MODULE 5

Cloud computing allows end users and developers to leverage large distributed computing infrastructures. This is made possible thanks to infrastructure management software and distributed computing platforms offering on-demand compute, storage, and, on top of these, more advanced services. There are several different options for building enterprise cloud computing applications or for using cloud computing technologies to integrate and extend existing industrial applications. An overview of a few prominent cloud computing platforms and a brief description of the types of service they offer are shown in Table 9.1. A cloud computing system can be developed using either a single technology and vendor or a combination of them.

This chapter presents some of the representative cloud computing solutions offered as Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS) services in the market. It provides some insights into and practical issues surrounding the architecture of the major cloud computing technologies and their service offerings.

9.1 Amazon web services

Amazon Web Services (AWS) is a platform that allows the development of flexible applications by providing solutions for elastic infrastructure scalability, messaging, and data storage. The platform is accessible through SOAP or RESTful Web service interfaces and provides a Web-based console where users can handle administration and monitoring of the resources required, as well as their expenses computed on a pay-as-you-go basis.

Figure 9.1 shows all the services available in the AWS ecosystem. At the base of the solution stack are services that provide raw compute and raw storage: *Amazon Elastic Compute (EC2)* and *Amazon Simple Storage Service (S3)*. These are the two most popular services, which are generally complemented with other offerings for building a complete system. At the higher level, *Elastic MapReduce* and *AutoScaling* provide additional capabilities for building smarter and more elastic computing systems. On the data side, *Elastic Block Store (EBS)*, *Amazon SimpleDB*, *Amazon RDS*, and *Amazon ElastiCache* provide solutions for reliable data snapshots and the management of structured and semistructured data. Communication needs are covered at the networking level by *Amazon Virtual Private Cloud (VPC)*, *Elastic Load Balancing*, *Amazon Route 53*, and *Amazon Direct Connect*. More advanced services for connecting applications are *Amazon Simple Queue*

Table 9.1 Some Example Cloud Computing Offerings

Vendor/Product	Service Type	Description
Amazon Web Services	IaaS, PaaS, SaaS	Amazon Web Services (AWS) is a collection of Web services that provides developers with compute, storage, and more advanced services. AWS is mostly popular for IaaS services and primarily for its elastic compute service EC2.
Google AppEngine	PaaS	Google AppEngine is a distributed and scalable runtime for developing scalable Web applications based on Java and Python runtime environments. These are enriched with access to services that simplify the development of applications in a scalable manner.
Microsoft Azure	PaaS	Microsoft Azure is a cloud operating system that provides services for developing scalable applications based on the proprietary Hyper-V virtualization technology and the .NET framework.
SalesForce.com and Force.com	SaaS, PaaS	SalesForce.com is a Software-as-a-Service solution that allows prototyping of CRM applications. It leverages the Force.com platform, which is made available for developing new components and capabilities for CRM applications.
Heroku	PaaS	Heroku is a scalable runtime environment for building applications based on Ruby.
RightScale	IaaS	Rightscale is a cloud management platform with a single dashboard to manage public and hybrid clouds.

Service (SQS), *Amazon Simple Notification Service (SNS)*, and *Amazon Simple E-mail Service (SES)*. Other services include:

- *Amazon CloudFront* content delivery network solution
- *Amazon CloudWatch* monitoring solution for several Amazon services
- *Amazon Elastic BeanStalk* and *CloudFormation* flexible application packaging and deployment

As shown, AWS comprise a wide set of services. We discuss the most important services by examining the solutions proposed by AWS regarding compute, storage, communication, and complementary services.

9.1.1 Compute services

Compute services constitute the fundamental element of cloud computing systems. The fundamental service in this space is Amazon EC2, which delivers an IaaS solution that has served as a reference model for several offerings from other vendors in the same market segment. Amazon EC2 allows deploying servers in the form of virtual machines created as instances of a specific image. Images come with a preinstalled operating system and a software stack, and instances can be configured for memory, number of processors, and storage. Users are provided with credentials to remotely access the instance and further configure or install software if needed.

9.1.1.1 Amazon machine images

Amazon Machine Images (AMIs) are templates from which it is possible to create a virtual machine. They are stored in Amazon S3 and identified by a unique identifier in the form of *ami-xxxxx* and

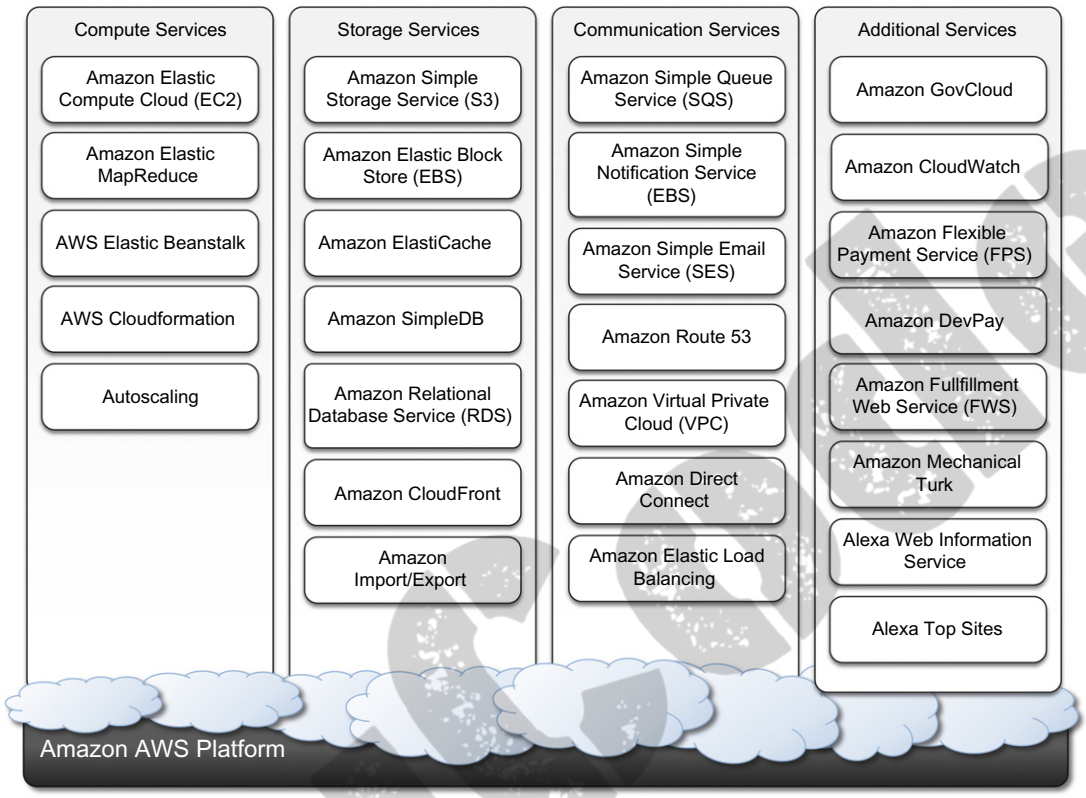

FIGURE 9.1

Amazon Web Services ecosystem.

a manifest XML file. An AMI contains a physical file system layout with a predefined operating system installed. These are specified by the *Amazon Ramdisk Image* (*ARI*, id: *ari-yyyyyy*) and the *Amazon Kernel Image* (*AKI*, id: *aki-zzzzzz*), which are part of the configuration of the template. AMIs are either created from scratch or "bundled" from existing EC2 instances. A common practice is to prepare new AMIs to create an instance from a preexisting AMI, log into it once it is booted and running, and install all the software needed. Using the tools provided by Amazon, we can convert the instance into a new image. Once an AMI is created, it is stored in an S3 bucket and the user can decide whether to make it available to other users or keep it for personal use. Finally, it is also possible to associate a product code with a given AMI, thus allowing the owner of the AMI to get revenue every time this AMI is used to create EC2 instances.

9.1.1.2 EC2 instances

EC2 instances represent virtual machines. They are created using AMI as templates, which are specialized by selecting the number of cores, their computing power, and the installed memory. The processing power is expressed in terms of virtual cores and EC2 Compute Units (ECUs). The ECU

is a measure of the computing power of a virtual core; it is used to express a predictable quantity of real CPU power that is allocated to an instance. By using compute units instead of real frequency values, Amazon can change over time the mapping of such units to the underlying real amount of computing power allocated, thus keeping the performance of EC2 instances consistent with standards set by the times. Over time, the hardware supporting the underlying infrastructure will be replaced by more powerful hardware, and the use of ECUs helps give users a consistent view of the performance offered by EC2 instances. Since users rent computing capacity rather than buying hardware, this approach is reasonable. One ECU is defined as giving the same performance as a 1.0−1.2 GHz 2007 Opteron or 2007 Xeon processor.[1]

Table 9.2 shows all the currently available configurations for EC2 instances. We can identify six major categories:

- *Standard instances*. This class offers a set of configurations that are suitable for most applications. EC2 provides three different categories of increasing computing power, storage, and memory.
- *Micro instances*. This class is suitable for those applications that consume a limited amount of computing power and memory and occasionally need bursts in CPU cycles to process surges in the workload. Micro instances can be used for small Web applications with limited traffic.
- *High-memory instances*. This class targets applications that need to process huge workloads and require large amounts of memory. Three-tier Web applications characterized by high traffic are the target profile. Three categories of increasing memory and CPU are available, with memory proportionally larger than computing power.
- *High-CPU instances*. This class targets compute-intensive applications. Two configurations are available where computing power proportionally increases more than memory.
- *Cluster Compute instances*. This class is used to provide virtual cluster services. Instances in this category are characterized by high CPU compute power and large memory and an extremely high I/O and network performance, which makes it suitable for HPC applications.
- *Cluster GPU instances*. This class provides instances featuring graphic processing units (GPUs) and high compute power, large memory, and extremely high I/O and network performance. This class is particularly suited for cluster applications that perform heavy graphic computations, such as rendering clusters. Since GPU can be used for general-purpose computing, users of such instances can benefit from additional computing power, which makes this class suitable for HPC applications.

EC2 instances are priced hourly according to the category they belong to. At the beginning of every hour of usage, the user will be charged the cost of the entire hour. The hourly expense charged for one instance is constant. Instance owners are responsible for providing their own backup strategies, since there is no guarantee that the instance will run for the entire hour. Another alternative is represented by *spot instances*. These instances are much more dynamic in terms of pricing and lifetime since they are made available to the user according to the load of EC2 and the availability of resources. Users define an upper bound for a price they want to pay for these instances; as long as the current price (the spot price) remains under the given bound, the instance is kept running. The price is sampled at the beginning of each hour. Spot instances are more volatile than normal instances; whereas for normal instances EC2 will try as much as possible to keep

[1] http://aws.amazon.com/ec2/faqs/#What_is_an_EC2_Compute_Unit_and_why_did_you_introduce_it.

Table 9.2 Amazon EC2 (On-Demand) Instances Characteristics

Instance Type	ECU	Platform	Memory	Disk Storage	Price (U.S. East) (USD/hour)
Standard instances					
Small	1(1 × 1)	32 bit	1.7 GB	160 GB	$0.085 Linux $0.12 Windows
Large	4(2 × 2)	64 bit	7.5 GB	850 GB	$0.340 Linux $0.48 Windows
Extra Large	8(4 × 2)	64 bit	15 GB	1,690 GB	$0.680 Linux $0.96 Windows
Micro instances					
Micro	< = 2	32/64 bit	613 MB	EBS Only	$0.020 Linux $0.03 Windows
High-Memory instances					
Extra Large	6.5(2 × 3.25)	64 bit	17.1 GB	420 GB	$0.500 Linux $0.62 Windows
Double Extra Large	13(4 × 3.25)	64 bit	34.2 GB	850 GB	$1.000 Linux $1.24 Windows
Quadruple Extra Large	26(8 × 3.25)	64 bit	68.4 GB	1,690 GB	$2.000 Linux $2.48 Windows
High-CPU instances					
Medium	5(2 × 2.5)	32 bit	1.7 GB	350 GB	$0.170 Linux $0.29 Windows
Extra Large	20(8 × 2.5)	64 bit	7 GB	1,690 GB	$0.680 Linux $1.16 Windows
Cluster instances					
Quadruple Extra Large	33.5	64 bit	23 GB	1,690 GB	$1.600 Linux $1.98 Windows
Cluster GPU instances					
Quadruple Extra Large	33.5	64 bit	22 GB	1,690 GB	$2.100 Linux $2.60 Windows

them active, there is no such guarantee for spot instances. Therefore, implementing backup and checkpointing strategies is inevitable.

EC2 instances can be run either by using the command-line tools provided by Amazon, which connects the Amazon Web Service that provides remote access to the EC2 infrastructure, or via the AWS console, which allows the management of other services, such as S3. By default an EC2 instance is created with the kernel and the disk associated to the AMI. These define the architecture (32 bit or 64 bit) and the space of disk available to the instance. This is an ephemeral disk; once the instance is shut down, the content of the disk will be lost. Alternatively, it is possible to attach an EBS volume to the instance, the content of which will be stored in S3. If the default AKI and ARI are not suitable, EC2 provides capabilities to run EC2 instances by specifying a different AKI and ARI, thus giving flexibility in the creation of instances.

9.1.1.3 EC2 environment

EC2 instances are executed within a virtual environment, which provides them with the services they require to host applications. The EC2 environment is in charge of allocating addresses, attaching storage volumes, and configuring security in terms of access control and network connectivity.

By default, instances are created with an internal IP address, which makes them capable of communicating within the EC2 network and accessing the Internet as clients. It is possible to associate an *Elastic IP* to each instance, which can then be remapped to a different instance over time. Elastic IPs allow instances running in EC2 to act as servers reachable from the Internet and, since they are not strictly bound to specific instances, to implement failover capabilities. Together with an external IP, EC2 instances are also given a domain name that generally is in the form *ec2-xxx-xxx-xxx.compute-x.amazonaws.com*, where *xxx-xxx-xxx* normally represents the four parts of the external IP address separated by a dash, and *compute-x* gives information about the availability zone where instances are deployed. Currently, there are five availability zones that are priced differently: two in the United States (Virginia and Northern California), one in Europe (Ireland), and two in Asia Pacific (Singapore and Tokyo).

Instance owners can partially control where to deploy instances. Instead, they have a finer control over the security of the instances as well as their network accessibility. Instance owners can associate a key pair to one or more instances when these instances are created. A key pair allows the owner to remotely connect to the instance once this is running and gain root access to it. Amazon EC2 controls the accessibility of a virtual instance with basic firewall configuration, allowing the specification of source address, port, and protocols (TCP, UDP, ICMP). Rules can also be attached to security groups, and instances can be made part of one or more groups before their deployment. Security groups and firewall rules constitute a flexible way of providing basic security for EC2 instances, which has to be complemented by appropriate security configuration within the instance itself.

9.1.1.4 Advanced compute services

EC2 instances and AMIs constitute the basic blocks for building an IaaS computing cloud. On top of these, Amazon Web Services provide more sophisticated services that allow the easy packaging and deploying of applications and a computing platform that supports the execution of MapReduce-based applications.

AWS CloudFormation

AWS CloudFormation constitutes an extension of the simple deployment model that characterizes EC2 instances. CloudFormation introduces the concepts of *templates*, which are JSON formatted text files that describe the resources needed to run an application or a service in EC2 together with the relations between them. CloudFormation allows easily and explicitly linking EC2 instances together and introducing dependencies among them. Templates provide a simple and declarative way to build complex systems and integrate EC2 instances with other AWS services such as S3, SimpleDB, SQS, SNS, Route 53, Elastic Beanstalk, and others.

AWS elastic beanstalk

AWS Elastic Beanstalk constitutes a simple and easy way to package applications and deploy them on the AWS Cloud. This service simplifies the process of provisioning instances and deploying

application code and provides appropriate access to them. Currently, this service is available only for Web applications developed with the Java/Tomcat technology stack. Developers can conveniently package their Web application into a WAR file and use Beanstalk to automate its deployment on the AWS Cloud.

With respect to other solutions that automate cloud deployment, Beanstalk simplifies tedious tasks without removing the user's capability of accessing—and taking over control of—the underlying EC2 instances that make up the virtual infrastructure on top of which the application is running. With respect to AWS CloudFormation, AWS Elastic Beanstalk provides a higher-level approach for application deployment on the cloud, which does not require the user to specify the infrastructure in terms of EC2 instances and their dependencies.

Amazon elastic MapReduce

Amazon Elastic MapReduce provides AWS users with a cloud computing platform for MapReduce applications. It utilizes Hadoop as the MapReduce engine, deployed on a virtual infrastructure composed of EC2 instances, and uses Amazon S3 for storage needs.

Apart from supporting all the application stack connected to Hadoop (Pig, Hive, etc.), Elastic MapReduce introduces elasticity and allows users to dynamically size the Hadoop cluster according to their needs, as well as select the appropriate configuration of EC2 instances to compose the cluster (Small, High-Memory, High-CPU, Cluster Compute, and Cluster GPU). On top of these services, basic Web applications allowing users to quickly run data-intensive applications without writing code are offered.

9.1.2 Storage services

AWS provides a collection of services for data storage and information management. The core service in this area is represented by Amazon *Simple Storage Service (S3)*. This is a distributed object store that allows users to store information in different formats. The core components of S3 are two: *buckets* and *objects*. Buckets represent virtual containers in which to store objects; objects represent the content that is actually stored. Objects can also be enriched with metadata that can be used to tag the stored content with additional information.

9.1.2.1 S3 key concepts

As the name suggests, S3 has been designed to provide a simple storage service that's accessible through a Representational State Transfer (REST) interface, which is quite similar to a distributed file system but which presents some important differences that allow the infrastructure to be highly efficient:

- *The storage is organized in a two-level hierarchy*. S3 organizes its storage space into buckets that cannot be further partitioned. This means that it is not possible to create directories or other kinds of physical groupings for objects stored in a bucket. Despite this fact, there are few limitations in naming objects, and this allows users to simulate directories and create logical groupings.
- *Stored objects cannot be manipulated like standard files*. S3 has been designed to essentially provide storage for objects that will not change over time. Therefore, it does not allow renaming, modifying, or relocating an object. Once an object has been added to a bucket, its

content and position is immutable, and the only way to change it is to remove the object from the store and add it again.

- *Content is not immediately available to users.* The main design goal of S3 is to provide an eventually consistent data store. As a result, because it is a large distributed storage facility, changes are not immediately reflected. For instance, S3 uses replication to provide redundancy and efficiently serve objects across the globe; this practice introduces latencies when adding objects to the store—especially large ones—which are not available instantly across the entire globe.
- *Requests will occasionally fail.* Due to the large distributed infrastructure being managed, requests for object may occasionally fail. Under certain conditions, S3 can decide to drop a request by returning an internal server error. Therefore, it is expected to have a small failure rate during day-to-day operations, which is generally not identified as a persistent failure.

Access to S3 is provided with RESTful Web services. These express all the operations that can be performed on the storage in the form of HTTP requests (*GET, PUT, DELETE, HEAD*, and *POST*), which operate differently according to the element they address. As a rule of thumb *PUT/POST* requests add new content to the store, *GET/HEAD* requests are used to retrieve content and information, and *DELETE* requests are used to remove elements or information attached to them.

Resource naming

Buckets, objects, and attached metadata are made accessible through a REST interface. Therefore, they are represented by *uniform resource identifiers (URIs)* under the *s3.amazonaws.com* domain. All the operations are then performed by expressing the entity they are directed to in the form of a request for a URI.

Amazon offers three different ways of addressing a bucket:

- *Canonical form:* *http://s3.amazonaws.com/bukect_name/*. The bucket name is expressed as a path component of the domain name s3.amazonaws.com. This is the naming convention that has less restriction in terms of allowed characters, since all the characters that are allowed for a path component can be used.
- *Subdomain form:* *http://bucketname.s3.amazon.com/*. Alternatively, it is also possible to reference a bucket as a subdomain of s3.amazonaws.com. To express a bucket name in this form, the name has to do all of the following:
  - Be between 3 and 63 characters long
  - Contain only letters, numbers, periods, and dashes
  - Start with a letter or a number
  - Contain at least one letter
  - Have no fragments between periods that start with a dash or end with a dash or that are empty strings

  This form is equivalent to the previous one when it can be used, but it is the one to be preferred since it works more effectively for all the geographical locations serving resources stored in S3.
- *Virtual hosting form:* *http://bucket-name.com/*. Amazon also allows referencing of its resources with custom URLs. This is accomplished by entering a CNAME record into the DNS that points to the subdomain form of the bucket URI.

Since S3 is logically organized as a flat data store, all the buckets are managed under the s3.amazonaws.com domain. Therefore, the names of buckets must be unique across all the users.

Objects are always referred as resources local to a given bucket. Therefore, they always appear as a part of the resource component of a URI. Since a bucket can be expressed in three different ways, objects indirectly inherit this flexibility:

- Canonical form: http://s3.amazonaws.com/bukect_name/object_name
- Subdomain form: http://bucket-name/s3.amzonaws.com/object_name
- Virtual hosting form: http://bucket-name.com/object_name

Except for the *?*, which separates the resource path of a URI from the set of parameters passed with the request, all the characters that follow the */* after the bucket reference constitute the name of the object. For instance, path separator characters expressed as part of the object name do not have corresponding physical layout within the bucket store. Despite this fact, they can still be used to create logical groupings that look like directories.

Finally, specific information about a given object, such as its access control policy or the server logging settings defined for a bucket, can be referenced using a specific parameter. More precisely:

- Object ACL: http://s3.amazonaws.com/bukect_name/object_name?acl
- Bucket server logging: http://s3.amzonaws.com/bucket_name?logging

Object metadata are not directly accessible through a specific URI, but they are manipulated by adding attributes in the request of the URL and are not part of the identifier.

Buckets

A *bucket* is a container of objects. It can be thought of as a virtual drive hosted on the S3 distributed storage, which provides users with a flat store to which they can add objects. Buckets are top-level elements of the S3 storage architecture and do not support nesting. That is, it is not possible to create "subbuckets" or other kinds of physical divisions.

A bucket is located in a specific geographic location and eventually replicated for fault tolerance and better content distribution. Users can select the location at which to create buckets, which by default are created in Amazon's U.S. datacenters. Once a bucket is created, all the objects that belong to the bucket will be stored in the same availability zone of the bucket. Users create a bucket by sending a *PUT* request to http://s3.amazonaws.com/ with the name of the bucket and, if they want to specify the availability zone, additional information about the preferred location. The content of a bucket can be listed by sending a *GET* request specifying the name of the bucket. Once created, the bucket cannot be renamed or relocated. If it is necessary to do so, the bucket needs to be deleted and recreated. The deletion of a bucket is performed by a *DELETE* request, which can be successful if and only if the bucket is empty.

Objects and metadata

Objects constitute the content elements stored in S3. Users either store files or push to the S3 text stream representing the object's content. An object is identified by a name that needs to be unique within the bucket in which the content is stored. The name cannot be longer than 1,024 bytes when encoded in UTF-8, and it allows almost any character. Since buckets do not support nesting, even

characters normally used as path separators are allowed. This actually compensates for the lack of a structured file system, since directories can be emulated by properly naming objects.

Users create an object via a *PUT* request that specifies the name of the object together with the bucket name, its contents, and additional properties. The maximum size of an object is 5 GB. Once an object is created, it cannot be modified, renamed, or moved into another bucket. It is possible to retrieve an object via a *GET* request; deleting an object is performed via a *DELETE* request.

Objects can be tagged with metadata, which are passed as properties of the *PUT* request. Such properties are retrieved either with a *GET* request or with a *HEAD* request, which only returns the object's metadata without the content. Metadata are both system and user defined: the first ones are used by S3 to control the interaction with the object, whereas the second ones are meaningful to the user, who can store up to 2 KB per metadata property represented by a key-value pair of strings.

Access control and security

Amazon S3 allows controlling the access to buckets and objects by means of *Access Control Policies (ACPs)*. An ACP is a set of *grant permissions* that are attached to a resource expressed by means of an XML configuration file. A policy allows defining up to 100 access rules, each of them granting one of the available permissions to a grantee. Currently, five different permissions can be used:

- *READ* allows the grantee to retrieve an object and its metadata and to list the content of a bucket as well as getting its metadata.
- *WRITE* allows the grantee to add an object to a bucket as well as modify and remove it.
- *READ_ACP* allows the grantee to read the ACP of a resource.
- *WRITE_ACP* allows the grantee to modify the ACP of a resource.
- *FULL_CONTROL* grants all of the preceding permissions.

Grantees can be either single users or groups. Users can be identified by their canonical IDs or the email addresses they provided when they signed up for S3. For groups, only three options are available: all users, authenticated users, and log delivery users.[2]

Once a resource is created, S3 attaches a default ACP granting full control permissions to its owner only. Changes to the ACP can be made by using the request to the resource URI followed by *?acl*. A *GET* method allows retrieval of the ACP; a *PUT* method allows uploading of a new ACP to replace the existing one. Alternatively, it is possible to use a predefined set of permissions called *canned policies* to set the ACP at the time a resource is created. These policies represent the most common access patterns for S3 resources.

ACPs provide a set of powerful rules to control S3 users' access to resources, but they do not exhibit fine grain in the case of nonauthenticated users, who cannot be differentiated and are considered as a group. To provide a finer grain in this scenario, S3 allows defining *signed URIs*, which grant access to a resource for a limited amount of time to all the requests that can provide a temporary access token.

[2]This group identifies a specific group of accounts that automated processes use to perform bucket access logging.

Advanced features

Besides the management of buckets, objects, and ACPs, S3 offers other additional features that can be helpful. These features are server access logging and integration with the *BitTorrent* file-sharing network.

Server access logging allows bucket owners to obtain detailed information about the request made for the bucket and all the objects it contains. By default, this feature is turned off; it can be activated by issuing a *PUT* request to the bucket URI followed by *?logging*. The request should include an XML file specifying the target bucket in which to save the logging files and the file name prefix. A *GET* request to the same URI allows the user to retrieve the existing logging configuration for the bucket.

The second feature of interest is represented by the capability of exposing S3 objects to the *BitTorrent* network, thus allowing files stored in S3 to be downloaded using the *BitTorrent* protocol. This is done by appending *?torrent* to the URI of the S3 object. To actually download the object, its ACP must grant read permission to everyone.

9.1.2.2 Amazon elastic block store

The Amazon Elastic Block Store (EBS) allows AWS users to provide EC2 instances with persistent storage in the form of volumes that can be mounted at instance startup. They accommodate up to 1 TB of space and are accessed through a block device interface, thus allowing users to format them according to the needs of the instance they are connected to (raw storage, file system, or other). The content of an EBS volume survives the instance life cycle and is persisted into S3. EBS volumes can be cloned, used as boot partitions, and constitute durable storage since they rely on S3 and it is possible to take incremental snapshots of their content.

EBS volumes normally reside within the same availability zone of the EC2 instances that will use them to maximize the I/O performance. It is also possible to connect volumes located in different availability zones. Once mounted as volumes, their content is lazily loaded in the background and according to the request made by the operating system. This reduces the number of I/O requests that go to the network. Volume images cannot be shared among instances, but multiple (separate) active volumes can be created from them. In addition, it is possible to attach multiple volumes to a single instance or create a volume from a given snapshot and modify its size, if the formatted file system allows such an operation.

The expense related to a volume comprises the cost generated by the amount of storage occupied in S3 and by the number of I/O requests performed against the volume. Currently, Amazon charges $0.10/GB/month of allocated storage and $0.10 per 1 million requests made to the volume.

9.1.2.3 Amazon ElastiCache

ElastiCache is an implementation of an elastic in-memory cache based on a cluster of EC2 instances. It provides fast data access from other EC2 instances through a Memcached-compatible protocol so that existing applications based on such technology do not need to be modified and can transparently migrate to ElastiCache.

ElastiCache is based on a cluster of EC2 instances running the caching software, which is made available through Web services. An ElastiCache cluster can be dynamically resized according to the demand of the client applications. Furthermore, automatic patch management and failure

detection and recovery of cache nodes allow the cache cluster to keep running without administrative intervention from AWS users, who have only to elastically size the cluster when needed.

ElastiCache nodes are priced according to the EC2 costing model, with a small price difference due to the use of the caching service installed on such instances. It is possible to choose between different types of instances; Table 9.3 provides an overview of the pricing options.

The prices indicated in Table 9.3 are related to the Amazon offerings during 2011–2012, and the amount of memory specified represents the memory available after taking system software overhead into account.

9.1.2.4 Structured storage solutions

Enterprise applications quite often rely on databases to store data in a structured form, index, and perform analytics against it. Traditionally, RDBMS have been the common data back-end for a wide range of applications, even though recently more scalable and lightweight solutions have been proposed. Amazon provides applications with structured storage services in three different forms: preconfigured EC2 AMIs, *Amazon Relational Data Storage (RDS)*, and *Amazon SimpleDB*.

Preconfigured EC2 AMIs

Preconfigured EC2 AMIs are predefined templates featuring an installation of a given database management system. EC2 instances created from these AMIs can be completed with an EBS volume for storage persistence. Available AMIs include installations of IBM DB2, Microsoft SQL Server, MySQL, Oracle, PostgreSQL, Sybase, and Vertica. Instances are priced hourly according to the EC2 cost model. This solution poses most of the administrative burden on the EC2 user, who has to configure, maintain, and manage the relational database, but offers the greatest variety of products to choose from.

Amazon RDS

RDS is relational database service that relies on the EC2 infrastructure and is managed by Amazon. Developers do not have to worry about configuring the storage for high availability, designing

Table 9.3 Amazon EC2 (On-Demand) Cache Instances Characteristics, 2011–2012

Instance Type	ECU	Platform	Memory	I/O Capacity	Price (U.S. East) (USD/hour)
Standard instances					
Small	1(1 × 1)	64 bit	1.3 GB	Moderate	$0.095
Large	4(2 × 2)	64 bit	7.1 GB	High	$0.380
Extra Large	8(4 × 2)	64 bit	14.6 GB	High	$0.760
High-Memory instances					
Extra Large	6.5(2 × 3.25)	64 bit	16.7 GB	High	$0.560
Double Extra Large	13(4 × 3.25)	64 bit	33.8 GB	High	$1.120
Quadruple Extra Large	26(8 × 3.25)	64 bit	68 GB	High	$2.240
High-CPU instances					
Extra Large	26(8 × 3.25)	64 bit	6.6 GB	High	$0.760

failover strategies, or keeping the servers up-to-date with patches. Moreover, the service provides users with automatic backups, snapshots, point-in-time recoveries, and facilities for implementing replications. These and the common database management services are available through the AWS console or a specific Web service. Two relational engines are available: MySQL and Oracle.

Two key advanced features of RDS are *multi-AZ deployment* and *read replicas*. The first option provides users with a failover infrastructure for their RDBMS solutions. The high-availability solution is implemented by keeping in standby synchronized copies of the services in different availability zones that are activated if the primary service goes down. The second option provides users with increased performance for applications that are heavily based on database reads. In this case, Amazon deploys copies of the primary service that are only available for database reads, thus cutting down the response time of the service.

The available options and the relative pricing of the service during 2011–2012 are shown in Table 9.4. The table shows the costing details of the on-demand instances. There is also the possibility of using reserved instances for long terms (one to three years) by paying up-front at discounted hourly rates.

With respect to the previous solution, users are not responsible for managing, configuring, and patching the database management software, but these operations are performed by the AWS. In addition, support for elastic management of servers is simplified. Therefore, this solution is optimal for applications based on the Oracle and MySQL engines, which are migrated on the AWS infrastructure and require a scalable database solution.

Amazon SimpleDB

Amazon SimpleDB is a lightweight, highly scalable, and flexible data storage solution for applications that do not require a fully relational model for their data. SimpleDB provides support for semistructured data, the model for which is based on the concept of *domains*, *items*, and *attributes*. With respect to the relational model, this model provides fewer constraints on the structure of data entries, thus obtaining improved performance in querying large quantities of data. As happens for Amazon RDS, this service frees AWS users from performing configuration, management, and high-availability design for their data stores.

Table 9.4 Amazon RDS (On-Demand) Instances Characteristics, 2011–2012

Instance Type	ECU	Platform	Memory	I/O Capacity	Price (U.S. East) (USD/hour)
Standard instances					
Small	1(1 × 1)	64 bit	1.7 GB	Moderate	$0.11
Large	4(2 × 2)	64 bit	7.5 GB	High	$0.44
Extra Large	8(4 × 2)	64 bit	15 GB	High	$0.88
High-Memory instances					
Extra Large	6.5(2 × 3.25)	64 bit	17.1 GB	High	$0.65
Double Extra Large	13(4 × 3.25)	64 bit	34 GB	High	$1.30
Quadruple Extra Large	26(8 × 3.25)	64 bit	68 GB	High	$2.60

SimpleDB uses *domains* as top-level elements to organize a data store. These domains are roughly comparable to tables in the relational model. Unlike tables, they allow items not to have all the same column structure; each item is therefore represented as a collection of attributes expressed in the form of a key-value pair. Each domain can grow up to 10 GB of data, and by default a single user can allocate a maximum of 250 domains. Clients can create, delete, modify, and make snapshots of domains. They can insert, modify, delete, and query items and attributes. Batch insertion and deletion are also supported. The capability of querying data is one of the most relevant functions of the model, and the *select* clause supports the following test operators: $=$, $!=$, $<$, $>$, $<=$, $>=$, *like*, *not like*, *between*, *is null*, *is not null*, and *every()*. Here is a simple example on how to query data:

```
select * from domain_name where every(attribute_name) = 'value'
```

Moreover, the *select* operator can extend its query beyond the boundaries of a single domain, thus allowing users to query effectively a large amount of data.

To efficiently provide AWS users with a scalable and fault-tolerant service, SimpleDB implements a relaxed constraint model, which leads to *eventually consistent* data. The adverb *eventually* denotes the fact that multiple accesses on the same data might not read the same value in the very short term, but they will eventually converge over time. This is because SimpleDB does not lock all the copies of the data during an update, which is propagated in the background. Therefore, there is a transient period of time in which different clients can access different copies of the same data that have different values. This approach is very scalable with minor drawbacks, and it is also reasonable, since the application scenario for SimpleDB is mostly characterized by querying and indexing operations on data. Alternatively, it is possible to change the default behavior and ensure that all the readers are blocked during an update.

Even though SimpleDB is not a transactional model, it allows clients to express conditional insertions or deletions, which are useful to prevent lost updates in multiple-writer scenarios. In this case, the operation is executed if and only if the condition is verified. This condition can be used to check preexisting values of attributes for an item.

Table 9.5 provides an overview of the pricing options for the SimpleDB service for data transfer during 2011–2012. The service charges either for data transfer or stored data. Data transfer within the AWS network is not charged. In addition, SimpleDB also charges users for machine usage. The first 25 SimpleDB instances per month are free; after this threshold there is an hourly charge ($0.140 hour in the U.S. East region).

If we compare this cost model with the one characterizing S3, it becomes evident that S3 is a cheaper option for storing large objects. This is useful information for clarifying the different nature of SimpleDB with respect to S3: The former has been designed to provide fast access to semistructured collections of small objects and not for being a long-term storage option for large objects.

9.1.2.5 Amazon CloudFront

CloudFront is an implementation of a content delivery network on top of the Amazon distributed storage infrastructure. It leverages a collection of edge servers strategically located around the globe to better serve requests for static and streaming Web content so that the transfer time is reduced as much as possible.

Table 9.5 Amazon SimpleDB Data Transfer Charges, 2011–2012

Instance Type	Price (U.S. East) (USD)
Data Transfer In	
All data transfer in	$0.000
Data Transfer Out	
1st GB/month	$0.000
Up to 10 TB/month	$0.120
Next 40 TB/month	$0.090
Next 100 TB/month	$0.070
Next 350 TB/month	$0.050
Next 524 TB/month	Special arrangements
Next 4 PB/month	Special arrangements
Greater than 5 PB/month	Special arrangements

AWS provides users with simple Web service APIs to manage CloudFront. To make available content through CloudFront, it is necessary to create a distribution. This identifies an origin server, which contains the original version of the content being distributed, and it is referenced by a DNS domain under the *Cloudfront.net* domain name (i.e., my-distribution.Cloudfront.net). It is also possible to map a given domain name to a distribution. Once the distribution is created, it is sufficient to reference the distribution name, and the CloudFront engine will redirect the request to the closest replica and eventually download the original version from the origin server if the content is not found or expired on the selected edge server.

The content that can be delivered through CloudFront is static (HTTP and HTTPS) or streaming (Real Time Messaging Protocol, or RMTP). The origin server hosting the original copy of the distributed content can be an S3 bucket, an EC2 instance, or a server external to the Amazon network. Users can restrict access to the distribution to only one or a few of the available protocols, or they can set up access rules for finer control. It is also possible to invalidate content to remove it from the distribution or force its update before expiration.

Table 9.6 provides a breakdown of the pricing during 2011–2012. Note that CloudFront is cheaper than S3. This reflects its different purpose: CloudFront is designed to optimize the distribution of very popular content that is frequently downloaded, potentially from the entire globe and not only the Amazon network.

9.1.3 Communication services

Amazon provides facilities to structure and facilitate the communication among existing applications and services residing within the AWS infrastructure. These facilities can be organized into two major categories: *virtual networking* and *messaging*.

9.1.3.1 Virtual networking

Virtual networking comprises a collection of services that allow AWS users to control the connectivity to and between compute and storage services. *Amazon Virtual Private Cloud (VPC)* and

Table 9.6 Amazon CloudFront On-Demand Pricing, 2011–2012

Pricing Item	United States	Europe	Hong Kong and Singapore	Japan	South America
Requests					
Per 10,000 HTTP requests	$0.0075	$0.0090	$0.0090	$0.0095	$0.0160
Per 10,000 HTTPS requests	$0.0100	$0.0120	$0.0120	$0.0130	$0.0220
Regional Data Transfer Out					
First 10 TB/month	$0.120/GB	$0.120/GB	$0.190/GB	$0.201/GB	$0.250/GB
Next 40 TB/month	$0.080/GB	$0.080/GB	$0.140/GB	$0.148/GB	$0.200/GB
Next 100 TB/month	$0.060/GB	$0.060/GB	$0.120/GB	$0.127/GB	$0.180/GB
Next 350 TB/month	$0.040/GB	$0.040/GB	$0.100/GB	$0.106/GB	$0.160/GB
Next 524 TB/month	$0.030/GB	$0.030/GB	$0.080/GB	$0.085/GB	$0.140/GB
Next 4 PB/month	$0.025/GB	$0.025/GB	$0.070/GB	$0.075/GB	$0.130/GB
Greater than 5 PB/month	$0.020/GB	$0.020/GB	$0.060/GB	$0.065/GB	$0.125/GB

Amazon Direct Connect provide connectivity solutions in terms of infrastructure; *Route 53* facilitates connectivity in terms of naming.

Amazon VPC provides a great degree of flexibility in creating virtual private networks within the Amazon infrastructure and beyond. The service providers prepare either templates covering most of the usual scenarios or a fully customizable network service for advanced configurations. Prepared templates include public subnets, isolated networks, private networks accessing Internet through network address translation (NAT), and hybrid networks including AWS resources and private resources. Also, it is possible to control connectivity between different services (EC2 instances and S3 buckets) by using the *Identity Access Management (IAM)* service. During 2011, the cost of Amazon VPC was $0.50 per connection hour.

Amazon Direct Connect allows AWS users to create dedicated networks between the user private network and Amazon Direct Connect locations, called *ports*. This connection can be further partitioned in multiple logical connections and give access to the public resources hosted on the Amazon infrastructure. The advantage of using Direct Connect versus other solutions is the consistent performance of the connection between the users' premises and the Direct Connect locations. This service is compatible with other services such as EC2, S3, and Amazon VPC and can be used in scenarios requiring high bandwidth between the Amazon network and the outside world. There are only two available ports located in the United States, but users can leverage external providers that offer guaranteed high bandwidth to these ports. Two different bandwidths can be chosen: 1 Gbps, priced at $0.30 per hour, and 10 Gbps, priced at $2.25 per hour. Inbound traffic is free; outbound traffic is priced at $0.02 per GB.

Amazon Route 53 implements dynamic DNS services that allow AWS resources to be reached through domain names different from the amazon.com domain. By leveraging the large and globally distributed network of Amazon DNS servers, AWS users can expose EC2 instances or S3 buckets as resources under a domain of their property, for which Amazon DNS servers become

authoritative.[3] EC2 instances are likely to be more dynamic than the physical machines, and S3 buckets might also exist for a limited time. To cope with such a volatile nature, the service provides AWS users with the capability of dynamically mapping names to resources as instances are launched on EC2 or as new buckets are created in S3. By interacting with the Route 53 Web service, users can manage a set of *hosted zones*, which represent the user domains controlled by the service, and edit the resources made available through it. Currently, a single user can have up to 100 zones. The costing model includes a fixed amount ($1 per zone per month) and a dynamic component that depends on the number of queries resolved by the service for the hosted zones ($0.50 per million queries for the first billion of queries a month, $0.25 per million queries over 1 billion of queries a month).

9.1.3.2 Messaging

Messaging services constitute the next step in connecting applications by leveraging AWS capabilities. The three different types of messaging services offered are *Amazon Simple Queue Service (SQS)*, *Amazon Simple Notification Service (SNS)*, and *Amazon Simple Email Service (SES)*.

Amazon SQS constitutes disconnected model for exchanging messages between applications by means of message queues, hosted within the AWS infrastructure. Using the AWS console or directly the underlying Web service AWS, users can create an unlimited number of message queues and configure them to control their access. Applications can send messages to any queue they have access to. These messages are securely and redundantly stored within the AWS infrastructure for a limited period of time, and they can be accessed by other (authorized) applications. While a message is being read, it is kept locked to avoid spurious processing from other applications. Such a lock will expire after a given period.

Amazon SNS provides a publish-subscribe method for connecting heterogeneous applications. With respect to Amazon SQS, where it is necessary to continuously poll a given queue for a new message to process, Amazon SNS allows applications to be notified when new content of interest is available. This feature is accessible through a Web service whereby AWS users can create a topic, which other applications can subscribe to. At any time, applications can publish content on a given topic and subscribers can be automatically notified. The service provides subscribers with different notification models (HTTP/HTTPS, email/email JSON, and SQS).

Amazon SES provides AWS users with a scalable email service that leverages the AWS infrastructure. Once users are signed up for the service, they have to provide an email that SES will use to send emails on their behalf. To activate the service, SES will send an email to verify the given address and provide the users with the necessary information for the activation. Upon verification, the user is given an SES sandbox to test the service, and he can request access to the production version. Using SES, it is possible to send either SMTP-compliant emails or raw emails by specifying email headers and Multipurpose Internet Mail Extension (MIME) types. Emails are queued for

[3] A DNS server is responsible for resolving a name to a corresponding IP address. Since DNS servers implement a distributed database without a single global control, a single DNS server does not have the complete knowledge of all the mappings between names and IP addresses, but it has direct knowledge only of a small subset of them. Such a DNS server is therefore authoritative for these names because it can directly resolve the names. For resolving the other names, the nearest authoritative DNS is contacted.

delivery, and the users are notified of any failed delivery. SES also provides a wide range of statistics that help users to improve their email campaigns for effective communication with customers.

With regard to the costing, all three services do not require a minimum commitment but are based on a pay-as-you go model. Currently, users are not charged until they reach a minimum threshold. In addition, data transfer-in is not charged, but data transfer-out is charged by ranges.

9.1.4 Additional services

Besides compute, storage, and communication services, AWS provides a collection of services that allow users to utilize services in aggregation. The two relevant services are *Amazon CloudWatch* and *Amazon Flexible Payment Service (FPS)*.

Amazon CloudWatch is a service that provides a comprehensive set of statistics that help developers understand and optimize the behavior of their application hosted on AWS. CloudWatch collects information from several other AWS services: EC2, S3, SimpleDB, CloudFront, and others. Using CloudWatch, developers can see a detailed breakdown of their usage of the service they are renting on AWS and can devise more efficient and cost-saving applications. Earlier services of CloudWatch were offered only through subscription, but now it is made available for free to all the AWS users.

Amazon FPS infrastructure allows AWS users to leverage Amazon's billing infrastructure to sell goods and services to other AWS users. Using Amazon FPS, developers do not have to set up alternative payment methods, and they can charge users via a billing service. The payment models available through FPS include one-time payments and delayed and periodic payments, required by subscriptions and usage-based services, transactions, and aggregate multiple payments.

9.1.5 Summary

Amazon provides a complete set of services for developing, deploying, and managing cloud computing systems by leveraging the large and distributed AWS infrastructure. Developers can use EC2 to control and configure the computing infrastructure hosted in the cloud. They can leverage other services, such as AWS CloudFormation, Elastic Beanstalk, or Elastic MapReduce, if they do not need complete control over the computing stack. Applications hosted in the AWS Cloud can leverage S3, SimpleDB, or other storage services to manage structured and unstructured data. These services are primarily meant for storage, but other options, such as Amazon SQS, SNS, and SES, provide solutions for dynamically connecting applications from both inside and outside the AWS Cloud. Network connectivity to AWS applications is addressed by Amazon VPC and Amazon Direct Connect.

9.2 Google AppEngine

Google AppEngine is a PaaS implementation that provides services for developing and hosting scalable Web applications. AppEngine is essentially a distributed and scalable runtime environment that leverages Google's distributed infrastructure to scale out applications facing a large number of requests by allocating more computing resources to them and balancing the load among them. The runtime is completed by a collection of services that allow developers to design and implement

applications that naturally scale on AppEngine. Developers can develop applications in Java, Python, and Go, a new programming language developed by Google to simplify the development of Web applications. Application usage of Google resources and services is metered by AppEngine, which bills users when their applications finish their free quotas.

9.2.1 Architecture and core concepts

AppEngine is a platform for developing scalable applications accessible through the Web (see Figure 9.2). The platform is logically divided into four major components: infrastructure, the runtime environment, the underlying storage, and the set of scalable services that can be used to develop applications.

9.2.1.1 Infrastructure

AppEngine hosts Web applications, and its primary function is to serve users requests efficiently. To do so, AppEngine's infrastructure takes advantage of many servers available within Google datacenters. For each HTTP request, AppEngine locates the servers hosting the application that processes the request, evaluates their load, and, if necessary, allocates additional resources (i.e., servers) or redirects the request to an existing server. The particular design of applications, which does not expect any state information to be implicitly maintained between requests to the same application, simplifies the work of the infrastructure, which can redirect each of the requests to any of the servers hosting the target application or even allocate a new one.

The infrastructure is also responsible for monitoring application performance and collecting statistics on which the billing is calculated.

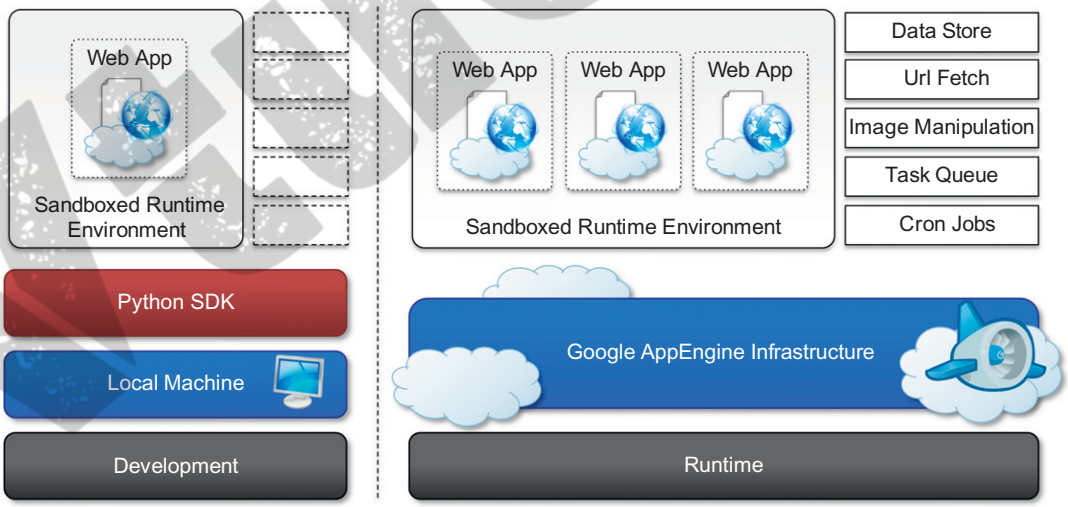

FIGURE 9.2

Google AppEngine platform architecture.

9.2.1.2 Runtime environment

The runtime environment represents the execution context of applications hosted on AppEngine. With reference to the AppEngine infrastructure code, which is always active and running, the runtime comes into existence when the request handler starts executing and terminates once the handler has completed.

Sandboxing

One of the major responsibilities of the runtime environment is to provide the application environment with an isolated and protected context in which it can execute without causing a threat to the server and without being influenced by other applications. In other words, it provides applications with a *sandbox*.

Currently, AppEngine supports applications that are developed only with managed or interpreted languages, which by design require a runtime for translating their code into executable instructions. Therefore, sandboxing is achieved by means of modified runtimes for applications that disable some of the common features normally available with their default implementations. If an application tries to perform any operation that is considered potentially harmful, an exception is thrown and the execution is interrupted. Some of the operations that are not allowed in the sandbox include writing to the server's file system; accessing computer through network besides using *Mail*, *UrlFetch*, and *XMPP*; executing code outside the scope of a request, a queued task, and a cron job; and processing a request for more than 30 seconds.

Supported runtimes

Currently, it is possible to develop AppEngine applications using three different languages and related technologies: *Java*, *Python*, and *Go*.

AppEngine currently supports Java 6, and developers can use the common tools for Web application development in Java, such as the *Java Server Pages (JSP)*, and the applications interact with the environment by using the *Java Servlet* standard. Furthermore, access to AppEngine services is provided by means of Java libraries that expose specific interfaces of provider-specific implementations of a given abstraction layer. Developers can create applications with the AppEngine Java SDK, which allows developing applications with either Java 5 or Java 6 and by using any Java library that does not exceed the restrictions imposed by the sandbox.

Support for Python is provided by an optimized Python 2.5.2 interpreter. As with Java, the runtime environment supports the Python standard library, but some of the modules that implement potentially harmful operations have been removed, and attempts to import such modules or to call specific methods generate exceptions. To support application development, AppEngine offers a rich set of libraries connecting applications to AppEngine services. In addition, developers can use a specific Python Web application framework, called *webapp*, simplifying the development of Web applications.

The Go runtime environment allows applications developed with the Go programming language to be hosted and executed in AppEngine. Currently the release of Go that is supported by AppEngine is r58.1. The SDK includes the compiler and the standard libraries for developing applications in Go and interfacing it with AppEngine services. As with the Python environment, some of the functionalities have been removed or generate a runtime exception. In addition, developers can include third-party libraries in their applications as long as they are implemented in pure Go.

9.2.1.3 Storage

AppEngine provides various types of storage, which operate differently depending on the volatility of the data. There are three different levels of storage: in memory-cache, storage for semistructured data, and long-term storage for static data. In this section, we describe *DataStore* and the use of static file servers. We cover *MemCache* in the application services section.

Static file servers

Web applications are composed of dynamic and static data. Dynamic data are a result of the logic of the application and the interaction with the user. Static data often are mostly constituted of the components that define the graphical layout of the application (CSS files, plain HTML files, JavaScript files, images, icons, and sound files) or data files. These files can be hosted on static file servers, since they are not frequently modified. Such servers are optimized for serving static content, and users can specify how dynamic content should be served when uploading their applications to AppEngine.

DataStore

DataStore is a service that allows developers to store semistructured data. The service is designed to scale and optimized to quickly access data. DataStore can be considered as a large object database in which to store objects that can be retrieved by a specified key. Both the type of the key and the structure of the object can vary.

With respect to the traditional Web applications backed by a relational database, DataStore imposes less constraint on the regularity of the data but, at the same time, does not implement some of the features of the relational model (such as reference constraints and join operations). These design decisions originated from a careful analysis of data usage patterns for Web applications and were taken in order to obtain a more scalable and efficient data store. The underlying infrastructure of *DataStore* is based on *Bigtable* [93], a redundant, distributed, and semistructured data store that organizes data in the form of tables (see Section 8.2.1).

DataStore provides high-level abstractions that simplify interaction with Bigtable. Developers define their data in terms of *entity* and *properties*, and these are persisted and maintained by the service into tables in *Bigtable*. An entity constitutes the level of granularity for the storage, and it identifies a collection of properties that define the data it stores. Properties are defined according to one of the several primitive types supported by the service. Each entity is associated with a key, which is either provided by the user or created automatically by AppEngine. An entity is associated with a *named kind* that AppEngine uses to optimize its retrieval from Bigtable. Although entities and properties seem to be similar to rows and tables in SQL, there are a few differences that have to be taken into account. Entities of the same kind might not have the same properties, and properties of the same name might contain values of different types. Moreover, properties can store different versions of the same values. Finally, keys are immutable elements and, once created, they cannot be changed.

DataStore also provides facilities for creating indexes on data and to update data within the context of a transaction. Indexes are used to support and speed up queries. A query can return zero or more objects of the same kind or simply the corresponding keys. It is possible to query the data store by specifying either the key or conditions on the values of the properties. Returned result sets can be sorted by key value or properties value. Even though the queries are quite similar to SQL

queries, their implementation is substantially different. DataStore has been designed to be extremely fast in returning result sets; to do so it needs to know in advance all the possible queries that can be done for a given kind, because it stores for each of them a separate index. The indexes are provided by the user while uploading the application to AppEngine and can be automatically defined by the development server. When the developer tests the application, the server monitors all the different types of queries made against the simulated data store and creates an index for them. The structure of the indexes is saved in a configuration file and can be further changed by the developer before uploading the application. The use of precomputed indexes makes the query execution time-independent from the size of the stored data but only influenced by the size of the result set.

The implementation of transaction is limited in order to keep the store scalable and fast. AppEngine ensures that the update of a single entity is performed atomically. Multiple operations on the same entity can be performed within the context of a transaction. It is also possible to update multiple entities atomically. This is only possible if these entities belong to the same *entity group.* The entity group to which an entity belongs is specified at the time of entity creation and cannot be changed later. With regard to concurrency, AppEngine uses an *optimistic concurrency control*: If one user tries to update an entity that is already being updated, the control returns and the operation fails. Retrieving an entity never incurs into exceptions.

9.2.1.4 Application services

Applications hosted on AppEngine take the most from the services made available through the runtime environment. These services simplify most of the common operations that are performed in Web applications: access to data, account management, integration of external resources, messaging and communication, image manipulation, and asynchronous computation.

UrlFetch

Web 2.0 has introduced the concept of composite Web applications. Different resources are put together and organized as meshes within a single Web page. Meshes are fragments of HTML generated in different ways. They can be directly obtained from a remote server or rendered from an XML document retrieved from a Web service, or they can be rendered by the browser as the result of an embedded and remote component. A common characteristic of all these examples is the fact that the resource is not local to the server and often not even in the same administrative domain. Therefore, it is fundamental for Web applications to be able to retrieve remote resources.

The sandbox environment does not allow applications to open arbitrary connections through sockets, but it does provide developers with the capability of retrieving a remote resource through HTTP/HTTPS by means of the *UrlFetch* service. Applications can make synchronous and asynchronous Web requests and integrate the resources obtained in this way into the normal request-handling cycle of the application. One of the interesting features of UrlFetch is the ability to set deadlines for requests so that they can be completed (or aborted) within a given time. Moreover, the ability to perform such requests asynchronously allows the applications to continue with their logic while the resource is retrieved in the background. UrlFetch is not only used to integrate meshes into a Web page but also to leverage remote Web services in accordance with the SOA reference model for distributed applications.

MemCache

AppEngine provides developers with access to fast and reliable storage, which is DataStore. Despite this, the main objective of the service is to serve as a scalable and long-term storage, where data are persisted to disk redundantly in order to ensure reliability and availability of data against failures. This design poses a limit on how much faster the store can be compared to other solutions, especially for objects that are frequently accessed—for example, at each Web request.

AppEngine provides caching services by means of *MemCache*. This is a distributed in-memory cache that is optimized for fast access and provides developers with a volatile store for the objects that are frequently accessed. The caching algorithm implemented by MemCache will automatically remove the objects that are rarely accessed. The use of MemCache can significantly reduce the access time to data; developers can structure their applications so that each object is first looked up into MemCache and if there is a miss, it will be retrieved from DataStore and put into the cache for future lookups.

Mail and instant messaging

Communication is another important aspect of Web applications. It is common to use email for following up with users about operations performed by the application. Email can also be used to trigger activities in Web applications. To facilitate the implementation of such tasks, AppEngine provides developers with the ability to send and receive mails through *Mail*. The service allows sending email on behalf of the application to specific user accounts. It is also possible to include several types of attachments and to target multiple recipients. Mail operates asynchronously, and in case of failed delivery the sending address is notified through an email detailing the error.

AppEngine provides also another way to communicate with the external world: the Extensible Messaging and Presence Protocol (XMPP). Any chat service that supports XMPP, such as Google Talk, can send and receive chat messages to and from the Web application, which is identified by its own address. Even though the chat is a communication medium mostly used for human interactions, XMPP can be conveniently used to connect the Web application with chat bots or to implement a small administrative console.

Account management

Web applications often keep various data that customize their interaction with users. These data normally go under the user profile and are attached to an account. AppEngine simplifies account management by allowing developers to leverage Google account management by means of *Google Accounts*. The integration with the service also allows Web applications to offload the implementation of authentication capabilities to Google's authentication system.

Using Google Accounts, Web applications can conveniently store profile settings in the form of key-value pairs, attach them to a given Google account, and quickly retrieve them once the user authenticates. With respect to a custom solution, the use of Google Accounts requires users to have a Google account, but it does not require any further implementation. The use of Google Accounts is particularly advantageous for developing Web applications within a corporate environment using Google Apps. In this case, the applications can be easily integrated with all the other services (and profile settings) included in Google Apps.

Image manipulation

Web applications render pages with graphics. Often simple operations, such as adding watermarks or applying simple filters, are required. AppEngine allows applications to perform image resizing, rotation, mirroring, and enhancement by means of *Image Manipulation*, a service that is also used in other Google products. Image Manipulation is mostly designed for lightweight image processing and is optimized for speed.

9.2.1.5 Compute services

Web applications are mostly designed to interface applications with users by means of a ubiquitous channel, that is, the Web. Most of the interaction is performed synchronously: Users navigate the Web pages and get instantaneous feedback in response to their actions. This feedback is often the result of some computation happening on the Web application, which implements the intended logic to serve the user request. Sometimes this approach is not applicable—for example, in long computations or when some operations need to be triggered at a given point in time. A good design for these scenarios provides the user with immediate feedback and a notification once the required operation is completed. AppEngine offers additional services such as *Task Queues* and *Cron Jobs* that simplify the execution of computations that are off-bandwidth or those that cannot be performed within the timeframe of the Web request.

Task queues

Task Queues allow applications to submit a task for a later execution. This service is particularly useful for long computations that cannot be completed within the maximum response time of a request handler. The service allows users to have up to 10 queues that can execute tasks at a configurable rate.

In fact, a task is defined by a Web request to a given URL, and the queue invokes the request handler by passing the payload as part of the Web request to the handler. It is the responsibility of the request handler to perform the "task execution," which is seen from the queue as a simple Web request. The queue is designed to reexecute the task in case of failure in order to avoid transient failures preventing the task from a successful completion.

Cron jobs

Sometimes the length of computation might not be the primary reason that an operation is not performed within the scope of the Web request. It might be possible that the required operation needs to be performed at a specific time of the day, which does not coincide with the time of the Web request. In this case, it is possible to schedule the required operation at the desired time by using the *Cron Jobs* service. This service operates similarly to Task Queues but invokes the request handler specified in the task at a given time and does not reexecute the task in case of failure. This behavior can be useful to implement maintenance operations or send periodic notifications.

9.2.2 Application life cycle

AppEngine provides support for almost all the phases characterizing the life cycle of an application: testing and development, deployment, and monitoring. The SDKs released by Google provide

developers with most of the functionalities required by these tasks. Currently there are two SDKs available for development: Java SDK and Python SDK.

9.2.2.1 Application development and testing

Developers can start building their Web applications on a local development server. This is a self-contained environment that helps developers tune applications without uploading them to AppEngine. The development server simulates the AppEngine runtime environment by providing a mock implementation of DataStore, MemCache, UrlFetch, and the other services leveraged by Web applications. Besides hosting Web applications, the development server contains a complete set of monitoring features that are helpful to profile the behavior of applications, especially regarding access to the DataStore service and the queries performed against it. This is a particularly important feature that will be of relevance in deploying the application to AppEngine. As discussed earlier, AppEngine builds indexes for each of the queries performed by a given application in order to speed up access to the relevant data. This capability is enabled by *a priori* knowledge about all the possible queries made by the application; such knowledge is made available to AppEngine by the developer while uploading the application. The development server analyzes application behavior while running and traces all the queries made during testing and development, thus providing the required information about the indexes to be built.

Java SDK

The Java SDK provides developers with the facility for building applications with the Java 5 and Java 6 runtime environments. Alternatively, it is possible to develop applications within the Eclipse development environment by using the Google AppEngine plug-in, which integrates the features of the SDK within the powerful Eclipse environment. Using the Eclipse software installer, it is possible to download and install Java SDK, Google Web Toolkit, and Google AppEngine plug-ins into Eclipse. These three components allow developers to program powerful and rich Java applications for AppEngine.

The SDK supports the development of applications by using the *servlet* abstraction, which is a common development model. Together with servlets, many other features are available to build applications. Moreover, developers can easily create Web applications by using the *Eclipse Web Platform*, which provides a set of tools and components.

The plug-in allows developing, testing, and deploying applications on AppEngine. Other tasks, such as retrieving the log of applications, are available by means of command-line tools that are part of the SDK.

Python SDK

The Python SDK allows developing Web applications for AppEngine with Python 2.5. It provides a standalone tool, called *GoogleAppEngineLauncher*, for managing Web applications locally and deploying them to AppEngine. The tool provides a convenient user interface that lists all the available Web applications, controls their execution, and integrates them with the default code editor for editing application files. In addition, the launcher provides access to some important services for application monitoring and analysis, such as the logs, the SDK console, and the dashboard. The log console captures all the information that is logged by the application while it is running. The console SDK provides developers with a Web interface via which they can see the application profile

in terms of utilized resource. This feature is particularly useful because it allows developers to preview the behavior of the applications once they are deployed on AppEngine, and it can be used to tune applications made available through the runtime.

The Python implementation of the SDK also comes with an integrated Web application framework called *webapp* that includes a set of models, components, and tools that simplify the development of Web applications and enforce a set of coherent practices. This is not the only Web framework that can be used to develop Web applications. There are dozens of available Python Web frameworks that can be used. However, due to the restrictions enforced by the sandboxed environment, all of them cannot be used seamlessly. The *webapp* framework has been reimplemented and made available in the Python SDK so that it can be used with AppEngine. Another Web framework that is known to work well is *Django*.[4]

The SDK is completed by a set of command-line tools that allows developers to perform all the operations available through the launcher and more from the command shell.

9.2.2.2 Application deployment and management

Once the application has been developed and tested, it can be deployed on AppEngine with a simple click or command-line tool. Before performing such task, it is necessary to create an application identifier, which will be used to locate the application from the Web browser by typing the address *http://<application-id>.appspot.com*. Alternatively, it is also possible to map the application with a registered DNS domain name. This is particularly useful for commercial development, where users want to make the application available through a more appropriate name.

An application identifier is mandatory because it allows unique identification of the application while it's interacting with AppEngine. Developers use an app identifier to upload and update applications. Besides being unique, it also needs to be compliant to the rules that are enforced for domain names. It is possible to register an application identifier by logging into AppEngine and selecting the "Create application" option. It is also possible to provide an application title that is descriptive of the application; the title can be changed over time.

Once an application identifier has been created, it is possible to deploy an application on AppEngine. This task can be done using either the respective development environment (*GoogleAppEngineLauncher* and *Google AppEngine* plug-in) or the command-line tools. Once the application is uploaded, nothing else needs to be done to make it available. AppEngine will take care of everything. Developers can then manage the application by using the administrative console. This is the primary tool used for application monitoring and provides users with insight into resource usage (CPU, bandwidth) and services and other useful counters. It is also possible to manage multiple versions of a single application, select the one available for the release, and manage its billing-related issues.

9.2.3 Cost model

AppEngine provides a free service with limited quotas that get reset every 24 hours. Once the application has been tested and tuned for AppEngine, it is possible to set up a billing account and obtain more allowance and be charged on a pay-per-use basis. This allows developers to identify the appropriate daily budget that they want to allocate for a given application.

[4] www.djangoproject.com.

An application is measured against *billable quotas*, *fixed quotas*, and *per-minute quotas*. Google AppEngine uses these quotas to ensure that users do not spend more than the allocated budget and that applications run without being influenced by each other from a performance point of view. Billable quotas identify the daily quotas that are set by the application administrator and are defined by the daily budget allocated for the application. AppEngine will ensure that the application does not exceed these quotas. Free quotas are part of the billable quota and identify the portion of the quota for which users are not charged. Fixed quotas are internal quotas set by AppEngine that identify the infrastructure boundaries and define operations that the application can carry out on the infrastructure (services and runtime). These quotas are generally bigger than billable quotas and are set by AppEngine to avoid applications impacting each other's performance or overloading the infrastructure. The costing model also includes per-minute quotas, which are defined in order to avoid applications consuming all their credit in a very limited period of time, monopolizing a resource, and creating service interruption for other applications.

Once an application reaches the quota for a given resource, the resource is depleted and will not be available to the application until the quota is replenished. Once a resource is depleted, subsequent requests to that resource will generate an error or an exception. Resources such as CPU time and incoming or outgoing bandwidth will return an "HTTP 403" error page to users; all the other resources and services will generate an exception that can be trapped in code to provide more useful feedback to users.

Resources and services quotas are organized into free default quotas and billing-enabled default quotas. For these two categories, a daily limit and a maximum rate are defined. A detailed explanation of how quotas work, their limits, and the amount that is charged to the user can be found on the AppEngine Website at the following Internet address: http://code.google.com/appengine/docs/quotas.html.

9.2.4 Observations

AppEngine, a framework for developing scalable Web applications, leverages Google's infrastructure. The core components of the service are a scalable and sandboxed runtime environment for executing applications and a collection of services that implement most of the common features required for Web development and that help developers build applications that are easy to scale. One of the characteristic elements of AppEngine is the use of simple interfaces that allow applications to perform specific operations that are optimized and designed to scale. Building on top of these blocks, developers can build applications and let AppEngine scale them out when needed.

With respect to the traditional approach to Web development, the implementation of rich and powerful applications requires a change of perspective and more effort. Developers have to become familiar with the capabilities of AppEngine and implement the required features in a way that conforms with the AppEngine application model.

10.1 Scientific applications

Scientific applications are a sector that is increasingly using cloud computing systems and technologies. The immediate benefit seen by researchers and academics is the potentially infinite availability of computing resources and storage at sustainable prices compared to a complete in-house deployment. Cloud computing systems meet the needs of different types of applications in the scientific domain: high-performance computing (HPC) applications, high-throughput computing (HTC) applications, and data-intensive applications. The opportunity to use cloud resources is even more appealing because minimal changes need to be made to existing applications in order to leverage cloud resources.

The most relevant option is IaaS solutions, which offer the optimal environment for running bag-of-tasks applications and workflows. Virtual machine instances are opportunely customized to host the required software stack for running such applications and coordinated together with distributed computing middleware capable of interacting with cloud-based infrastructures. PaaS solutions have been considered as well. They allow scientists to explore new programming models for tackling computationally challenging problems. Applications have been redesigned and implemented on top of cloud programming application models and platforms to leverage their unique capabilities. For instance, the MapReduce programming model provides scientists with a very simple and effective model for building applications that need to process large datasets. Therefore it has been widely used to develop data-intensive scientific applications. Problems that require a higher degree of flexibility in terms of structuring of their computation model can leverage platforms such as Aneka, which supports MapReduce and other programming models. We now discuss some interesting case studies in which Aneka has been used.

10.1.1 Healthcare: ECG analysis in the cloud

Healthcare is a domain in which computer technology has found several and diverse applications: from supporting the business functions to assisting scientists in developing solutions to cure diseases.

An important application is the use of cloud technologies to support doctors in providing more effective diagnostic processes. In particular, here we discuss electrocardiogram (ECG) data analysis on the cloud [160].

The capillary development of Internet connectivity and its accessibility from any device at any time has made cloud technologies an attractive option for developing health-monitoring systems. ECG data analysis and monitoring constitute a case that naturally fits into this scenario. ECG is the electrical manifestation of the contractile activity of the heart's myocardium. This activity produces a specific waveform that is repeated over time and that represents the heartbeat. The analysis of the shape of the ECG waveform is used to identify arrhythmias and is the most common way to detect heart disease. Cloud computing technologies allow the remote monitoring of a patient's heartbeat data, data analysis in minimal time, and the notification of first-aid personnel and doctors should these data reveal potentially dangerous conditions. This way a patient at risk can be constantly monitored without going to a hospital for ECG analysis. At the same time, doctors and first-aid personnel can instantly be notified of cases that require their attention.

An illustration of the infrastructure and model for supporting remote ECG monitoring is shown in Figure 10.1. Wearable computing devices equipped with ECG sensors constantly monitor the patient's heartbeat. Such information is transmitted to the patient's mobile device, which will eventually forward it to the cloud-hosted Web service for analysis. The Web service forms the front-end of a platform that is entirely hosted in the cloud and that leverages the three layers of the cloud computing stack: SaaS, PaaS, and IaaS. The Web service constitute the SaaS application that will store ECG data in the Amazon S3 service and issue a processing request to the scalable cloud platform. The runtime platform is composed of a dynamically sizable number of instances running the workflow engine and Aneka. The number of workflow engine instances is controlled according to the number of requests in the queue of each instance, while Aneka controls the number of EC2 instances used to execute the single tasks defined by the workflow engine for a single ECG processing job. Each of these jobs consists of a set of operations involving the extraction of the waveform from the heartbeat data and the comparison of the waveform with a reference waveform to detect anomalies. If anomalies are found, doctors and first-aid personnel can be notified to act on a specific patient.

Even though remote ECG monitoring does not necessarily require cloud technologies, cloud computing introduces opportunities that would be otherwise hardly achievable. The first advantage is the elasticity of the cloud infrastructure that can grow and shrink according to the requests served. As a result, doctors and hospitals do not have to invest in large computing infrastructures designed after capacity planning, thus making more effective use of budgets. The second advantage is ubiquity. Cloud computing technologies have now become easily accessible and promise to deliver systems with minimum or no downtime. Computing systems hosted in the cloud are accessible from any Internet device through simple interfaces (such as SOAP and REST-based Web services). This makes these systems not only ubiquitous, but they can also be easily integrated with other systems maintained on the hospital's premises. Finally, cost savings constitute another reason for the use of cloud technology in healthcare. Cloud services are priced on a pay-per-use basis and with volume prices for large numbers of service requests. These two models provide a set of flexible options that can be used to price the service, thus actually charging costs based on effective use rather than capital costs.

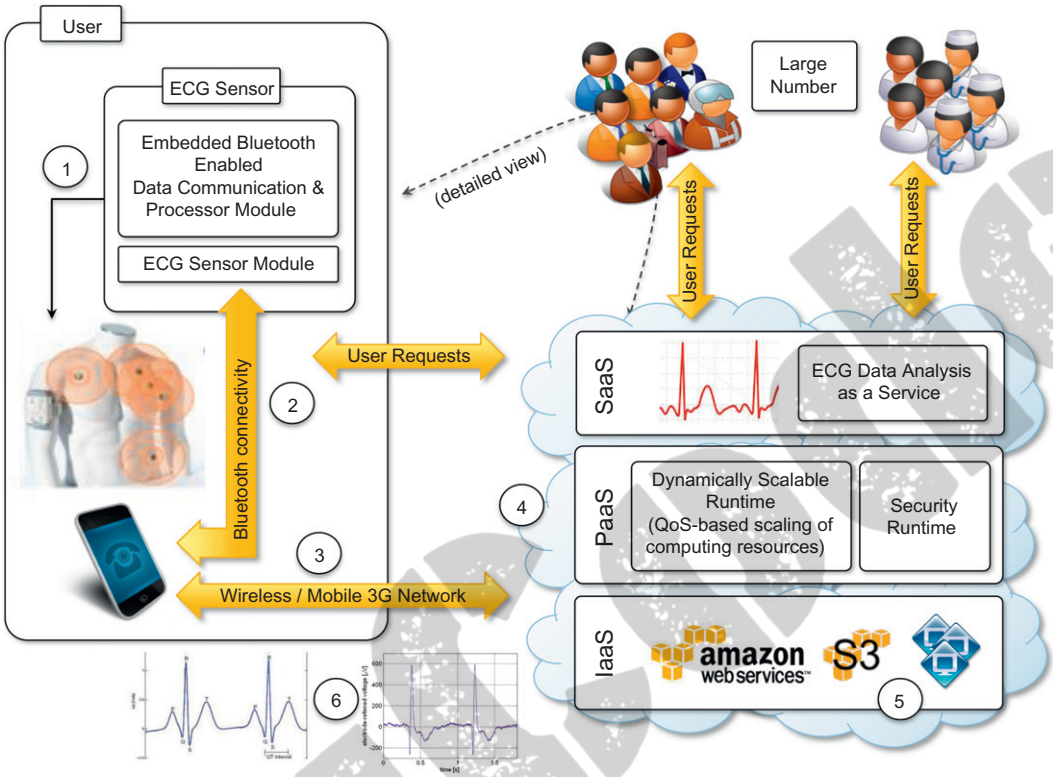

FIGURE 10.1

An online health monitoring system hosted in the cloud.

10.1.2 Biology: protein structure prediction

Applications in biology often require high computing capabilities and often operate on large datasets that cause extensive I/O operations. Because of these requirements, biology applications have often made extensive use of supercomputing and cluster computing infrastructures. Similar capabilities can be leveraged on demand using cloud computing technologies in a more dynamic fashion, thus opening new opportunities for bioinformatics applications.

Protein structure prediction is a computationally intensive task that is fundamental to different types of research in the life sciences. Among these is the design of new drugs for the treatment of diseases. The geometric structure of a protein cannot be directly inferred from the sequence of genes that compose its structure, but it is the result of complex computations aimed at identifying the structure that minimizes the required energy. This task requires the investigation of a space with a massive number of states, consequently creating a large number of computations for each of these states. The computational power required for protein structure prediction can now be acquired on demand, without owning a cluster or navigating the bureaucracy to get access to

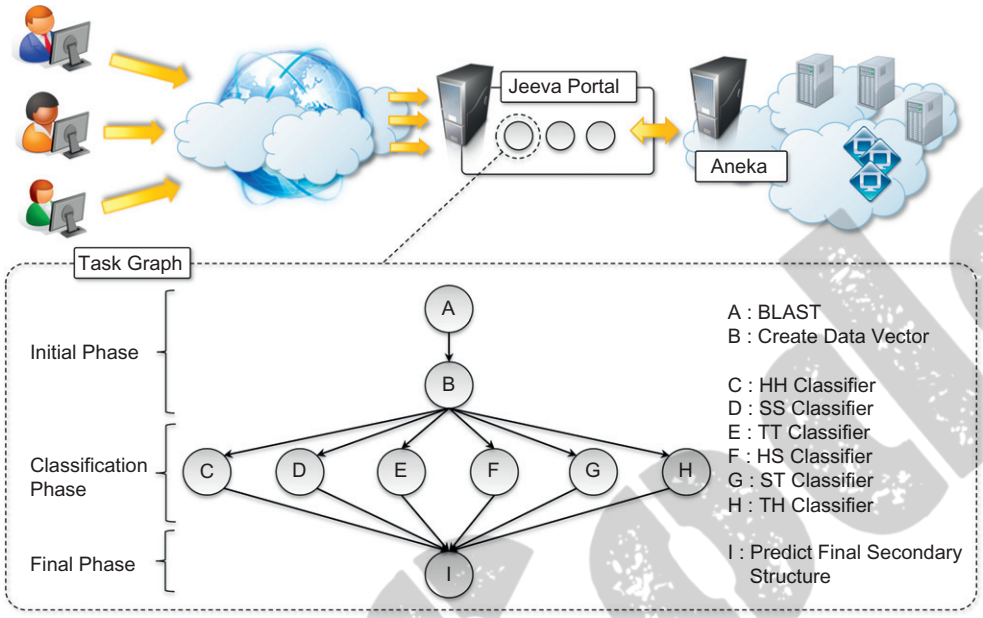

FIGURE 10.2

Architecture and overview of the Jeeva Portal.

parallel and distributed computing facilities. Cloud computing grants access to such capacity on a pay-per-use basis.

One project that investigates the use of cloud technologies for protein structure prediction is *Jeeva* [161]—an integrated Web portal that enables scientists to offload the prediction task to a computing cloud based on Aneka (see Figure 10.2). The prediction task uses machine learning techniques (support vector machines) for determining the secondary structure of proteins. These techniques translate the problem into one of pattern recognition, where a sequence has to be classified into one of three possible classes (E, H, and C). A popular implementation based on support vector machines divides the pattern recognition problem into three phases: *initialization*, *classification*, and a *final phase*. Even though these three phases have to be executed in sequence, it is possible to take advantage of parallel execution in the classification phase, where multiple classifiers are executed concurrently. This creates the opportunity to sensibly reduce the computational time of the prediction. The prediction algorithm is then translated into a task graph that is submitted to Aneka. Once the task is completed, the middleware makes the results available for visualization through the portal.

The advantage of using cloud technologies (i.e., Aneka as scalable cloud middleware) versus conventional grid infrastructures is the capability to leverage a scalable computing infrastructure that can be grown and shrunk on demand. This concept is distinctive of cloud technologies and constitutes a strategic advantage when applications are offered and delivered as a service.

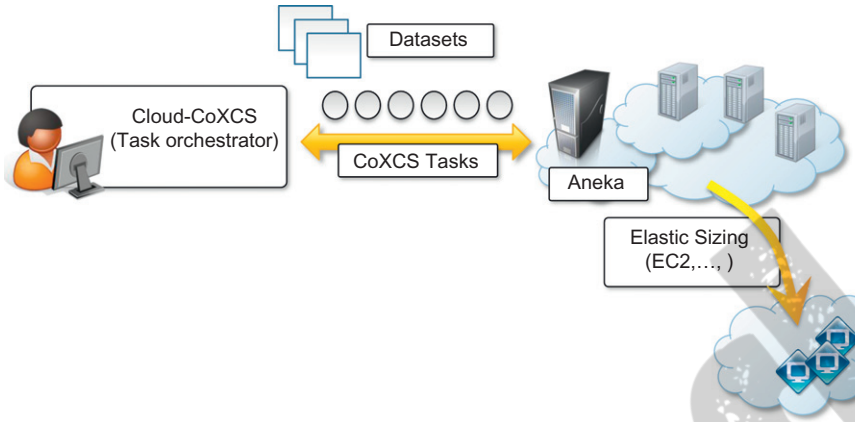

FIGURE 10.3

Cloud-CoXCS: An environment for microarray data processing on the cloud.

10.1.3 Biology: gene expression data analysis for cancer diagnosis

Gene expression profiling is the measurement of the expression levels of thousands of genes at once. It is used to understand the biological processes that are triggered by medical treatment at a cellular level. Together with protein structure prediction, this activity is a fundamental component of drug design, since it allows scientists to identify the effects of a specific treatment.

Another important application of gene expression profiling is cancer diagnosis and treatment. Cancer is a disease characterized by uncontrolled cell growth and proliferation. This behavior occurs because genes regulating the cell growth mutate. This means that all the cancerous cells contain mutated genes. In this context, gene expression profiling is utilized to provide a more accurate classification of tumors. The classification of gene expression data samples into distinct classes is a challenging task. The dimensionality of typical gene expression datasets ranges from several thousands to over tens of thousands of genes. However, only small sample sizes are typically available for analysis.

This problem is often approached with learning classifiers, which generate a population of condition-action rules that guide the classification process. Among these, the *eXtended Classifier System (XCS)* has been successfully utilized for classifying large datasets in the bioinformatics and computer science domains. However, the effectiveness of XCS, when confronted with high dimensional datasets (such as microarray gene expression data sets), has not been explored in detail. A variation of this algorithm, CoXCS [162], has proven to be effective in these conditions. CoXCS divides the entire search space into subdomains and employs the standard XCS algorithm in each of these subdomains. Such a process is computationally intensive but can be easily parallelized because the classifications problems on the subdomains can be solved concurrently. Cloud-CoXCS (see Figure 10.3) is a cloud-based implementation of CoXCS that leverages Aneka to solve the classification problems in parallel and compose their outcomes. The algorithm is controlled by strategies, which define the way the outcomes are composed together and whether the process needs to be iterated.

Because of the dynamic nature of XCS, the number of required compute resources to execute it can vary over time. Therefore, the use of scalable middleware such as Aneka offers a distinctive advantage.

10.1.4 Geoscience: satellite image processing

Geoscience applications collect, produce, and analyze massive amounts of geospatial and nonspatial data. As the technology progresses and our planet becomes more instrumented (i.e., through the deployment of sensors and satellites for monitoring), the volume of data that needs to be processed increases significantly. In particular, the geographic information system (GIS) is a major element of geoscience applications. GIS applications capture, store, manipulate, analyze, manage, and present all types of geographically referenced data. This type of information is now becoming increasingly relevant to a wide variety of application domains: from advanced farming to civil security and natural resources management. As a result, a considerable amount of geo-referenced data is ingested into computer systems for further processing and analysis. Cloud computing is an attractive option for executing these demanding tasks and extracting meaningful information to support decision makers.

Satellite remote sensing generates hundreds of gigabytes of raw images that need to be further processed to become the basis of several different GIS products. This process requires both I/O and compute-intensive tasks. Large images need to be moved from a ground station's local storage to compute facilities, where several transformations and corrections are applied. Cloud computing provides the appropriate infrastructure to support such application scenarios. A cloud-based implementation of such a workflow has been developed by the Department of Space, Government of India [163]. The system shown in Figure 10.4 integrates several technologies across the entire computing stack. A SaaS application provides a collection of services for such tasks as geocode generation and data visualization. At the PaaS level, Aneka controls the importing of data into the virtualized infrastructure and the execution of image-processing tasks that produce the desired outcome from raw satellite images. The platform leverages a Xen private cloud and the Aneka technology to dynamically provision the required resources (i.e., grow or shrink) on demand.

The project demonstrates how cloud computing technologies can be effectively employed to offload local computing facilities from excessive workloads and leverage more elastic computing infrastructures.

10.2 Business and consumer applications

The business and consumer sector is the one that probably benefits the most from cloud computing technologies. On one hand, the opportunity to transform capital costs into operational costs makes clouds an attractive option for all enterprises that are IT-centric. On the other hand, the sense of ubiquity that the cloud offers for accessing data and services makes it interesting for end users as well. Moreover, the elastic nature of cloud technologies does not require huge up-front investments, thus allowing new ideas to be quickly translated into products and services that can comfortably grow with the demand. The combination of all these elements has made cloud computing the

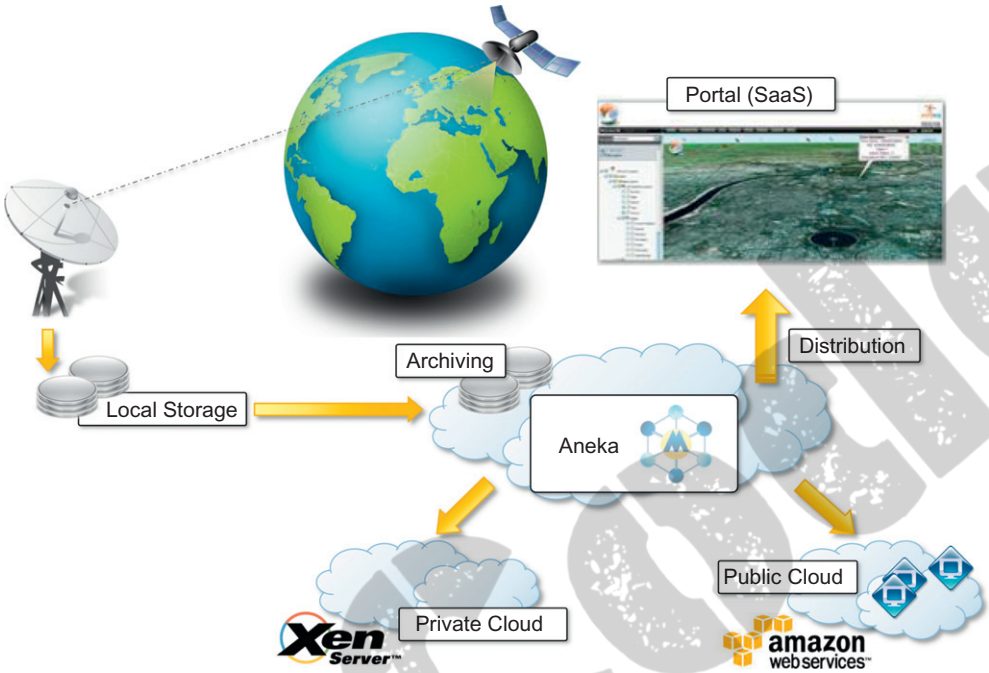

FIGURE 10.4

A cloud environment for satellite data processing.

preferred technology for a wide range of applications, from CRM and ERP systems to productivity and social-networking applications.

10.2.1 CRM and ERP

Customer relationship management (CRM) and *enterprise resource planning (ERP)* applications are market segments that are flourishing in the cloud, with CRM applications the more mature of the two. Cloud CRM applications constitute a great opportunity for small enterprises and start-ups to have fully functional CRM software without large up-front costs and by paying subscriptions. Moreover, CRM is not an activity that requires specific needs, and it can be easily moved to the cloud. Such a characteristic, together with the possibility of having access to your business and customer data from everywhere and from any device, has fostered the spread of cloud CRM applications. ERP solutions on the cloud are less mature and have to compete with well-established in-house solutions. ERP systems integrate several aspects of an enterprise: finance and accounting, human resources, manufacturing, supply chain management, project management, and CRM. Their goal is to provide a uniform view and access to all operations that need to be performed to sustain a complex organization. Because of the organizations that they target, the transition to cloud-based models is more difficult: he cost advantage over the long term might not be clear, and the switch to

the cloud could be difficult if organizations already have large ERP installations. For this reason cloud ERP solutions are less popular than CRM solutions at this time.

10.2.1.1 Salesforce.com

Salesforce.com is probably the most popular and developed CRM solution available today. As of today more than 100,000 customers have chosen Safesforce.com to implement their CRM solutions. The application provides customizable CRM solutions that can be integrated with additional features developed by third parties. Salesforce.com is based on the Force.com cloud development platform. This represents scalable and high-performance middleware executing all the operations of all Salesforce.com applications.

The architecture of the Force.com platform is shown in Figure 10.5. Initially designed to support scalable CRM applications, the platform has evolved to support the entire life cycle of a wider range of cloud applications by implementing a flexible and scalable infrastructure. At the core of the platform resides its metadata architecture, which provides the system with flexibility and scalability. Rather than being built on top of specific components and tables, application core logic and business rules are saved as metadata into the Force.com store. Both application structure and

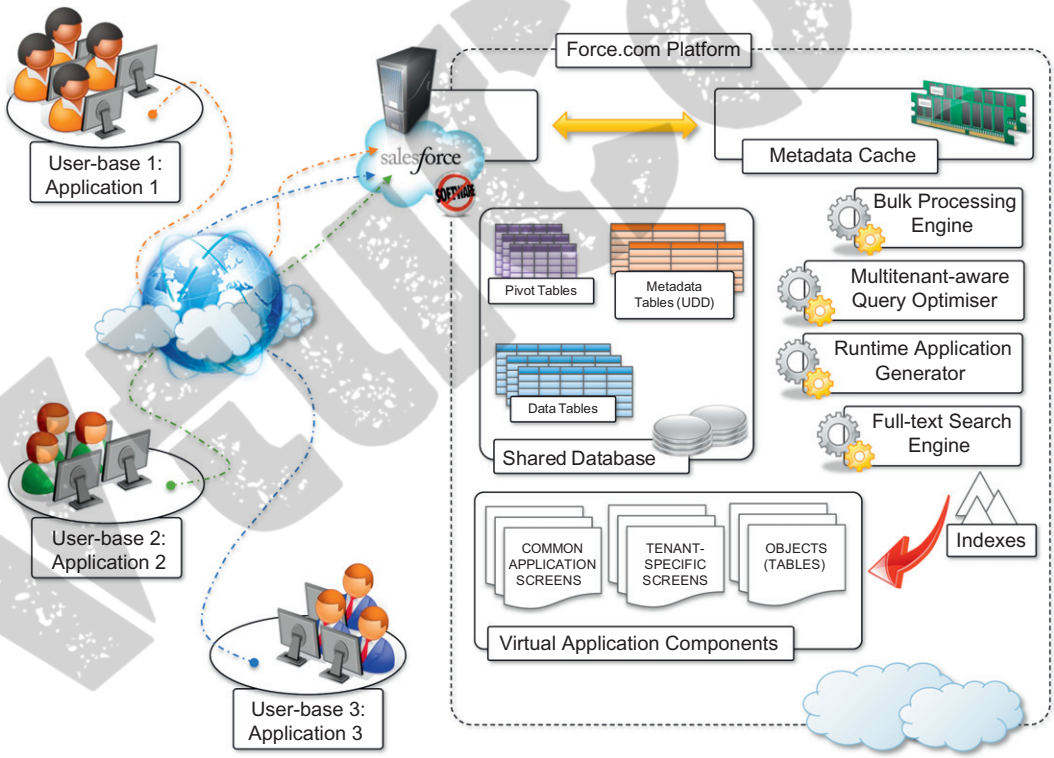

FIGURE 10.5

Salesforce.com and Force.com architecture.

application data are stored in the store. A runtime engine executes application logic by retrieving its metadata and then performing the operations on the data. Although running in isolated containers, different applications logically share the same database structure, and the runtime engine executes all of them uniformly. A full-text search engine supports the runtime engine. This allows application users to have an effective user experience despite the large amounts of data that need to be crawled. The search engine maintains its indexing data in a separate store and is constantly updated by background processes triggered by user interaction.

Users can customize their application by leveraging the "native" Force.com application framework or by using programmatic APIs in the most popular programming languages. The application framework allows users to visually define either the data or the core structure of a Force.com application, while the programmatic APIs provide them with a more conventional way for developing applications that relies on Web services to interact with the platform. Customization of application processes and logic can also be implemented by developing scripts in APEX. This is a Java-like language that provides object-oriented and procedural capabilities for defining either scripts executed on demand or triggers. APEX also offers the capability of expressing searches and queries to have complete access to the data managed by the Force.com platform.

10.2.1.2 Microsoft dynamics CRM

Microsoft Dynamics CRM is the solution implemented by Microsoft for customer relationship management. Dynamics CRM is available either for installation on the enterprise's premises or as an online solution priced as a monthly per-user subscription.

The system is completely hosted in Microsoft's datacenters across the world and offers to customers a 99.9% SLA, with bonus credits if the system does not fulfill the agreement. Each CRM instance is deployed on a separate database, and the application provides users with facilities for marketing, sales, and advanced customer relationship management. Dynamics CRM Online features can be accessed either through a Web browser interface or programmatically by means of SOAP and RESTful Web services. This allows Dynamics CRM to be easily integrated with both other Microsoft products and line-of-business applications. Dynamics CRM can be extended by developing plug-ins that allow implementing specific behaviors triggered on the occurrence of given events. Dynamics CRM can also leverage the capability of Windows Azure for the development and integration of new features.

10.2.1.3 NetSuite

NetSuite provides a collection of applications that help customers manage every aspect of the business enterprise. Its offering is divided into three major products: *NetSuite Global ERP*, *NetSuite Global CRM+*, and *NetSuite Global Ecommerce*. Moreover, an all-in-one solution: *NetSuite One World*, integrates all three products together.

The services NetSuite delivers are powered by two large datacenters on the East and West coasts of the United States, connected by redundant links. This allows NetSuite to guarantee 99.5% uptime to its customers. Besides the prepackaged solutions, NetSuite also provides an infrastructure and a development environment for implementing customized applications. The *NetSuite Business Operating System (NS-BOS)* is a complete stack of technologies for building SaaS business applications that leverage the capabilities of NetSuite products. On top of the SaaS infrastructure, the NetSuite Business Suite components offer accounting, ERP, CRM, and ecommerce capabilities.

An online development environment, *SuiteFlex*, allows integrating such capabilities into new Web applications, which are then packaged for distribution by *SuiteBundler*. The entire infrastructure is hosted in the NetSuite datacenters, which provide warranties regarding application uptime and availability.

10.2.2 Productivity

Productivity applications replicate in the cloud some of the most common tasks that we are used to performing on our desktop: from document storage to office automation and complete desktop environments hosted in the cloud.

10.2.2.1 Dropbox and iCloud

One of the core features of cloud computing is availability anywhere, at any time, and from any Internet-connected device. Therefore, document storage constitutes a natural application for such technology. Online storage solutions preceded cloud computing, but they never became popular. With the development of cloud technologies, online storage solutions have turned into SaaS applications and become more usable as well as more advanced and accessible.

Perhaps the most popular solution for online document storage is *Dropbox*, an online application that allows users to synchronize any file across any platform and any device in a seamless manner (see Figure 10.6). Dropbox provides users with a free amount of storage that is accessible through the abstraction of a folder. Users can either access their Dropbox folder through a browser or by downloading and installing a Dropbox client, which provides access to the online storage by means of a special folder. All the modifications into this folder are silently synched so that changes are notified to all the local instances of the Dropbox folder across all the devices. The key

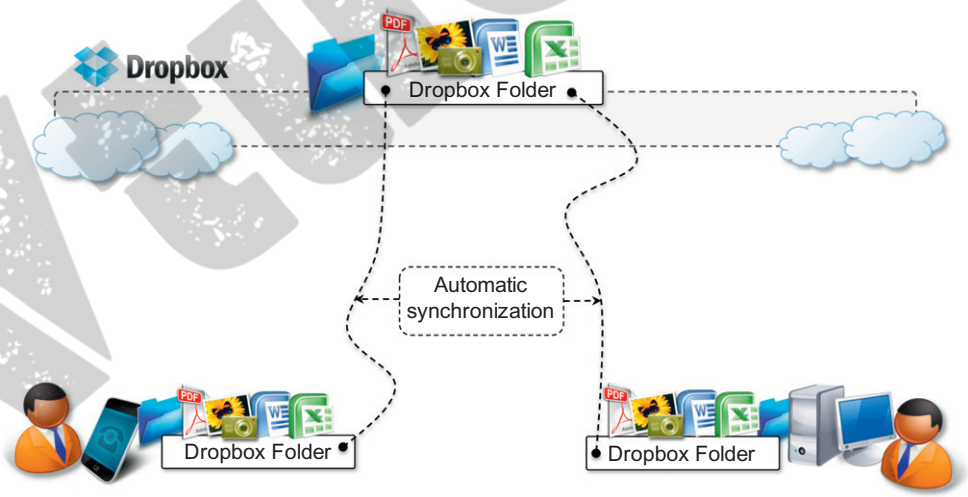

FIGURE 10.6

Dropbox usage scenario.

advantage of Dropbox is its availability on different platforms (Windows, Mac, Linux, and mobile) and the capability to work seamlessly and transparently across all of them.

Another interesting application in this area is *iCloud*, a cloud-based document-sharing application provided by Apple to synchronize iOS-based devices in a completely transparent manner. Unlike Dropbox, which provides synchronization through the abstraction of a local folder, iCloud has been designed to be completely transparent once it has been set up. Documents, photos, and videos are automatically synched as changes are made, without any explicit operation. This allows the system to efficiently automate common operations without any human intervention: taking a picture with your iPhone and having it automatically available in iPhoto on your Mac at home; editing a document on the iMac at home and having the changes updated in your iPad. Unfortunately, this capability is limited to iOS devices, and currently there are no plans to provide iCloud with a Web-based interface that would make user content accessible from even unsupported platforms.

There are other solutions for online document sharing, such as *Windows Live*, *Amazon Cloud Drive*, and *CloudMe*, that are popular and that we did not cover. These solutions offer more or less the same capabilities of those we've discussed, with different levels of integration between platform and devices.

10.2.2.2 Google docs

Google Docs is a SaaS application that delivers the basic office automation capabilities with support for collaborative editing over the Web. The application is executed on top of the Google distributed computing infrastructure, which allows the system to dynamically scale according to the number of users using the service.

Google Docs allows users to create and edit text documents, spreadsheets, presentations, forms, and drawings. It aims to replace desktop products such as Microsoft Office and OpenOffice and provide similar interface and functionality as a cloud service. It supports collaborative editing over the Web for most of the applications included in the suite. This eliminates tedious emailing and synchronization tasks when documents need to be edited by multiple users. By being stored in the Google infrastructure, these documents are always available from anywhere and from any device that is connected to the Internet. Moreover, the suite allows users to work offline if Internet connectivity is not available. Support for various formats such as those that are produced by the most popular desktop office solutions allows users to easily import and move documents in and out of Google Docs, thus eliminating barriers to the use of this application.

Google Docs is a good example of what cloud computing can deliver to end users: ubiquitous access to resources, elasticity, absence of installation and maintenance costs, and delivery of core functionalities as a service.

10.2.2.3 Cloud desktops: EyeOS and XIOS/3

Asynchronous JavaScript and XML (AJAX) technologies have considerably augmented the capabilities that can be implemented in Web applications. This is a fundamental aspect for cloud computing, which delivers a considerable amount of its services through the Web browser. Together with the opportunity to leverage large-scale storage and computation, this technology has made possible the replication of complex desktop environments in the cloud and made them available through the Web browser. These applications, called *cloud desktops*, are rapidly gaining in popularity.

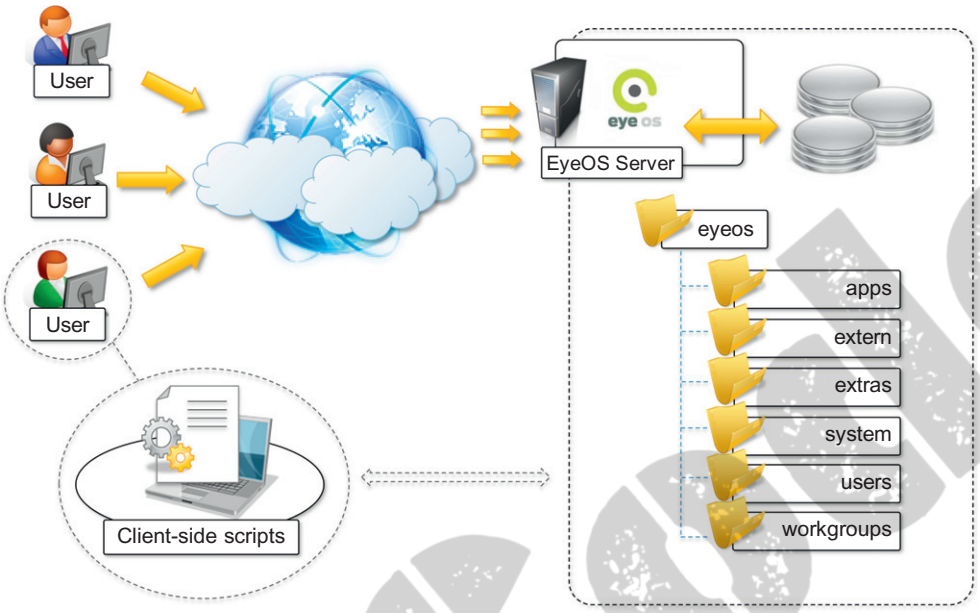

FIGURE 10.7

EyeOS architecture.

EyeOS[1] is one of the most popular Web desktop solutions based on cloud technologies. It replicates the functionalities of a classic desktop environment and comes with pre-installed applications for the most common file and document management tasks (see Figure 10.7). Single users can access the EyeOS desktop environment from anywhere and through any Internet-connected device, whereas organizations can create a private EyeOS Cloud on their premises to virtualize the desktop environment of their employees and centralize their management.

The EyeOS architecture is quite simple: On the server side, the EyeOS application maintains the information about user profiles and their data, and the client side constitutes the access point for users and administrators to interact with the system. EyeOS stores the data about users and applications on the server file system. Once the user has logged in by providing credentials, the desktop environment is rendered in the client's browser by downloading all the JavaScript libraries required to build the user interface and implement the core functionalities of EyeOS. Each application loaded in the environment communicates with the server by using AJAX; this communication model is used to access user data as well as to perform application operations: editing documents, visualizing images, copying and saving files, sending emails, and chatting.

EyeOS also provides APIs for developing new applications and integrating new capabilities into the system. EyeOS applications are server-side components that are defined by at least two files (stored in the *eyeos/apps/appname* directory): *appname.php* and *appname.js*. The first file defines

[1] www.eyeos.org.

and implements all the operations that the application exposes; the JavaScript file contains the code that needs to be loaded in the browser in order to provide user interaction with the application.

Xcerion XML Internet OS/3 (XIOS/3) is another example of a Web desktop environment. The service is delivered as part of the CloudMe application, which is a solution for cloud document storage. The key differentiator of XIOS/3 is its strong leverage of XML, used to implement many of the tasks of the OS: rendering user interfaces, defining application business logics, structuring file system organization, and even application development.The architecture of the OS concentrates most of the functionalities on the client side while implementing server-based functionalities by means of XML Web services. The client side renders the user interface, orchestrates processes, and provides data-binding capabilities on XML data that is exchanged with Web services. The server is responsible for implementing core functions such as transaction management for documents edited in a collaborative mode and core logic of installed applications into the environment. XIOS/3 also provides an environment for developing applications (XIDE), which allows users to quickly develop complex applications by visual tools for the user interface and XML documents for business logic.

XIOS/3 is released as open-source software and implements a marketplace where third parties can easily deploy applications that can be installed on top of the virtual desktop environment. It is possible to develop any type of application and feed it with data accessible through XML Web services: developers have to define the user interface, bind UI components to service calls and operations, and provide the logic on how to process the data. XIDE will package this information into a proper set of XML documents, and the rest will be performed by an XML virtual machine implemented in XIOS.

XIOS/3 is an advanced Web desktop environment that focuses on the integration of services into the environment by means of XML-based services and that simplifies collaboration with peers.

10.2.3 Social networking

Social networking applications have grown considerably in the last few years to become the most active sites on the Web. To sustain their traffic and serve millions of users seamlessly, services such as Twitter and Facebook have leveraged cloud computing technologies. The possibility of continuously adding capacity while systems are running is the most attractive feature for social networks, which constantly increase their user base.

10.2.3.1 Facebook

Facebook is probably the most evident and interesting environment in social networking. With more than 800 million users, it has become one of the largest Websites in the world. To sustain this incredible growth, it has been fundamental that Facebook be capable of continuously adding capacity and developing new scalable technologies and software systems while maintaining high performance to ensure a smooth user experience.

Currently, the social network is backed by two data centers that have been built and optimized to reduce costs and impact on the environment. On top of this highly efficient infrastructure, built and designed out of inexpensive hardware, a completely customized stack of opportunely modified and refined open-source technologies constitutes the back-end of the largest social network. Taken all together, these technologies constitute a powerful platform for developing cloud applications.

This platform primarily supports Facebook itself and offers APIs to integrate third-party applications with Facebook's core infrastructure to deliver additional services such as social games and quizzes created by others.

The reference stack serving Facebook is based on *LAMP* (*Linux*, *Apache*, *MySQL*, and *PHP*). This collection of technologies is accompanied by a collection of other services developed in-house. These services are developed in a variety of languages and implement specific functionalities such as search, news feeds, notifications, and others. While serving page requests, the *social graph* of the user is composed. The social graph identifies a collection of interlinked information that is of relevance for a given user. Most of the user data are served by querying a distributed cluster of MySQL instances, which mostly contain key-value pairs. These data are then cached for faster retrieval. The rest of the relevant information is then composed together using the services mentioned before. These services are located closer to the data and developed in languages that provide better performance than PHP.

The development of services is facilitated by a set of internally developed tools. One of the core elements is *Thrift*. This is a collection of abstractions (and language bindings) that allow cross-language development. Thrift allows services developed in different languages to communicate and exchange data. Bindings for Thrift in different languages take care of data serialization and deserialization, communication, and client and server boilerplate code. This simplifies the work of the developers, who can quickly prototype services and leverage existing ones. Other relevant services and tools are *Scribe*, which aggregates streaming log feeds, and applications for alerting and monitoring.

10.2.4 Media applications

Media applications are a niche that has taken a considerable advantage from leveraging cloud computing technologies. In particular, video-processing operations, such as encoding, transcoding, composition, and rendering, are good candidates for a cloud-based environment. These are computationally intensive tasks that can be easily offloaded to cloud computing infrastructures.

10.2.4.1 Animoto

Animoto[2] is perhaps the most popular example of media applications on the cloud. The Website provides users with a very straightforward interface for quickly creating videos out of images, music, and video fragments submitted by users. Users select a specific theme for a video, upload the photos and videos and order them in the sequence they want to appear, select the song for the music, and render the video. The process is executed in the background and the user is notified via email once the video is rendered.

The core value of Animoto is the ability to quickly create videos with stunning effects without user intervention. A proprietary artificial intelligence (AI) engine, which selects the animation and transition effects according to pictures and music, drives the rendering operation. Users only have to define the storyboard by organizing pictures and videos into the desired sequence. If users don't like the result, the video can be rendered again and the engine will select a different composition, thus producing a different outcome every time. The service allows users to create 30-second videos

[2] www.animoto.com.

for free. By paying a monthly or a yearly subscription it is possible to produce videos of any length and to choose among a wider range of templates.

The infrastructure supporting Animoto is complex and is composed of different systems that all need to scale (see Figure 10.8). The core function is implemented on top of the Amazon Web Services infrastructure. In particular, it uses Amazon EC2 for the Web front-end and the worker nodes; Amazon S3 for the storage of pictures, music, and videos; and Amazon SQS for connecting all the components. The system's auto-scaling capabilities are managed by Rightscale, which monitors the load and controls the creation of new worker instances as well as their reclaim. Front-end nodes collect the components required to make the video and store them in S3. Once the storyboard of the video is completed, a video-rendering request is entered into a SQS queue. Worker nodes pick up rendering requests and perform the rendering. When the process is completed, another message is entered into a different SQS queue and another request is served. This last queue is cleared routinely and users are notified about the completion. The life of EC2 instances is controlled by Rightscale, which constantly monitors the load and the performance of the system and decides whether it is necessary to grow or shrink.

The architecture of the system has proven to be very scalable and reliable by using up to 4,000 servers on EC2 in peak times without dropping requests but simply causing acceptable temporary delays for the rendering process.

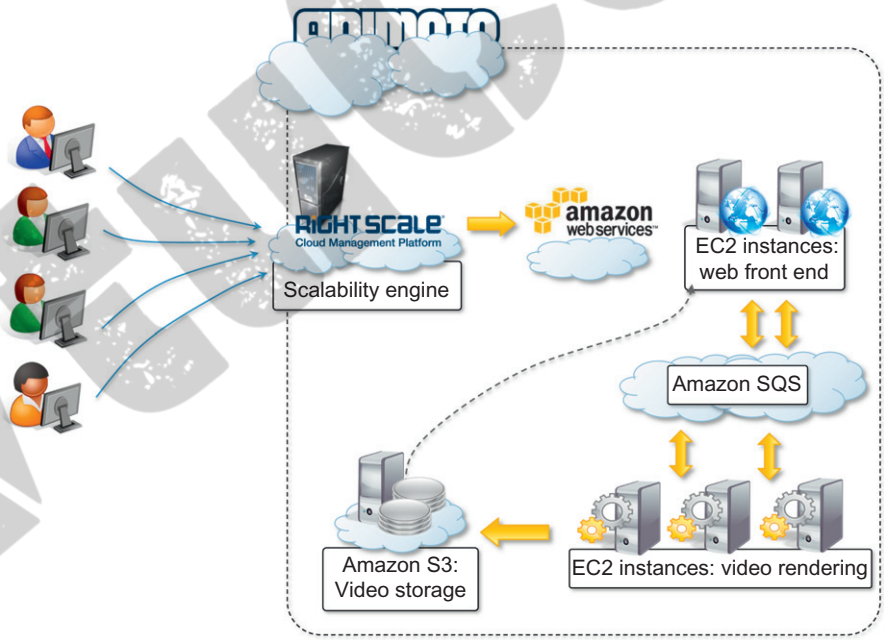

FIGURE 10.8

Animoto reference architecture.

10.2.4.2 Maya rendering with Aneka

Interesting applications of media processing are found in the engineering disciplines and the movie production industry. Operations such as rendering of models are now an integral part of the design workflow, which has become computationally demanding. The visualization of mechanical models is not only used at the end of the design process, it is iteratively used to improve the design. It is then fundamental to perform such tasks as fast as possible. Cloud computing provides engineers with the necessary computing power to make this happen.

A private cloud solution for rendering train designs has been implemented by the engineering department of GoFront group, a division of China Southern Railway (see Figure 10.9). The department is responsible for designing models of high-speed electric locomotives, metro cars, urban transportation vehicles, and motor trains. The design process for prototypes requires high-quality, three-dimensional (3D) images. The analysis of these images can help engineers identify problems and correct their design. Three-dimensional rendering tasks take considerable amounts of time, especially in the case of huge numbers of frames, but it is critical for the department to reduce the time spent in these iterations. This goal has been achieved by leveraging cloud computing technologies, which turned the network of desktops in the department into a desktop cloud managed by Aneka.

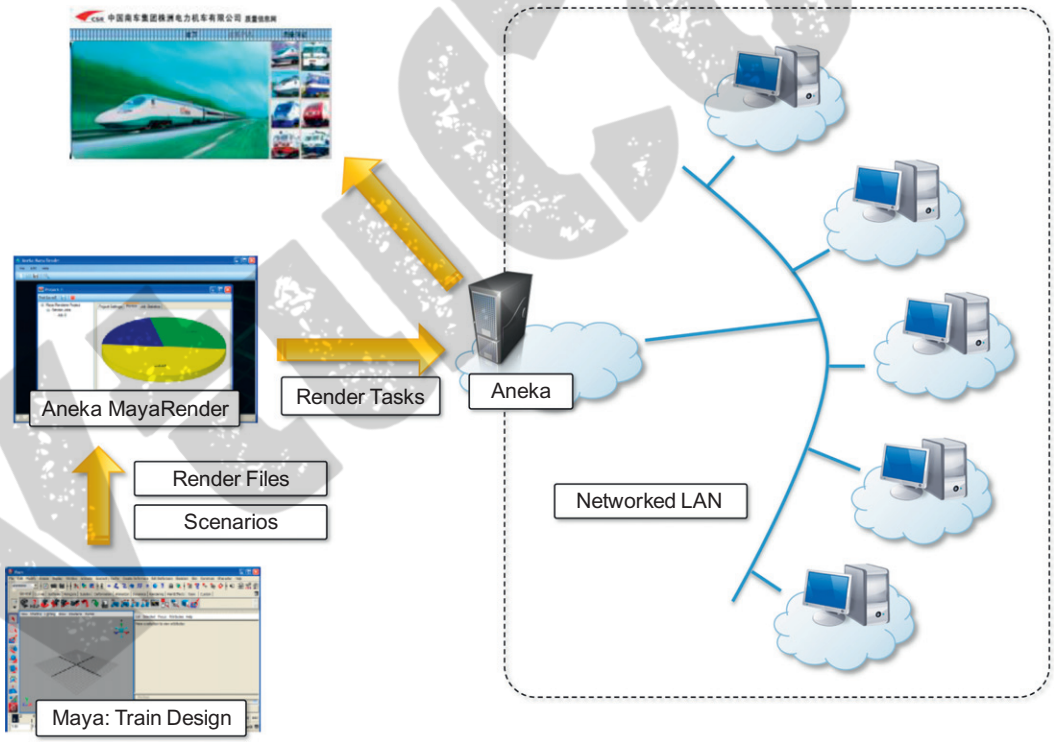

FIGURE 10.9

3D rendering on private clouds.

The implemented system includes a specialized client interface that can be used by GoFront engineers to enter all the details of the rendering process (the number of frames, the number of cameras, and other parameters). The application is used to submit the rendering tasks to the Aneka Cloud, which distributes the load across all the available machines. Every rendering task triggers the execution of the local Maya batch renderer and collects the result of the execution. The renders are then retrieved and put all together for visualization.

By turning the local network into a private cloud, the resources of which can be used off-peak (i.e., at night, when desktops are not utilized), it has been possible for GoFront to sensibly reduce the time spent in the rendering process from days to hours.

10.2.4.3 Video encoding on the cloud: Encoding.com

Video encoding and transcoding are operations that can greatly benefit from using cloud technologies: They are computationally intensive and potentially require considerable amounts of storage. Moreover, with the continuous improvement of mobile devices as well as the diffusion of the Internet, requests for video content have significantly increased. The variety of devices with video playback capabilities has led to an explosion of formats through which a video can be delivered. Software and hardware for video encoding and transcoding often have prohibitive costs or are not flexible enough to support conversion from any format to any format. Cloud technologies present an opportunity for turning these tedious and often demanding tasks into services that can be easily integrated into a variety of workflows or made available to everyone according to their needs.

Encoding.com is a software solution that offers video-transcoding services on demand and leverages cloud technology to provide both the horsepower required for video conversion and the storage for staging videos. The service integrates with both Amazon Web Services technologies (*EC2*, *S3*, and *CloudFront*) and Rackspace (*Cloud Servers*, *Cloud Files*, and *Limelight CDN* access). Users can access the services through a variety of interfaces: the Encoding.com Website, Web service XML APIs, desktop applications, and watched folders. To use the service, users have to specify the location of the video to transcode, the destination format, and the target location of the video. Encoding.com also offers other video-editing operations such as the insertion of thumbnails, watermarks, or logos. Moreover, it extends its capabilities to audio and image conversion.

The service provides various pricing options: monthly fee, pay-as-you-go (by batches), and special prices for high volumes. Encoding.com now has more than 2,000 customers and has already processed more than 10 million videos.

10.2.5 Multiplayer online gaming

Online multiplayer gaming attracts millions of gamers around the world who share a common experience by playing together in a virtual environment that extends beyond the boundaries of a normal LAN. Online games support hundreds of players in the same session, made possible by the specific architecture used to forward interactions, which is based on game log processing. Players update the game server hosting the game session, and the server integrates all the updates into a log that is made available to all the players through a TCP port. The client software used for the game connects to the log port and, by reading the log, updates the local user interface with the actions of other players.

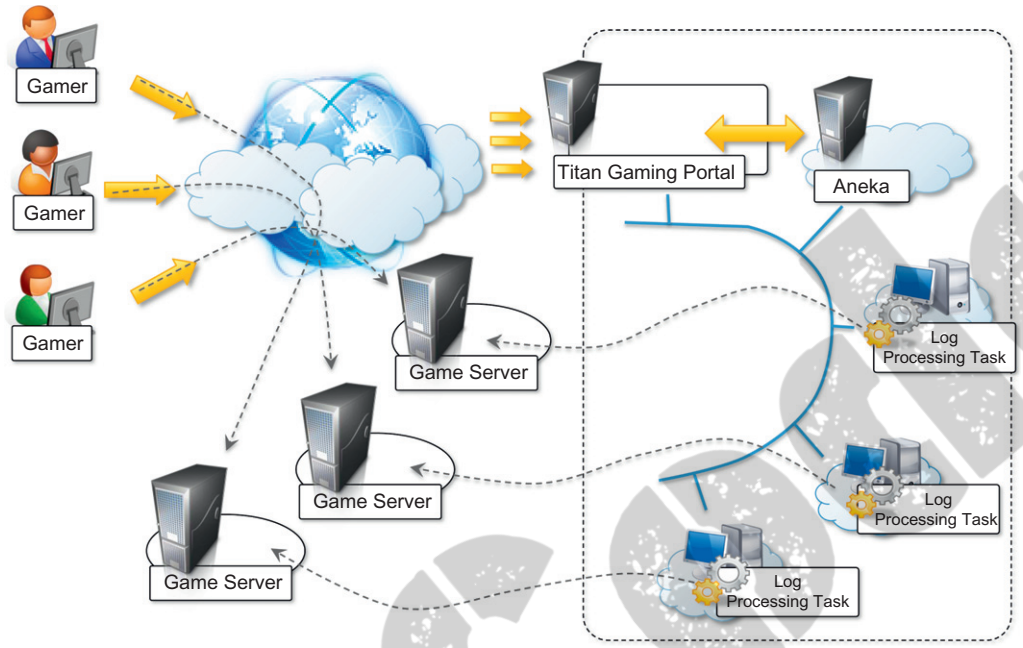

FIGURE 10.10

Scalable processing of logs for network games.

Game log processing is also utilized to build statistics on players and rank them. These features constitute the additional value of online gaming portals that attract more and more gamers. The processing of game logs is a potentially compute-intensive operation that strongly depends on the number of players online and the number of games monitored. Moreover, gaming portals are Web applications and therefore might suffer from the spiky behavior of users that can randomly generate large amounts of volatile workloads that do not justify capacity planning.

The use of cloud computing technologies can provide the required elasticity for seamlessly processing these workloads and scale as required when the number of users increases. A prototypal implementation of cloud-based game log processing has been implemented by Titan Inc. (now Xfire), a company based in California that extended its gaming portal for offload game log processing to an Aneka Cloud. The prototype (shown in Figure 10.10) uses a private cloud deployment that allowed Titan Inc. to process concurrently multiple logs and sustain a larger number of users.