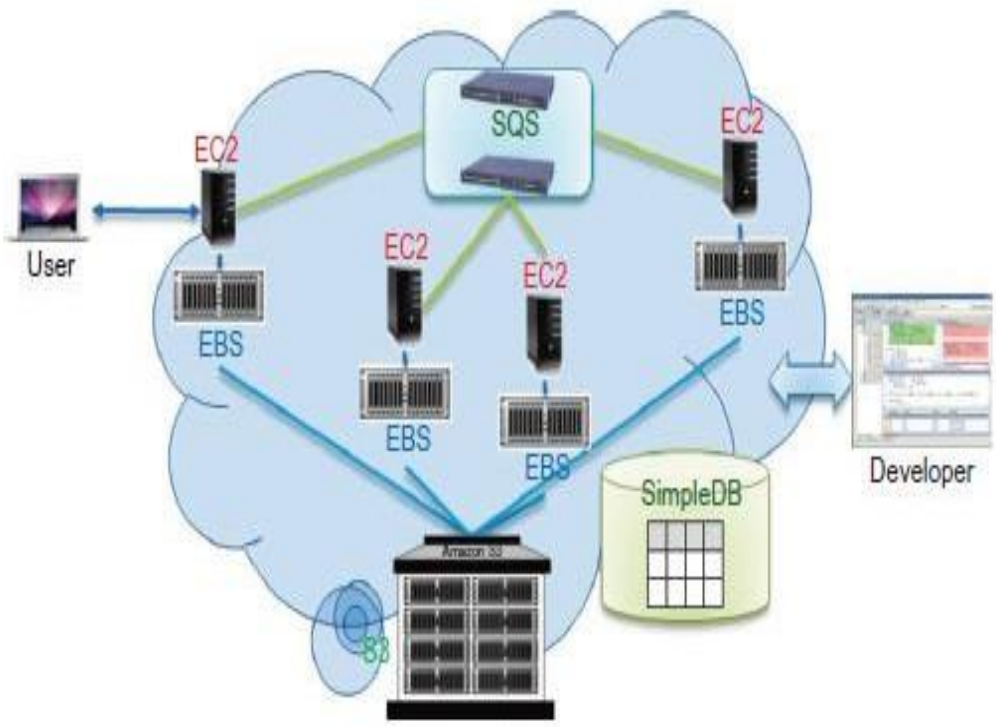


FIGURE 4.21

Amazon cloud computing infrastructure (Key services are identified here; many more are listed in Table 4.6).

Table 4.6 AWS Offerings in 2011

Service Area	Service Modules and Abbreviated Names
Compute	Elastic Compute Cloud (EC2), Elastic MapReduce, Auto Scaling
Messaging	Simple Queue Service (SQS), Simple Notification Service (SNS)
Storage	Simple Storage Service (S3), Elastic Block Storage (EBS), AWS Import/Export
Content Delivery	Amazon CloudFront
Monitoring	Amazon CloudWatch
Support	AWS Premium Support
Database	Amazon SimpleDB, Relational Database Service (RDS)
Networking	Virtual Private Cloud (VPC) (Example 4.1, Figure 4.6), Elastic Load Balancing
Web Traffic	Alexa Web Information Service, Alexa Web Sites
E-Commerce	Fulfillment Web Service (FWS)
Payments and Billing	Flexible Payments Service (FPS), Amazon DevPay
Workforce	Amazon Mechanical Turk

(Courtesy of Amazon, http://aws.amazon.com [3])

EBS (Elastic Block Service) provides the block storage interface which can be used to support traditional applications. SQS stands for Simple Queue Service, and its job is to ensure a reliable message service between two processes. The message can be kept reliably even when the receiver processes are not running. Users can access their objects through SOAP with either browsers or other client programs which support the SOAP standard. Table 4.6 summarizes the service offerings by AWS in 12 application tracks.

8. Microsoft Windows Azure:

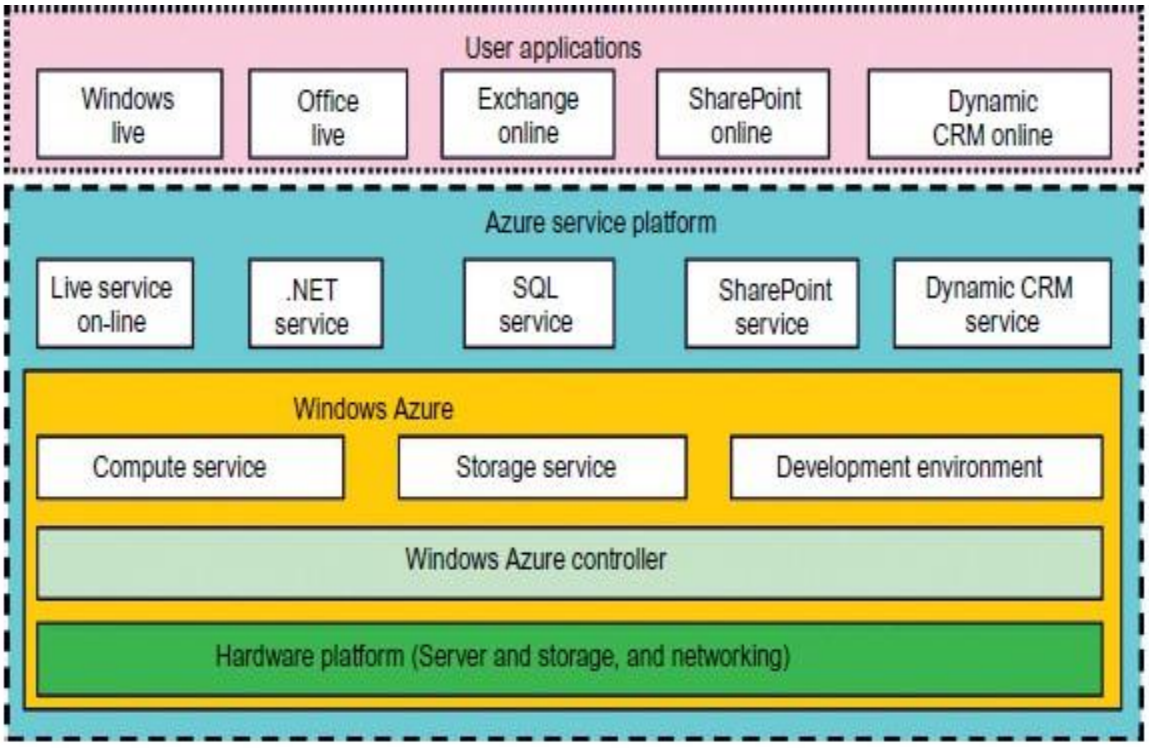


FIGURE 4.22

Microsoft Windows Azure platform for cloud computing.

In 2008, Microsoft launched a Windows Azure platform to meet the challenges in cloud computing. This platform is built over Microsoft data centers. Figure 4.22 shows the overall architecture of Microsoft's cloud platform. The platform is divided into three major component platforms. Windows Azure offers a cloud platform built on Windows OS and based on Microsoft virtualization technology. Applications are installed on VMs deployed on the data-center servers. Azure manages all servers, storage, and network resources of the data center. On top of the infrastructure are the various services for building different cloud applications. Cloud-level services provided by the Azure platform are introduced below.

- Live service Users can visit Microsoft Live applications and apply the data involved across multiple machines concurrently.

- .NET service This package supports application development on local hosts and execution on cloud machines associated with the SQL server in the cloud.

- SharePoint service This provides a scalable and manageable platform for users to develop their special business applications in upgraded web services.

- Dynamic CRM service This provides software developers a business platform in managing CRM applications in financing, marketing, and sales and promotions.

All these cloud services in Azure can interact with traditional Microsoft software applications, such as Windows Live, Office Live, Exchange online, SharePoint online, and dynamic CRM online. The Azure platform applies the standard web communication protocols SOAP and REST. The Azure service applications allow users to integrate the cloud application with other platforms or third-party clouds.

2. SERVICES AND SERVICE-ORIENTED ARCHITECTURE:

In general, SOA is about how to design a software system that makes use of services of new or legacy applications through their published or discoverable interfaces. These applications are often distributed over the networks. SOA also aims to make service interoperability extensible and effective. It prompts architecture styles such as loose coupling, published interfaces, and a standard communication model in order to support this goal. The World Wide Web Consortium (W3C) defines SOA as a form of distributed systems architecture characterized by the following properties:

Logical view: The SOA is an abstracted, logical view of actual programs, databases, business processes, and so on, defined in terms of what it does, typically carrying out a business-level operation. The service is formally defined in terms of the messages exchanged between provider agents and requester agents.

Message orientation: The internal structure of providers and requesters include the implementation language, process structure, and even database structure. These features are deliberately abstracted away in the SOA: Using the SOA discipline one does not and should not need to know how an agent implementing a service is constructed. A key benefit of this concerns legacy systems. By avoiding any knowledge of the internal structure of an agent, one can incorporate any software component or application to adhere to the formal service definition.

Description orientation: A service is described by machine-executable metadata. The description supports the public nature of the SOA: Only those details that are exposed to the public and are important for the use of the service should be included in the description. The semantics of a service should be documented, either directly or indirectly, by its description.

• Granularity Services tend to use a small number of operations with relatively large and complex messages.

• Network orientation Services tend to be oriented toward use over a network, though this is not an absolute requirement.

• Platform-neutral Messages are sent in a platform-neutral, standardized format delivered through the interfaces. XML is the most obvious format that meets this constraint.

SOA is related to early efforts on the architecture style of large-scale distributed systems, particularly Representational State Transfer (REST). Nowadays, REST still provides an alternative to the complex standard-driven web services technology and is used in many Web 2.0 services.

1. REST and Systems of Systems: (*****)

REST is a software architecture style for distributed systems, particularly distributed hypermedia systems, such as the World Wide Web. It has recently gained popularity among enterprises such as Google, Amazon, Yahoo!, and especially social networks such as Facebook and Twitter because of its simplicity, and its ease of being published and consumed by clients. REST is shown in Figure 5.1. The REST architectural style is based on four principles: Resource Identification through URIs: The RESTful web service exposes a set of resources which identify targets of interaction with its clients.

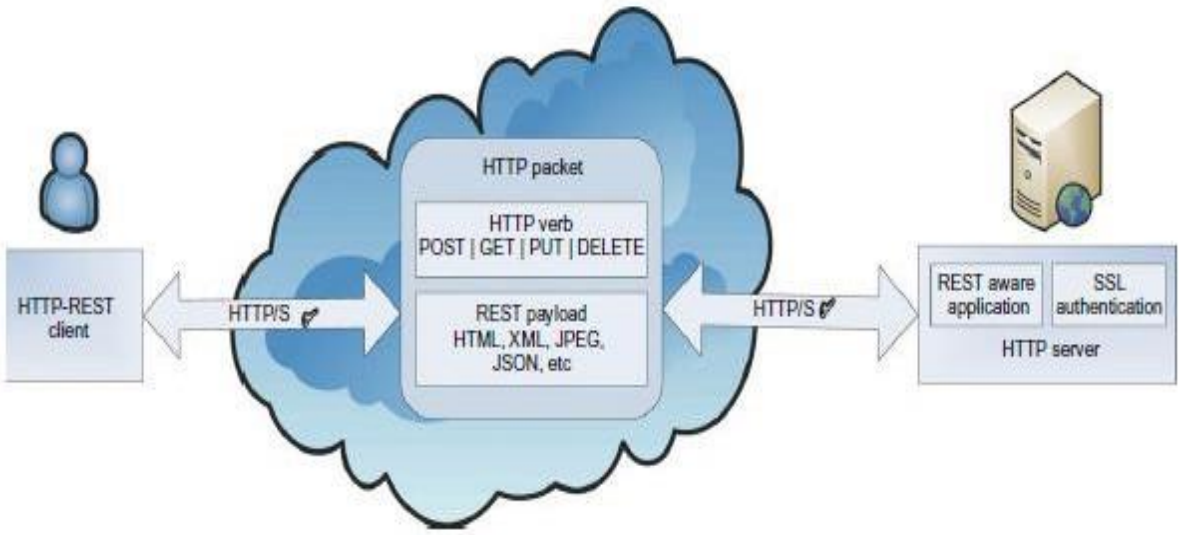


FIGURE 5.1

A simple REST interaction between user and server in HTTP specification.

The key abstraction of information in REST is a resource. Any information that can be named can be a resource, such as a document or image or a temporal service. A resource is a conceptual mapping to a set of

entities. Each particular resource is identified by a unique name, or more precisely, a Uniform Resource Identifier (URI) which is of type URL, providing a global addressing space for resources involved in an interaction between components as well as facilitating service discovery. The URIs can be bookmarked or exchanged via hyperlinks, providing more readability and the potential for advertisement.

Uniform, Constrained Interface: Interaction with RESTful web services is done via the HTTP standard, client/server cacheable protocol. Resources are manipulated using a fixed set of four CRUD (create, read, update, delete) verbs or operations: PUT, GET, POST, and DELETE. PUT creates a new resource, which can then be destroyed by using DELETE. GET retrieves the current state of a resource. POST transfers a new state onto a resource.

Self-Descriptive Message: A REST message includes enough information to describe how to process the message. This enables intermediaries to do more with the message without parsing the message contents. In REST, resources are decoupled from their representation so that their content can be accessed in a variety of standard formats (e.g., HTML, XML, MIME, plain text, PDF, JPEG, JSON, etc.). REST provides multiple/alternate representations of each resource. Metadata about the resource is available and can be used for various purposes, such as cache control, transmission error detection, authentication or authorization, and access control.

Stateless Interactions: The REST interactions are <stateless= in the sense that the meaning of a message does not depend on the state of the conversation. Stateless communications improve visibility, since a monitoring system does not have to look beyond a single request data field in order to determine the full nature of the request reliability as it facilitates the task of recovering from partial failures, and increases scalability as discarding state between requests allows the server component to quickly free resources. However, stateless interactions may decrease network performance by increasing the repetitive data (per-interaction overhead). Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, such as URI rewriting, cookies, and hidden form fields. State can be embedded in response messages to point to valid future states of the interaction. Such lightweight infrastructure, where services can be built with minimal development tools, is inexpensive and easy to adopt. The effort required to build a client to interact with a RESTful service is rather small as developers can begin testing such services from an ordinary web browser, without having to develop custom client-side software. From an operational point of view, a stateless RESTful web service is scalable to serve a very large number of clients, as a result of REST support for caching, clustering, and load balancing.

RESTful web services can be considered an alternative to SOAP stack or <big web services,= described in the next section, because of their simplicity, lightweight nature, and integration with HTTP. With the help of URIs and hyperlinks, REST has shown that it is possible to discover web resources without an approach based on registration to a centralized repository. Recently, the web Application Description Language (WADL) [3] has been proposed as an XML vocabulary to describe RESTful web services, enabling them to be discovered and accessed immediately by potential clients. However, there are not a variety of toolkits for developing RESTful applications. Also, restrictions on GET length, which does not allow encoding of more than 4 KB of data in the resource URI, can create problems because the server would reject such malformed URIs, or may even be subject to crashes. REST is not a standard. It is a design and architectural style for largescale distributed systems.

Table 5.1 REST Architectural Elements (Adapted from [2])

REST Elements	Elements	Example
Data elements	Resource	The intended conceptual target of a hypertext reference
	Resource identifier	URL
	Representation	HTML document, JPEG image, XML, etc.
	Representation metadata	Media type, last-modified time
	Resource metadata	Source link, alternates, vary
	Control data	If-modified-since, cache-control
Connectors	Client	libwww, libwww-perl
	Server	libwww, Apache API, NSAPI
	Cache	Browser cache, Akamai cache network
	Resolver	Bind (DNS lookup library)
	Tunnel	SSL after HTTP CONNECT
Components	Origin server	Apache httpd, Microsoft IIS
	Gateway	Squid, CGI, Reverse Proxy
	Proxy	CERN Proxy, Netscape Proxy, Gauntlet
	User agent	Netscape Navigator, Lynx, MOMspider

Table 5.1 lists the REST architectural elements. Several Java frameworks have emerged to help with building RESTful web services. Restlet, a lightweight framework, implements REST architectural elements such as resources, representation, connector, and media type for any kind of RESTful system, including web services. In the Restlet framework, both the client and the server are components. Components communicate with each other via connectors.

2. Services and Web Services:

In an SOA paradigm, software capabilities are delivered and consumed via loosely coupled, reusable, coarse-grained, discoverable, and self-contained services interacting via a message-based communication model. The web has becomes a medium for connecting remote clients with applications for years, and more recently, integrating applications across the Internet has gained in popularity. The term <web service= is often referred to a self-contained, self-describing, modular application designed to be used and accessible by other software

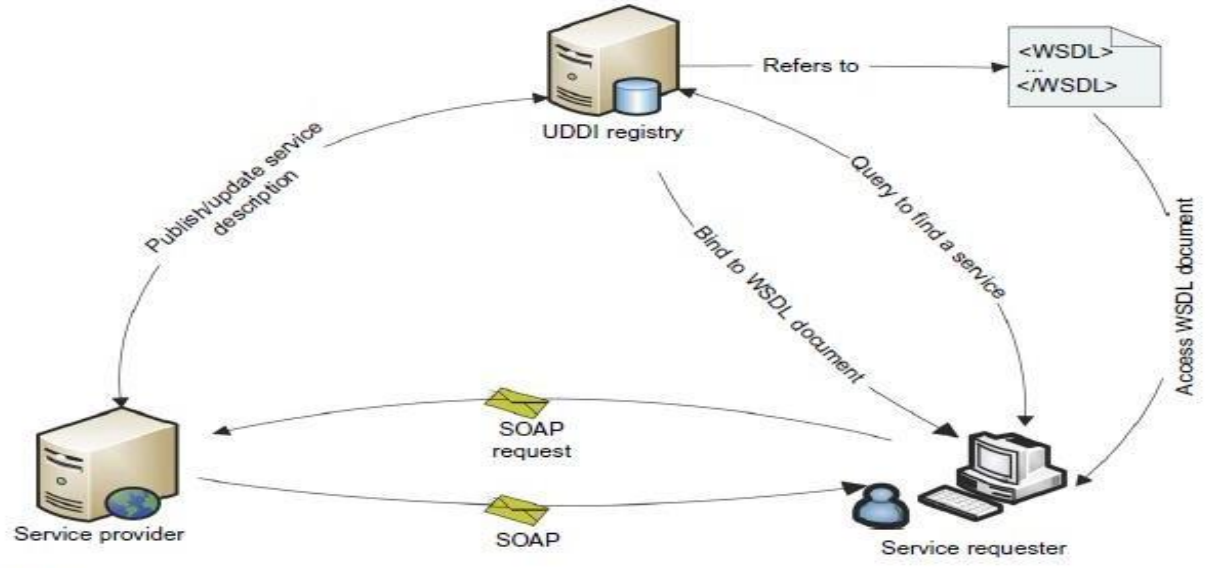


FIGURE 5.2
A simple web service interaction among provider, user, and the UDDI registry.

applications across the web. Once a web service is deployed, other applications and other web services can discover and invoke the deployed service (**Figure 5.2**).

In fact, a web service is one of the most common instances of an SOA implementation. The W3C working group [1] defines a web service as a software system designed to support interoperable machine-to-machine interaction over a network. According to this definition, a web service has an interface described in a machine-executable format (specifically Web Services Description Language or WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards. The technologies that make up the core of today's web services are as follows:

Simple Object Access Protocol (SOAP) SOAP provides a standard packaging structure for transmission of XML documents over various Internet protocols, such as SMTP, HTTP, and FTP. By having such a standard message format, heterogeneous middleware systems can achieve interoperability. A SOAP message consists of a root element called envelope, which contains a header: a container that can be extended by intermediaries with additional application-level elements such as routing information, authentication, transaction management, message parsing instructions, and Quality of Service (QoS) configurations, as well as a body element that carries the payload of the message. The content of the payload will be marshaled by the sender's SOAP engine and unmarshaled at the receiver side, based on the XML schema that describes the structure of the SOAP message.

Web Services Description Language (WSDL) WSDL describes the interface, a set of operations supported by a web service in a standard format. It standardizes the representation of input and output parameters of its operations as well as the service's protocol binding, the way in which the messages will be transferred on the wire. Using WSDL enables disparate clients to automatically understand how to interact with a web service.

Universal Description, Discovery, and Integration (UDDI) UDDI provides a global registry for advertising and discovery of web services, by searching for names, identifiers, categories, or the specification implemented by the web service.

SOAP is an extension, and an evolved version of XML-RPC, a simple and effective remote procedure call protocol which uses XML for encoding its calls and HTTP as a transport mechanism, introduced in 1999 [7]. According to its conventions, a procedure executed on the server and the value it returns was a formatted in XML. However, XML-RPC was not fully aligned with the latest XML standardization. Moreover, it did not allow developers to extend the request or response format of an XML-RPC call. As the XML schema became a W3C recommendation in 2001, SOAP mainly describes the protocols between interacting parties and leaves the data format of exchanging messages to XML schema to handle. The major difference between web service technology and other technologies such as J2EE, CORBA, and CGI scripting is its standardization, since it is based on standardized XML, providing a language-neutral representation of data. Most web services transmit messages over HTTP, making them available as Internet-scale applications. In addition, unlike CORBA and J2EE, using HTTP as the tunneling protocol by web services enables remote communication through firewalls and proxies. SOAP-based web services are also referred to as <big web services>. As we saw earlier in this chapter, RESTful services can also be considered a web service, in an HTTP context. SOAP-based web services interaction can be either synchronous or asynchronous, making them suitable for both request-response and one-way exchange patterns, thus increasing web service availability in case of failure.

WS-I Protocol Stack:

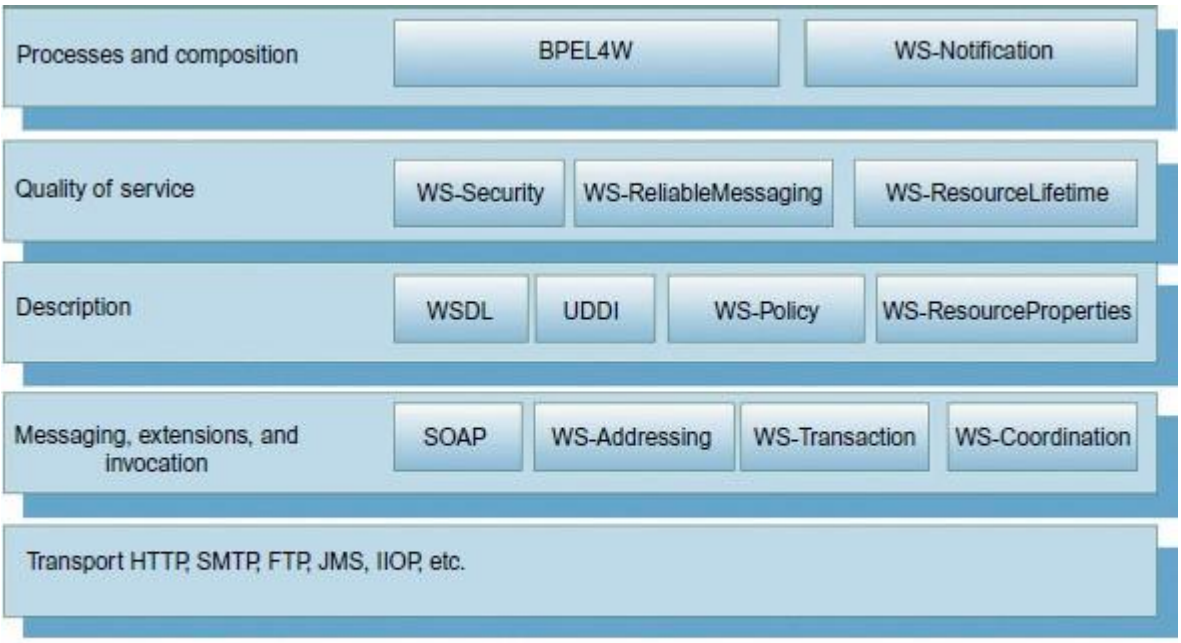


FIGURE 5.3

WS-I protocol stack and its related specifications.

Unlike RESTful web services that do not cover QoS and contractual properties, several optional specifications have been proposed for SOAP-based web services to define nonfunctional requirements and to guarantee a certain level of quality in message communication as well as reliable, transactional policies, such as WS-Security [9], WS-Agreement [10], WS-ReliableMessaging [11], WS-Transaction [12], and WS-Coordination [13] as shown in Figure 5.3.

As mentioned, SOAP messages are encoded using XML, which requires that all self-described data be sent as ASCII strings. The description takes the form of start and end tags which often constitute half or more of the message's bytes. Transmitting data using XML leads to a considerable transmission overhead, increasing the amount of transferred data by a factor 4 to 10 [14]. Moreover, XML processing is compute and memory intensive and grows with both the total size of the data and the number of data fields, making web services inappropriate for use by limited-profile devices, such as handheld PDAs and mobile phones.

Web services provide on-the-fly software composition, described further in Section 5.5, through the use of loosely coupled, reusable software components. By using Business Process Execution Language for Web Services (BPEL4WS), a standard executable language for specifying interactions between web services recommended by OASIS, web services can be composed together to make more complex web services and workflows. BPEL4WS is an XML-based language, built on top of web service specifications, which is used to define and manage long-lived service orchestrations or processes.

In BPEL, a business process is a large-grained stateful service, which executes steps to complete a business goal. That goal can be the completion of a business transaction, or fulfillment of the job of a service. The steps in the BPEL process execute activities (represented by BPEL language elements) to accomplish work. Those activities are centered on invoking partner services to perform tasks (their job) and return results back to the

process. BPEL enables organizations to automate their business processes by orchestrating services. Workflow in a grid context is defined [15] as <The automation of the processes, which involves the orchestration of a set of Grid services, agents and actors that must be combined together to solve a problem or to define a new service.

A SOAP message consists of an envelope used by the applications to enclose information that need to be sent. An envelope contains a header and a body block. The EncodingStyle element refers

to the URI address of an XML schema for encoding elements of the message. Each element of a SOAP message may have a different encoding, but unless specified, the encoding of the whole message is as defined in the XML schema of the root element. The header is an optional part of a SOAP message that may contain auxiliary information as mentioned earlier, which does not exist in this example.

The body of a SOAP request-response message contains the main information of the conversation, formatted in one or more XML blocks. In this example, the client is calling CreateBucket of the Amazon S3 web service interface. In case of an error in service invocation, a SOAP message including a Fault element in the body will be forwarded to the service client as a response, as an indicator of a protocol-level error.

WS-* Core SOAP Header Standards:

Table 5.4 summarizes some of the many (around 100) core SOAP header specifications. There are many categories and several overlapping standards in each category. Many are expanded in this chapter with XML, WSDL, SOAP, BPEL, WS-Security, UDDI, WSRF, and WSRP. The number and complexity of the WS-* standards have contributed to the growing trend of using REST and not web services. It was a brilliant idea to achieve interoperability through self-describing messages, but experience showed that it was too hard to build the required tooling with the required performance and short implementation time.

Table 5.4 The 10 Areas Covered by the Core WS-* Specifications

WS-* Specification Area	Examples
1. Core Service Model	XML, WSDL, SOAP
2. Service Internet	WS-Addressing, WS-MessageDelivery, Reliable WSRM, Efficient MOTM
3. Notification	WS-Notification, WS-Eventing (Publish-Subscribe)
4. Workflow and Transactions	BPEL, WS-Choreography, WS-Coordination
5. Security	WS-Security, WS-Trust, WS-Federation, SAML, WS-SecureConversation
6. Service Discovery	UDDI, WS-Discovery
7. System Metadata and State	WSRF, WS-MetadataExchange, WS-Context
8. Management	WSDM, WS-Management, WS-Transfer
9. Policy and Agreements	WS-Policy, WS-Agreement
10. Portals and User Interfaces	WSRP (Remote Portlets)

3.Enterprise Multitier Architecture:

Enterprise applications often use multitier architecture to encapsulate and integrate various functionalities. Multitier architecture is a kind of client/server architecture in which the presentation, the application processing, and the data management are logically separate processes. The simplest known

multilayer architecture is a two-tier or client/server system. This traditional two-tier, client/server model requires clustering and disaster recovery to ensure resiliency. While the use of fewer nodes in an enterprise simplifies manageability, change management is difficult as it requires servers to be taken offline for repair, upgrading, and new application deployments. Moreover, the deployment of new applications and enhancements is complex and time-consuming in fat-client environments, resulting in reduced availability. A three-tier information system consists of the following layers (Figure 5.4):

- Presentation layer Presents information to external entities and allows them to interact with the system by submitting operations and getting responses.

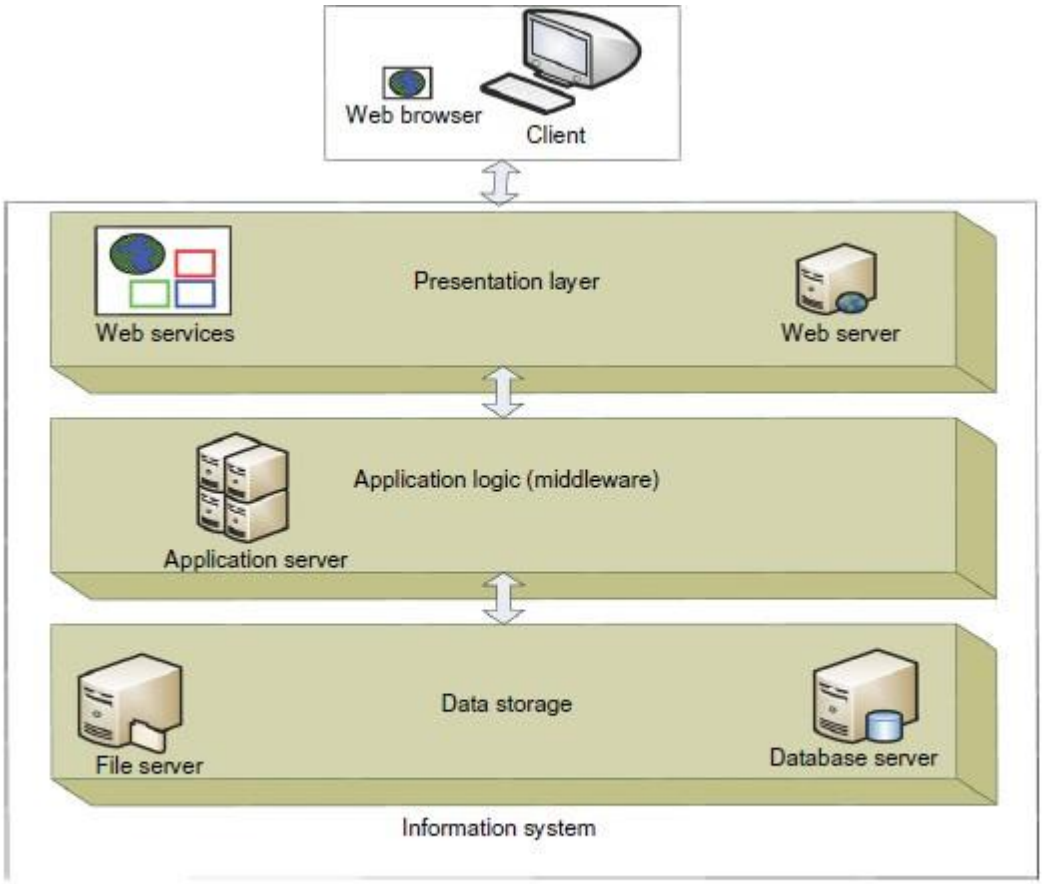


FIGURE 5.4
Three-tier system architecture.

- Business/application logic layer or middleware Programs that implement the actual operations requested by the client through the presentation layer. The middle tier can also control user authentication and access to resources, as well as performing some of the query processing for the client, thus removing some of the load from the database servers.

- Resource management layer Also known as the data layer, deals with and implements the different data sources of an information system.

In fact, a three-tier system is an extension of two-tier architecture where the application logic is separated from the resource management layer [21]. By the late 1990s, as the Internet became an important part of many applications, the industry extended the three-tier model to an N-tier approach. SOAP-based and RESTful web

services have become more integrated into applications. As a consequence, the data tier split into a data storage tier and a data access tier. In very sophisticated systems, an additional wrapper tier can be added to unify data access to both databases and web services. Web services can benefit from the separation of concerns inherent in multitier architecture in almost the same way as most dynamic web applications [22].

The business logic and data can be shared by both automated and GUI clients. The only differences are the nature of the client and the presentation layer of the middle tier. Moreover, separating business logic from data access enables database independence. N-tier architecture is characterized by the functional decomposition of applications, service components, and their distributed deployment. Such an architecture for both web services and dynamic web applications leads to reusability, simplicity, extensibility, and clear separation of component functionalities.

Web services can be seen as another tier on top of the middleware and application integration infrastructure [23], allowing systems to interact with a standard protocol across the Internet. Because each tier can be managed or scaled independently, flexibility is increased in the IT infrastructure that employs N-tier architecture. In the next section, we will describe OGSA, as multitier, service-oriented architecture for middleware which describes the capabilities of a grid computing environment and embodies web services to make computing resources accessible in large-scale heterogeneous environments.

3. Grid Services and OGSA:

The OGSA [24], developed within the OGSA Working Group of the Global Grid Forum (recently renamed to Open Grid Forum or OGF and being merged with the Enterprise Grid Alliance or EGA in June 2006), is a service-oriented architecture that aims to define a common, standard, and open architecture for grid-based applications. <Open= refers to both the process to develop standards and the standards themselves. In OGSA, everything from registries, to computational tasks, to data resources is considered a service. These extensible set of services are the building blocks of an OGSA-based grid. OGSA is intended to:

- Facilitate use and management of resources across distributed, heterogeneous environments
- Deliver seamless QoS
- Define open, published interfaces in order to provide interoperability of diverse resources
- Exploit industry-standard integration technologies
- Develop standards that achieve interoperability • Integrate, virtualize, and manage services and resources in a distributed, heterogeneous environment
- Deliver functionality as loosely coupled, interacting services aligned with industry-accepted web service standards

Based on OGSA, a grid is built from a small number of standards-based components, called grid services. [25] defines a grid service as <a web service that provides a set of well-defined interfaces, following specific conventions (expressed using WSDL).= OGSA gives a high-level architectural

view of grid services and doesn't go into much detail when describing grid services. It basically outlines what a grid service should have. A grid service implements one or more interfaces, where each interface defines a set of operations that are invoked by exchanging a defined sequence of messages, based on the Open Grid Services Infrastructure (OGSI) [26]. OGSI, also developed by the Global Grid Forum, gives a formal and technical specification of a grid service.

Grid service interfaces correspond to portTypes in WSDL. The set of portTypes supported by a grid service, along with some additional information relating to versioning, are specified in the grid service's serviceType, a WSDL extensibility element defined by OGSA. The interfaces address discovery, dynamic service creation, lifetime management, notification, and manageability; whereas the conventions address naming and upgradeability. Grid service implementations can target native platform facilities for integration with, and of, existing IT infrastructures.

According to [25], OGSA services fall into seven broad areas, defined in terms of capabilities frequently required in a grid scenario. Figure 5.5 shows the OGSA architecture. These services are summarized as follows:

- Infrastructure Services Refer to a set of common functionalities, such as naming, typically required by higher level services.

- Execution Management Services Concerned with issues such as starting and managing tasks, including placement, provisioning, and life-cycle management. Tasks may range from simple jobs to complex workflows or composite services.

- Data Management Services Provide functionality to move data to where it is needed, maintain replicated copies, run queries and updates, and transform data into new formats. These services must handle issues such as data consistency, persistency, and integrity. An OGSA data service is a web service that implements one or more of the base data interfaces to enable access to, and management of, data resources in a distributed environment. The three base interfaces, Data Access, Data Factory, and Data Management, define basic operations for representing, accessing, creating, and managing data.

- Resource Management Services Provide management capabilities for grid resources: management of the resources themselves, management of the resources as grid components, and management of the OGSA infrastructure. For example, resources can be monitored, reserved, deployed, and configured as needed to meet application QoS requirements. It also requires an information model (semantics) and data model (representation) of the grid resources and services.

- Security Services Facilitate the enforcement of security-related policies within a (virtual) organization, and supports safe resource sharing. Authentication, authorization, and integrity assurance are essential functionalities provided by these services.

- Information Services Provide efficient production of, and access to, information about the grid and its constituent resources. The term <information= refers to dynamic data or events used for

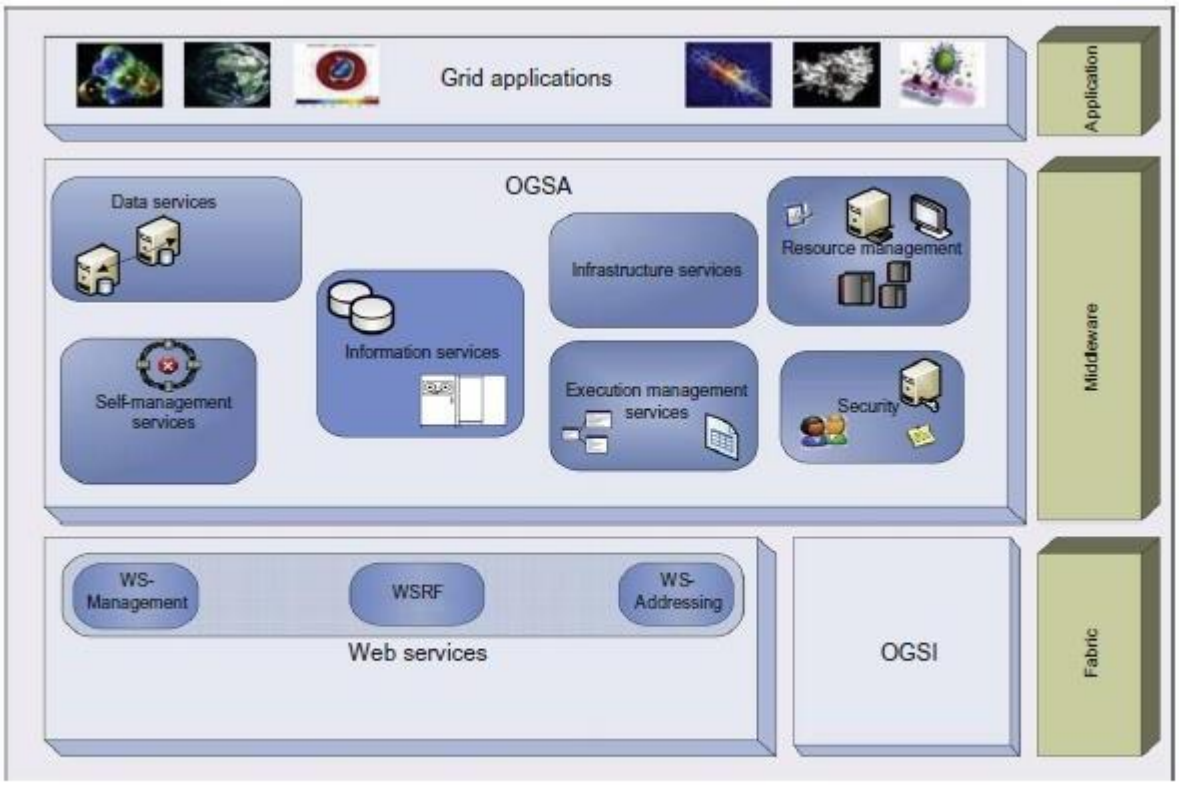


FIGURE 5.5
The OGSA architecture.

status monitoring; relatively static data used for discovery; and any data that is logged. Troubleshooting is just one of the possible uses for information provided by these services.

- Self-Management Services Support service-level attainment for a set of services (or resources), with as much automation as possible, to reduce the costs and complexity of managing the system. These services are essential in addressing the increasing complexity of owning and operating an IT infrastructure.

OGSA has been adopted as reference grid architecture by a number of grid projects. The first prototype grid service implementation was demonstrated January 29, 2002, at a Globus Toolkit tutorial held at Argonne National Laboratory. Since then, the Globus Toolkit 3.0 and 3.2 have offered an OGSA implementation based on OGSI. Two key properties of a grid service are transience and statefulness. Creation and destruction of a transient grid service can be done dynamically. The creation and lifetime of OGSA grid services are handled following the <factory pattern,= to be explained in Section 7.3.1. Web service technologies are designed to support loosely coupled, coarse-grained dynamic systems, and hence do not meet all grid requirements, such as keeping state information, and thus they are unable to fully address the wide range of distributed systems OGSA is designed to support.

OGSA applies a set of WSDL extensions to represent the identifiers necessary to implement a grid service instance across any system. These extensions were defined by OGSI. A key extension is the grid service reference: a network-wide pointer to a specific grid service instance, which makes that instance accessible to

remote client applications. These extensions, including the Grid Service Handle (GSH) and Grid Service Reference (GSR), will be described in Chapter 7. These extensions include stateful grid services and the shortcomings of OGSI with its dense and long specifications. Further problems concern incompatibility with some current web service tools and the fact that it takes a lot of concepts from object orientation.

Unlike the nature of web services, this has led to close cooperation between the grid and web service communities. As a result of these joint efforts, the Web Services Resource Framework (WSRF) [27], WS-Addressing [28], and WS-Notification (WSN) specifications have been proposed to OASIS. Conse- quently, OGSI extensions to web services have been deprecated in favor of new web service standards, and in particular, WSRF. WSRF is a collection of five different specifications. Of course, they all relate to the management of WS-Resources. Table 5.5 depicts WSRF-related interface operations.

Plain web services are usually stateless. This means the web service can't <remember= informa- tion, or keep state, from one invocation to another. However, since a web service is stateless, the following invocations have no idea of what was done in the previous invocations. Grid applications generally require web services to keep state information as they interact with the clients or other web services. The purpose of WSRF is to define a generic framework for modeling and accessing

Table 5.5 WSRF and Its Related Specifications

Specification			Description
WSRF Specifications	WS-ResourceProperties		Standardizes the definition of the resource properties, its association with the WS interface, and the messages defining the query and update capability against resource properties
	WS-ResourceLifetime		Provides standard mechanisms to manage the life cycle of WS-resources (e.g., setting termination time)
	WS-ServiceGroup		Standard expression of aggregating web services and WS-Resources
	WS-Basefault		Provides a standard way of reporting faults
WSRF-Related Specifications	WS-Notification	WS-Base Notification	Proposes a standard way of expressing the basic roles involved in web service publish and subscribe for notification message exchange
		WS-BrokeredNotification	Standardizes message exchanges involved in web service publish and subscribe of a message broker
		WS-Topics	Defines a mechanism to organize and categorize items of interest for a subscription known as "topics"
	WS-Addressing		Transport-neutral mechanisms to address web service and messages

persistent resources using web services in order to facilitate the definition and implementation of a service and the integration and management of multiple services. Note that <stateless= services can, in fact, remember state if that is carried in messages they receive. These could contain a token remembered in a cookie on the client side and a database or cache accessed by the service. Again, the user accessing a stateless service can establish state for the session through the user login that references permanent information stored in a database.

The state information of a web service is kept in a separate entity called a resource. A service may have more than one (singleton) resource, distinguished by assigning a unique key to each

resource. Resources can be either in memory or persistent, stored in secondary storage such as a file or database. The pairing of a web service with a resource is called a WS-Resource. The preferred way of addressing a specific WS-Resource is to use the qualified endpoint reference (EPR) construct, proposed by the WS-Addressing specification. Resources store actual data items, referred to as resource properties. Resource properties are usually used to keep service data values, providing information on the current state of the service, or metadata about such values, or they may contain information required to manage the state, such as the time when the resource must be destroyed. Currently, the Globus Toolkit 4.0 provides a set of OGSA capabilities based on WSRF.

4. Other Service-Oriented Architectures and Systems:

A survey of services and how they are used can be found in [29]. Here we give two examples: one of a system and one of a small grid.

4. Programming on Amazon AWS and Microsoft Azure: (*****)

1. Programming on Amazon EC2:

Amazon was the first company to introduce VMs in application hosting. Customers can rent VMs instead of physical machines to run their own applications. By using VMs, customers can load any software of their choice. The elastic feature of such a service is that a customer can create, launch, and terminate server instances as needed, paying by the hour for active servers. Amazon provides several types of preinstalled VMs. Instances are often called Amazon Machine Images (AMIs) which are preconfigured with operating systems based on Linux or Windows, and additional software.

Table 6.12 defines three types of AMI. Figure 6.24 shows an execution environment. AMIs are the templates for instances, which are running VMs.
The workflow to create a VM is Create an AMI→Create Key Pair→Configure Firewall→Launch

Table 6.12 Three Types of AMI

Image Type	AMI Definition
Private AMI	Images created by you, which are private by default. You can grant access to other users to launch your private images.
Public AMI	Images created by users and released to the AWS community, so anyone can launch instances based on them and use them any way they like. AWS lists all public images at http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=171.
Paid QAMI	You can create images providing specific functions that can be launched by anyone willing to pay you per each hour of usage on top of Amazon's charges.

This sequence is supported by public, private, and paid AMIs shown in Figure 6.24. The AMIs are formed from the virtualized compute, storage, and server resources shown at the bottom of Figure 6.23.

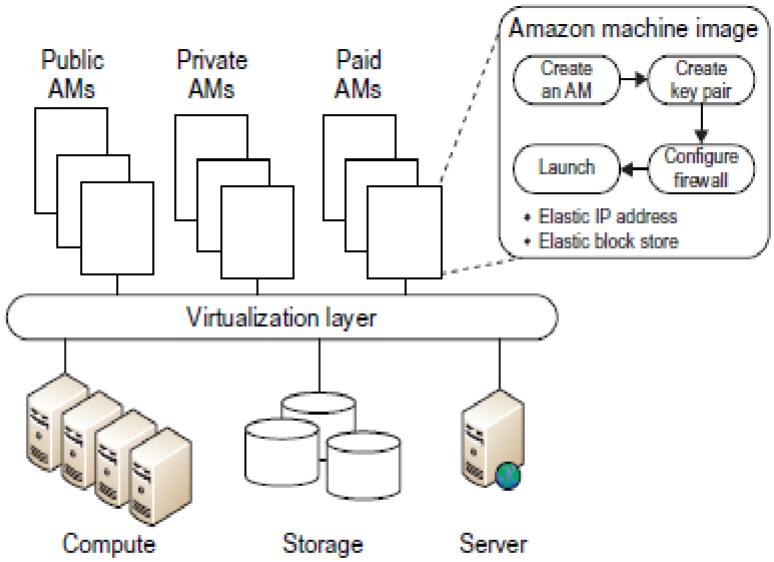


FIGURE 6.23

Amazon EC2 execution environment.

2. Amazon Simple Storage Service (S3): (*****)

Amazon S3 provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. S3 provides the object-oriented storage service for users. Users can access their objects through Simple Object Access Protocol (SOAP) with either browsers or other client programs which support SOAP. SQS is responsible for ensuring a reliable message service between two processes, even if the receiver processes are not running.

Figure 6.24 shows the S3 execution environment.

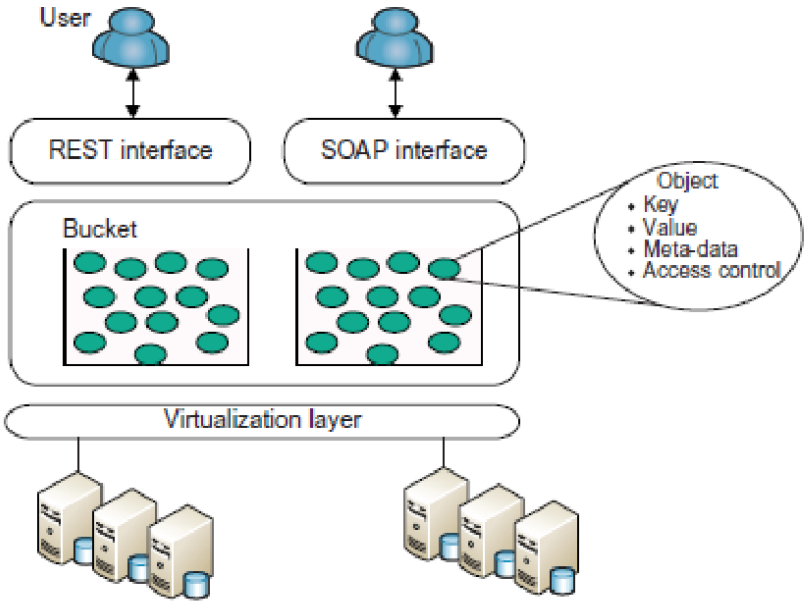


FIGURE 6.24

Amazon S3 execution environment.

The fundamental operation unit of S3 is called an object. Each object is stored in a bucket and retrieved via a unique, developer-assigned key. In other words, the bucket is the container of the object. Besides unique key attributes, the object has other attributes such as values, metadata, and access control information. From the programmer's perspective, the storage provided by S3 can be viewed as a very coarse-grained key-value pair. Through the key-value programming interface, users can write, read, and delete objects containing from 1 byte to 5 gigabytes of data each.
There are two types of web service interface for the user to access the data stored in Amazon clouds. One is a REST (web 2.0) interface, and the other is a SOAP interface. Here are some key features of S3:

• Redundant through geographic dispersion.
• Designed to provide 99.999999999 percent durability and 99.99 percent availability of objects over a given year with cheaper reduced redundancy storage (RRS).
• Authentication mechanisms to ensure that data is kept secure from unauthorized access. Objects can be made private or public, and rights can be granted to specific users.
• Per-object URLs and ACLs (access control lists).
• Default download protocol of HTTP. A BitTorrent protocol interface is provided to lower costs for high-scale distribution.
• $0.055 (more than 5,000 TB) to 0.15 per GB per month storage (depending on total amount).
• First 1 GB per month input or output free and then $.08 to $0.15 per GB for transfers outside an S3 region.
• There is no data transfer charge for data transferred between Amazon EC2 and Amazon S3 within the same region or for data transferred between the Amazon EC2 Northern Virginia region and the Amazon S3 U.S. Standard region (as of October 6, 2010).

3. Amazon Elastic Block Store (EBS) and SimpleDB:

The Elastic Block Store (EBS) provides the volume block interface for saving and restoring the virtual images of EC2 instances. Traditional EC2 instances will be destroyed after use. The status of EC2 can now be saved in the EBS system after the machine is shut down. Users can use EBS to save persistent data and mount to the running instances of EC2. Note that S3 is "Storage as a Service" with a messaging interface. EBS is analogous to a distributed file system accessed by traditional OS disk access mechanisms. EBS allows you to create storage volumes from 1 GB to 1 TB that can be mounted as EC2 instances.

Multiple volumes can be mounted to the same instance. These storage volumes behave like raw, unformatted block devices, with user-supplied device names and a block device interface. You can create a file system on top of Amazon EBS volumes, or use them in any other way you would use a block device (like a hard drive). Snapshots are provided so that the data can be saved incrementally.

This can improve performance when saving and restoring data. In terms of pricing, Amazon provides a similar pay-per-use schema as EC2 and S3. Volume storage charges are based on the amount of storage users allocate until it is released, and is priced at $0.10 per GB/month. EBS also charges $0.10 per 1 million I/O requests made to the storage (as of October 6, 2010). The equivalent of EBS has been offered in open source clouds such Nimbus.

Amazon SimpleDB Service:

SimpleDB provides a simplified data model based on the relational database data model. Structured data from users must be organized into domains. Each domain can be considered a table. The items are the rows in the table. A cell in the table is recognized as the value for a specific attribute (column name) of the corresponding row. This is similar to a table in a relational database. However, it is possible to assign multiple values to a single cell in the table. This is not permitted in a traditional relational database which wants to maintain data consistency.

Many developers simply want to quickly store, access, and query the stored data. SimpleDB removes the requirement to maintain database schemas with strong consistency. SimpleDB is priced at $0.140 per Amazon SimpleDB Machine Hour consumed with the first 25 Amazon SimpleDB Machine Hours consumed per month free (as of October 6, 2010). SimpleDB, like Azure Table, could be called "LittleTable," as they are aimed at managing small amounts of information stored in a distributed table; one could say BigTable is aimed at basic big data, whereas LittleTable is aimed at metadata. Amazon Dynamo is an early research system along the lines of the production SimpleDB system.

4. Microsoft Azure Programming Support: (**********)

When the system is running, services are monitored and one can access event logs, trace/debug data, performance counters, IIS web server logs, crash dumps, and other log files. This information can be saved in Azure storage. Note that there is no debugging capability for running cloud applications, but debugging is done from a trace. One can divide the basic features into storage and compute capabilities.

The Azure application is linked to the Internet through a customized compute VM called a web role supporting basic Microsoft web hosting. Such configured VMs are often called appliances. The other important compute class is the worker role reflecting the importance in cloud computing of a pool of compute resources that are scheduled as needed.

The roles support HTTP(S) and TCP. Roles offer the following methods:

- The OnStart() method which is called by the Fabric on startup, and allows you to perform initialization tasks. It reports a Busy status to the load balancer until you return true.
- The OnStop() method which is called when the role is to be shut down and gives a graceful exit.
- The Run() method which contains the main logic.

The Azure concept of roles is an interesting idea that we can expect to be expanded in terms of role types and use in other cloud environments.

Figure 6.25 shows that compute roles can be load-balanced, similar to what GAE and AWS clouds have done.

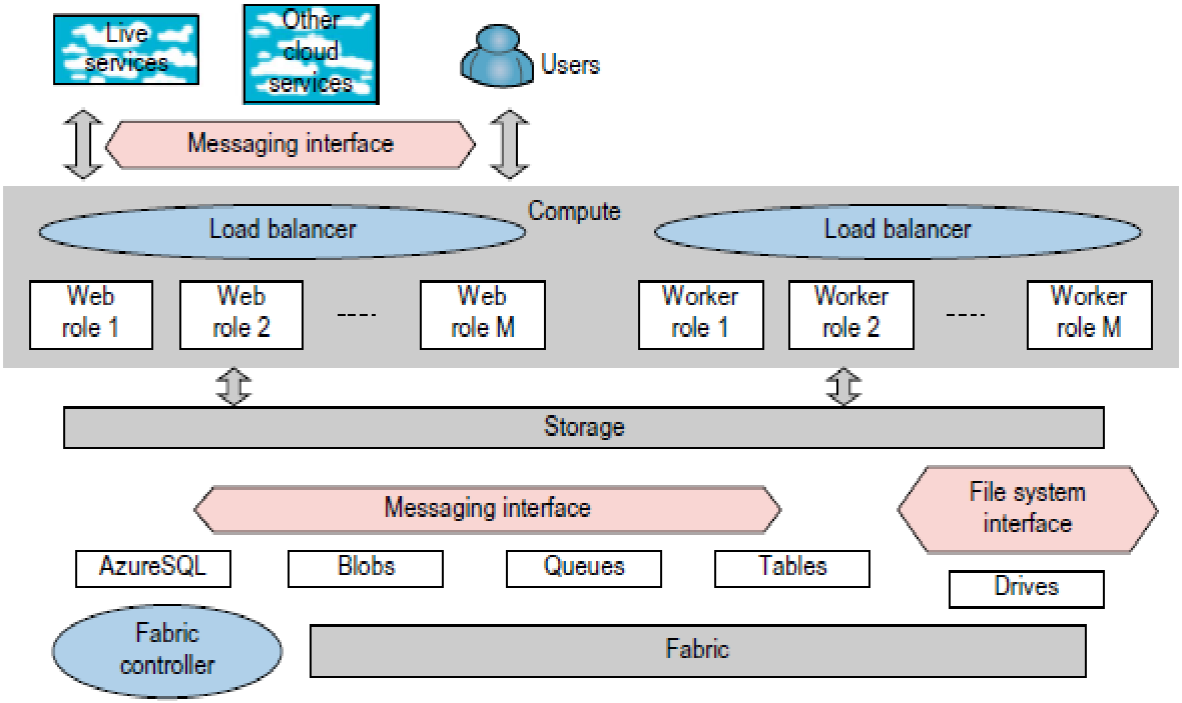


FIGURE 6.25

Features of the Azure cloud platform.

SQLAzure: (********)

Azure offers a very rich set of storage capabilities, as shown in Figure 6.25. The SQLAzure service offers SQL Server as a service and is described in detail in Example 6.10. All the storage modalities are accessed with REST interfaces except for the recently introduced Drives that are analogous to Amazon EBS and offer a file system interface as a durable NTFS volume backed by blob storage. The REST interfaces are automatically associated with URLs and all storage is replicated three times for fault tolerance and is guaranteed to be consistent in access.

The basic storage system is built from blobs which are analogous to S3 for Amazon. Blobs are arranged as a three-level hierarchy: Account → Containers → Page or Block Blobs. Containers are analogous to directories in traditional file systems with the account acting as the root. The block blob is used for streaming data and each such blob is made up as a sequence of blocks of up to 4MB each, while each block has a 64 byte ID. Block blobs can be up to 200GB in size.

Page blobs are for random read/write access and consist of an array of pages with a maximum blob size of 1 TB. One can associate metadata with blobs as <name, value> pairs with up to 8 KB per blob.

Example 6.10 SQLAzure Data Services

Azure provides a sophisticated database programming interface. (More details can be found in www.microsoft .com/azure/sql.mspx). The SQLAzure data services can be viewed as more of a traditional relational database. Built on top of the current mature commercial software packages, SQLAzure can be considered a highly scalable, on-demand data storage and query processing utility service. The service interface of SQLAzure is based on standard web protocols and SQLAzure supports both SOAP and REST. As based on the relational database, the data model provided by SQL Data Services (SDS) is richer than the two NOSQL data management services discussed earlier (BigTable and SimpleDB).

The data model in SQLAzure includes the concepts of authority, container, and entity, with a flexible schema. After a user signs up for the data service, she can create an authority which is represented as a DNS name such as mydomain.data.database.windows.net. Here, mydomain is an authority created by the user and data.database.windows.net refers to the service. Users can create multiple authorities at any time. This DNS name will be resolved to a specific IP address and maps to a specific data center. Thus, an authority as well as its data will be located at a single data center. Under the highest level "authority" comes the concept of a container.

An authority can have multiple containers (or none at all). A container has its identifications which can be used as the handle to find the corresponding container in an authority.

Containers are the places where users can put their data. Just like SimpleDB, users can begin to put data in a container without worrying about the data schema. Entities are the units that are stored in a container. An entity can store any number of user-defined properties and associated values (i.e., like the attributes and values in SimpleDB). There are two different kinds of containers: homogeneous and heterogeneous containers. A homogeneous container has all entities of the same type, like a relational database table. A heterogeneous container does not have such limitations. Figure 6.25 shows these concepts.

SDS is one of the building blocks of the Azure platform for building cloud applications. It does provide an enterprise-class data platform. Microsoft has built multiple data centers all over the world to host cloud applications from third parties. Multiple data centers can give the stored data high availability and security.

Users do not need to worry about loss of data. Another key feature that SQLAzure provides is ease of development. SDS (in fact, the total Azure Platform SDK) can be integrated with Microsoft's powerful Visual Studio development environment. This can greatly improve the effectiveness and efficiency with which developers can make cloud applications.

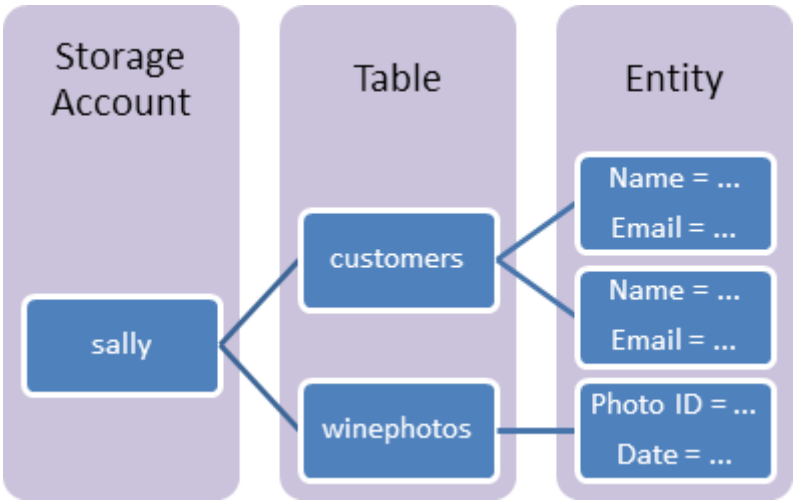
Azure Tables: (********)

The Azure Table and Queue storage modes are aimed at much smaller data volumes. Queues provide reliable message delivery and are naturally used to support work spooling between web and worker roles. Queues consist of an unlimited number of messages which can be retrieved and processed at least once with an 8 KB limit on message size. Azure supports PUT, GET, and DELETE message operations as well as CREATE and DELETE for queues. Each account can have any number of Azure tables which consist of rows called entities and columns called properties.

There is no limit to the number of entities in a table and the technology is designed to scale well to a large number of entities stored on distributed computers. All entities can have up to 255 general properties which are <name, type, value> triples. Two extra properties, PartitionKey and RowKey, must be defined for each entity, but otherwise, there are no constraints on the names of properties—this table is very flexible! RowKey is designed to give each entity a unique label while PartitionKey is designed to be shared and entities with the same PartitionKey are stored next to each other; a good use of PartitionKey can speed up search performance. An entity can have, at most, 1 MB storage; if you need large value sizes, just store a link to a blob store in the Table property value. ADO.NET and LINQ support table queries.

Table storage concepts

Table storage contains the following components:



- **URL format:** Azure Table Storage accounts use this format: `http://<storage account>.table.core.windows.net/<table>`

  Azure Cosmos DB for Table accounts use this format: `http://<storage account>.table.cosmosdb.azure.com/<table>`

  You can address Azure tables directly using this address with the OData protocol. For more information, see OData.org.

- **Accounts:** All access to Azure Storage is done through a storage account. For more information about storage accounts, see Storage account overview.

  All access to Azure Cosmos DB is done through an Azure Cosmos DB for Table account. For more information, see Create an Azure Cosmos DB for Table account.

- **Table**: A table is a collection of entities. Tables don't enforce a schema on entities, which means a single table can contain entities that have different sets of properties.
- **Entity**: An entity is a set of properties, similar to a database row. An entity in Azure Storage can be up to 1MB in size. An entity in Azure Cosmos DB can be up to 2MB in size.
- **Properties**: A property is a name-value pair. Each entity can include up to 252 properties to store data. Each entity also has three system properties that specify a partition key, a row key, and a timestamp. Entities with the same partition key can be queried more quickly, and inserted/updated in atomic operations. An entity's row key is its unique identifier within a partition.

IMPORTANT QUESTIONS

1. **State and explain service models of cloud computing with architectures?**
2. **Define cloud computing? Explain different types of clouds available**
3. **Draw a neat sketch of Google cloud platform and explain?**
4. **What is IaaS? Mention any two IaaS service providers?**
5. **Draw and explain Amazon cloud computing infrastructure?**
6. **List five public cloud offerings of PaaS?**
7. **Draw and explain Microsoft Windows Azure?**
8. **List five public cloud offerings of IaaS?**
9. **What is SOA? Discuss with architecture how two software communicate using SOA.**
10. **Explain the properties of Service Oriented Architecture.**
11. **Explain Amazon Elastic Block Structure (EBS) & Simple DB?**
12. **Explain Amazon Simple Storage S3?**
13. **Write and explain about programming on Amazon AWS and Microsoft Azure?**
14. **Explain SQL Azure & Azure tables?**