SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – I – Cloud Computing – SBS1207

UNIT I

The vision of cloud computing - The cloud computing reference model - Characteristics and benefits - Challenges ahead - Historical developments - Distributed systems -Virtualization - Building cloud computing environments - Application development - Infrastructure and system development - Computing platforms and technologies.Parallel vs. distributed computing - Elements of parallel computing - Hardware architectures for parallel processing Approaches to parallel programming - Laws of caution.

Cloud computing

Definition:

The cloud is a large group of interconnected computers. These computers can be personal computers or network servers; they can be public or private.

Cloud computing is a technology that uses the internet and central remote servers to maintain data and applications.

Eg: Yahoo email or Gmail etc

THE CLOUD COMPUTING REFERENCE MODEL

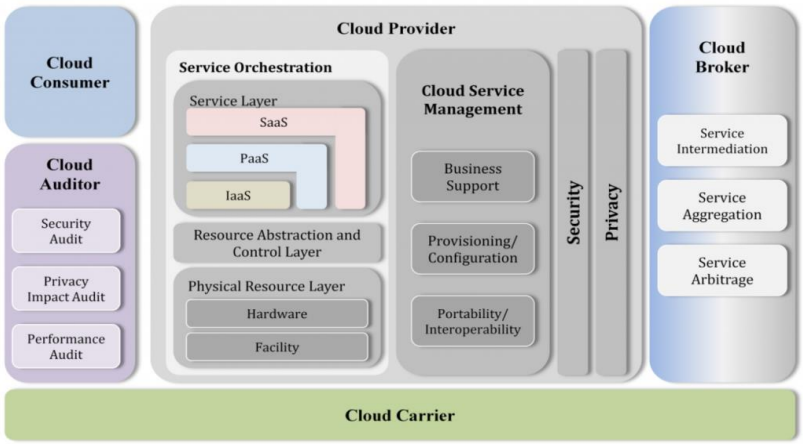


Fig 1: The Cloud Computing Reference Model

The NIST cloud computing reference architecture Fig 1 defines five major actors:

1. Cloud consumer
2. Cloud provider
3. Cloud carrier
4. Cloud auditor and
5. Cloud broker.

Each actor is an entity (a person or an organization) that participates in a transaction or process and/or performs tasks in cloud computing.

Actors in Cloud Computing

Table 1: Actors in Cloud Computing

SNO	Actor	Definition
1	Cloud Consumer	A person or organization that maintains a business relationship with, and uses service from, Cloud Providers.
2	Cloud Provider	A person, organization, or entity responsible for making a service available to interested parties.
3	Cloud Auditor	A party that can conduct independent assessment of cloud services, information system operations, performance and security of the cloud implementation.
4	Cloud Broker	An entity that manages the use, performance and delivery of cloud services, and negotiates relationships between Cloud Providers and Cloud Consumers.
5	Cloud Carrier	An intermediary that provides connectivity and transport of cloud services from Cloud Providers to Cloud Consumers.

Interactions among the actors:

A cloud consumer may request cloud services from a cloud provider directly or via a cloud broker.

A cloud auditor conducts independent audits and may contact the others to collect necessary information.

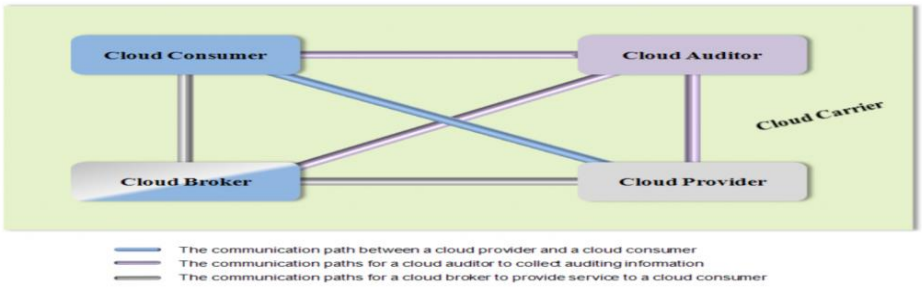


Fig 2: Interaction among Actors

Cloud carriers provide the connectivity and transport of cloud services from cloud providers to cloud consumers.

A cloud provider participates in and arranges for two unique service level agreements (SLAs), one with a cloud carrier (e.g. SLA2) and one with a cloud consumer (e.g. SLA1).

A cloud provider arranges service level agreements (SLAs) with encrypted connections to ensure the cloud services are consumed at a consistent level according to the contractual obligations with the cloud consumers.

In this case, the provider may specify its requirements on capability, flexibility and functionality in SLA2 in order to provide essential requirements in SLA1.

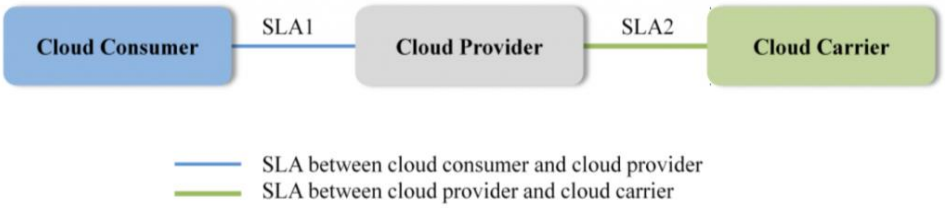


Fig 3 Usage Scenario for Cloud Carriers

A cloud service, a cloud auditor conducts independent assessments of the operation and security of the cloud service implementation. The audit may involve interactions with both the Cloud Consumer and the Cloud Provider.

Figure 5 – Usage Scenario for Cloud Auditor

CHARACTERISTICS OF CLOUD COMPUTING

1. **Cloud computing is user-centric.**

Once users are connected to the cloud, whatever is available (stored) in cloud as documents, messages, images, applications, it becomes users own.
Users can also share it with others.

2. **Cloud computing is task-centric.**

It focusing on the application for users on what users need to done and how the application can do it for users. Eg collage, Animation, Movie in Google Photos

3. **Cloud computing is powerful.**

Connecting hundreds or thousands of computers together in a cloud creates a wealth of computing power impossible with a single desktop PC.

4. **Cloud computing is accessible.**

Data is stored in the cloud; users can instantly retrieve more information from multiple repositories. You're not limited to a single source of data.

5. **Cloud computing is intelligent.**

With all the various data stored on the computers in a cloud, data mining and analysis are necessary to access that information in an intelligent manner.

6. **Cloud computing is programmable**.

Many of the tasks necessary with cloud computing must be automated.

For example, cloud must be replicated on other computers in the cloud. If that one computer goes offline, the cloud's programming automatically redistributes that computer's data to a new computer in the cloud.
Examples of cloud computing applications: Google Docs & Spreadsheets, Google Calendar, Gmail, Picasa.

PROS AND CONS OF CLOUD COMPUTING

Advantages

- Lower-Cost Computers for Users
- Improved Performance
- Lower IT Infrastructure Costs
- Fewer Maintenance Issues
- Lower Software Costs
- Instant Software Updates
- Increased Computing Power
- Unlimited Storage Capacity
- Increased Data Safety
- Improved Compatibility Between Operating Systems
- Improved Document Format Compatibility
- Easier Group Collaboration
- Universal Access to Documents
- Latest Version Availability
- Removes the Tether to Specific Devices

Dis-Advantages

- Requires a Constant Internet Connection
- Doesn’t Work Well with Low-Speed Connections
- Can Be Slow
- Features Might Be Limited
- Stored Data Might Not Be Secure
- Problem will arise If Data loss occurs

HISTORICAL DEVELOPMENTS

Client/Server Computing: Centralized Applications and Storage

In *client/server* model all the software applications, data, and the control resided on huge mainframe computers, known as ***servers***.

If a user wanted to access specific data or run a program, he had to connect to the mainframe, gain appropriate access, and then do his business.

Users connected to the server via a computer terminal, called a workstation or ***client.***

Drawbacks in client /server Model

Even on a mainframe computer, processing power is limited.

Access was not immediate nor could two users access the same data at the same time.
When multiple people are sharing a single computer, you have to wait for your turn.

There isn't always immediate access in a client/server environment.

So the client/server model, while providing similar centralized storage, differed from cloud computing in that it did not have a user-centric focus. It was not a user-enabling environment.

Peer-to-Peer Computing: Sharing Resources

P2P computing defines a network architecture in which each computer has equivalent capabilities and responsibilities.

In the P2P environment, every computer is a client *and* a server; there are no masters and slaves.

P2P enables direct exchange of resources and services.

There is no need for a central server, because any computer can function in that capacity when called on to do so.

P2P was a decentralizing concept. Control is decentralized, with all computers functioning as equals. Content is also dispersed among the various peer computers.

Distributed Computing: Providing More Computing Power

One of the **subsets** of the P2P model.

distributed computing, where idle PCs across a network or Internet are tapped to provide computing power for large, processor-intensive projects.

Collaborative Computing: Working as a Group

Multiple users to work simultaneously on the same computer-based project called collaborative computing.

The goal was to enable multiple users to collaborate on group projects online, in real time.

To collaborate on any project, users must first be able to talk to one another.

Most collaboration systems offer the complete range of audio/video options, for full-featured multiple-user video conferencing.

In addition, users must be able to share files and have multiple users work on the same document simultaneously.

Real-time white boarding is also common, especially in corporate and education environments.

Cloud Computing: The Next Step in Collaboration

With the growth of the Internet, there was no need to limit group collaboration to a single enterprise's network environment.

Users from multiple locations within a corporation, and from multiple organizations, desired to collaborate on projects that crossed company and geographic boundaries.

To do this, projects had to be housed in the "cloud" of the Internet, and accessed from any Internet-enabled location.

DISTRIBUTED SYSTEM

A distributed system contains multiple nodes that are physically separate but linked together using the network.
All the nodes in this system communicate with each other and handle processes in tandem.
Each of these nodes contains a small part of the distributed operating system software.

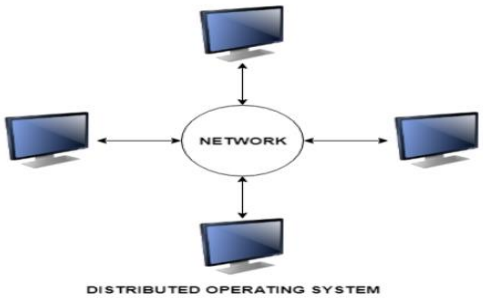


Fig 6: Distributed System

Types of Distributed Systems

The nodes in the distributed systems can be arranged in the form of client/server systems or peer to peer systems. Details about these are as follows:

Client/Server Systems
In client server systems, the client requests a resource and the server provides that resource.
A server may serve multiple clients at the same time while a client is in contact with only one server.
Both the client and server usually communicate via a computer network and so they are a part of distributed systems.

Peer to Peer Systems
The peer to peer systems contains nodes that are equal participants in data sharing.
All the tasks are equally divided between all the nodes. The nodes interact with each other as required as share resources. This is done with the help of a network.

Advantages of Distributed Systems

- can easily share data with other nodes.
- More nodes can easily be added to the distributed system i.e. it can be scaled as required.
- Failure of one node does not lead to the failure of the entire distributed system. Other nodes can still communicate with each other.

- Resources like printers can be shared with multiple nodes rather than being restricted to just one.

Disadvantages of Distributed Systems

- It is difficult to provide adequate security in distributed systems because the nodes as well as the connections need to be secured.
- Some messages and data can be lost in the network while moving from one node to another.
- The database connected to the distributed systems is quite complicated and difficult to handle as compared to a single user system.
- Overloading may occur in the network if all the nodes of the distributed system try to send data at once.

VIRTUALIZATION

Virtualization is a technique, which allows to share single physical instance of an application or resource among multiple organizations or tenants (customers).

Creating a virtual machine over existing operating system and hardware is referred as Hardware Virtualization. Virtual Machines provide an environment that is logically separated from the underlying hardware.

The machine on which the virtual machine is created is known as host machine and virtual machine is referred as a guest machine. This virtual machine is managed by a software or firmware, which is known as hypervisor.

Hypervisor

The hypervisor is a firmware or low-level program that acts as a Virtual Machine Manager.

There are two types of hypervisor:

Type 1 hypervisor executes on bare system.

Eg LynxSecure, RTS Hypervisor, Oracle VM, Sun xVM Server, VirtualLogic VLX.

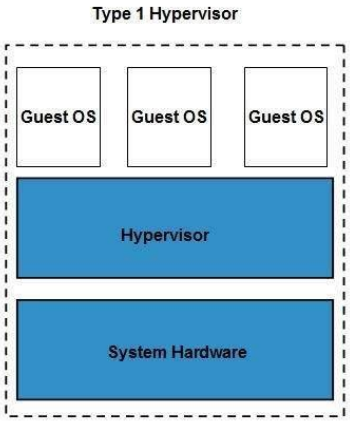


Fig 7: Type 1 Hypervisor

The type1 hypervisor does not have any host operating system because they are installed on a bare system.

Type 2 hypervisor is a software interface that emulates the devices with which a system normally interacts.

Eg Containers, KVM, Microsoft Hyper V, VMWare Fusion, Virtual Server 2005 R2, Windows Virtual PC and VMWare workstation 6.0.

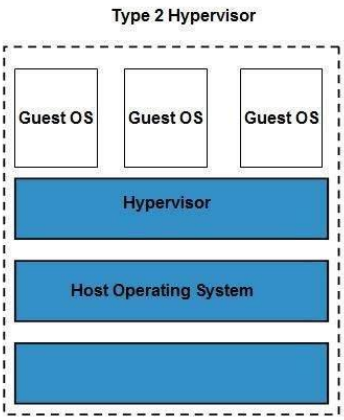


Fig 8: Type 2 Hypervisor

BUILDING CLOUD COMPUTING ENVIRONMENTS

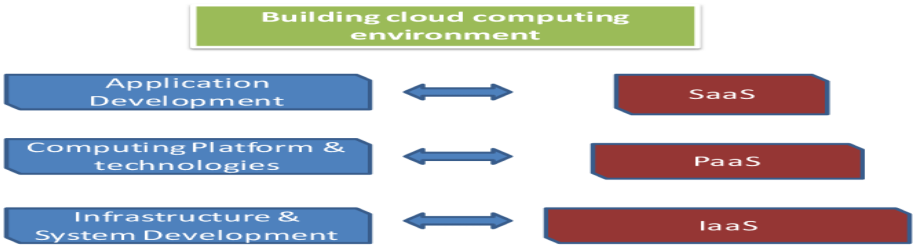


Fig 9: Building Cloud Computing Environment

Cloud Computing Services

The three major Cloud Computing Offerings are

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

SaaS (Software as a Service)

SaaS or software as a service is a software distribution model in which applications are hosted by a vendor or service provider and made available to customers over a network (internet).

PaaS (Platform as a Service)

Platform as a service, is referred as PaaS, it provides a platform and environment to allow developers to build applications and services.

This service is hosted in the cloud and accessed by the users via internet.

IaaS (Infrastructure as a Service)

IaaS (Infrastructure As A Service) is one of the fundamental service model of cloud computing alongside PaaS(Platform as a Service).

It provides access to computing resources in a virtualized environment "the cloud" on internet.

It provides computing infrastructure like virtual server space, network connections, bandwidth, load balancers and IP addresses.

The pool of hardware resource is extracted from multiple servers and networks usually distributed across numerous data centers.

This provides redundancy and reliability to IaaS.

(PaaS) that are available for the **users to build and host an application** are:

Amazon Web Services: AWS offers virtual compute, storage, networking & complete computing stacks.

It is known for its on-demand services namely Elastic Compute Cloud (EC2) and Simple Storage Service (S3).

Google AppEngine: Launched in 2008, it provides applications **(SaaS)** and raw hardware

(IaaS).

App Engine managed infrastructure, provides a development platform to create apps, leveraging Google's infrastructure as a hosting platform.

Microsoft Azure: It is also a scalable runtime environment for web & distributed applications.
it provides additional services such as support for storage (relational data & blobs), networking, caching, content delivery & others.

CLOUD APPLICATION DEVELOPMENT

Cloud application, or **cloud** app, is a **software** program where **cloud**-based and local components work together. This model relies on remote servers for processing logic that is accessed through a web browser with a continual internet connection.
Cloud application servers typically are located in a remote data center operated by a third-party cloud services infrastructure provider. Cloud-based application tasks may encompass email, file storage and sharing, order entry, inventory management, word processing, customer relationship management (CRM), data collection, or financial accounting features.

Benefits of cloud apps

Fast response to business needs. Cloud applications can be updated, tested and deployed quickly, providing enterprises with fast time to market and agility. This speed can lead to culture shifts in business operations.
Simplified operation. Infrastructure management can be outsourced to third-party cloud providers.
Instant scalability. As demand rises or falls, available capacity can be adjusted.
API use. Third-party data sources and storage services can be accessed with an application programming interface (API). Cloud applications can be kept smaller by using APIs to hand data to applications or API-based back-end services for processing or analytics computations, with the results handed back to the cloud application. Vetted APIs impose passive consistency that can speed development and yield predictable results.
Gradual adoption. Refactoring legacy, on-premises applications to a cloud architecture in steps, allows components to be implemented on a gradual basis.
Reduced costs. The size and scale of data centers run by major cloud infrastructure and service providers, along with competition among providers, has led to lower prices. Cloud-based applications can be less expensive to operate and maintain than equivalents on-premises installation.
Improved data sharing and security. Data stored on cloud services is instantly available to authorized users. Due to their massive scale, cloud providers can hire world-class security experts

and implement infrastructure security measures that typically only large enterprises can obtain. Centralized data managed by IT operations personnel is more easily backed up on a regular schedule and restored should disaster recovery become necessary.

How cloud apps work

Data is stored and compute cycles occur in a remote data center typically operated by a third-party company. A back end ensures uptime, security and integration and supports multiple access methods.

Cloud applications provide quick responsiveness and don't need to permanently reside on the local device. They can function offline, but can be updated online.

While under constant control, cloud applications don't always consume storage space on a computer or communications device. Assuming a reasonably fast internet connection, a well-written cloud application offers all the interactivity of a desktop application, along with the portability of a web application.

CLOUD COMPUTING PLATFORMS AND TECHNOLOGIES

Fig 10: Cloud Computing Top Platforms

Amazon Web Services (AWS) –

- AWS provides different wide-ranging clouds **IaaS services**, which ranges from virtual compute, storage, and networking to complete computing stacks.
- AWS is well known for its storage and compute on demand services, named as Elastic Compute Cloud (EC2) and Simple Storage Service (S3).
- EC2 offers customizable virtual hardware to the end user which can be utilized as the base infrastructure for deploying computing systems on the cloud.
- S3 is well ordered into buckets which contain objects that are stored in binary form and can be grown with attributes. End users can store objects of any size, from basic file to full disk images and have them retrieval from anywhere.
- In addition, EC2 and S3, a wide range of services can be leveraged to build virtual computing systems including: networking support, caching system, DNS, database support, and others.

Google AppEngine

- Google AppEngine is a scalable runtime environment frequently dedicated to executing web applications.
- These utilize the benefits of the large computing infrastructure of Google to dynamically scale as per the demand.
- AppEngine offers both a secure execution environment and a collection of which simplifies the development of scalable and high-performance Web applications.
- These **services include**: in-memory caching, scalable data store, job queues, messaging, and corn tasks.
- Currently, the supported programming languages are Python, Java, and Go.

Microsoft Azure–

- Microsoft Azure is a Cloud operating system and a platform in which users can develop the applications in the cloud.
- Azure provides a set of services that support storage, networking, caching, content delivery, and others.

Hadoop

- Apache Hadoop is an open source framework that is appropriate for processing large data sets on commodity hardware.
- Hadoop is an implementation of MapReduce, an application programming model which is developed by Google.
- This model provides two fundamental operations for data processing: map and reduce.

Force.com and Salesforce.com –

- Force.com is a Cloud computing platform at which users can develop social enterprise applications.
- The platform is the basis of SalesForce.com – a Software-as-a-Service solution for customer relationship management.
- Force.com allows creating applications by composing ready-to-use blocks: a complete set of components supporting all the activities of an enterprise are available.
- From the design of the data layout to the definition of business rules and user interface is provided by Force.com as a support.
- This platform is completely hostel in the Cloud, and provides complete access to its functionalities, and those implemented in the hosted applications through Web services technologies.

Difference between Parallel Computing and Distributed Computing

Parallel Computing:

In parallel computing multiple processors performs multiple tasks assigned to them simultaneously.
Memory in parallel systems can either be shared or distributed.
Parallel computing provides concurrency and saves time and money.

Distributed Computing:

In distributed computing we have multiple autonomous computers which seem to the user as a single system.
In distributed systems there is no shared memory and computers communicate with each other through message passing.
In distributed computing a single task is divided among different computers.

Table 3: Difference between Parallel Computing and Distributed Computing

S.NO	PARALLEL COMPUTING	DISTRIBUTED COMPUTING
1	Many operations are performed simultaneously	System components are located at different locations
2	Single computer is required	Uses multiple computers
3	Multiple processors perform multiple operations	Multiple computers perform multiple operations
4	It may have shared or distributed memory	It have only distributed memory
5	Processors communicate with each other through bus	Computers communicate with each other through message passing.
6	Improves the system performance	Improves system scalability, fault tolerance and resource sharing capabilities

ELEMENTS OF PARALLEL COMPUTING

The primary goal of parallel computing is to increase the computational power available to your essential applications.

Typically, This infrastructure is where the set of processors are present on a server, or separate servers are connected to each other to solve a computational problem.

In the earliest computer software, that executes a single instruction (having a single Central Processing Unit (CPU)) at a time that has written for serial computation. A Problem is broken down into multiple series of instructions, and that Instructions executed one after another. Only one computational instruction complete at a time.

Main Reasons to use Parallel Computing is that:

1. Save time and money.
2. Solve larger problems.
3. Provide concurrency.
4. Multiple execution units

Types of parallel computing

Bit-level parallelism

In the Bit-level parallelism every task is running on the processor level and depends on processor word size (32-bit, 64-bit, etc.) and we need to divide the maximum size of instruction into multiple series of instructions in the tasks. For Example, if we want to do an operation on 16-bit numbers in the 8-bit processor, then we would require dividing the process into two 8 bit operations.

Instruction-level parallelism (ILP)

Instruction-level parallelism (ILP) is running on the hardware level (dynamic parallelism), and it includes how many instructions executed simultaneously in a single CPU clock cycle.

Data Parallelism

The multiprocessor system can execute a single set of instructions (SIMD), data parallelism achieved when several processors simultaneously perform the same task on the separate section of the distributed data.

Task Parallelism

Task parallelism is the parallelism in which tasks are splitting up between the processors to perform at once.

Hardware architecture of parallel computing –

The hardware architecture of parallel computing is disturbed along the following categories as given below :

1. Single-instruction, single-data (SISD) systems
2. Single-instruction, multiple-data (SIMD) systems
3. Multiple-instruction, single-data (MISD) systems
4. Multiple-instruction, multiple-data (MIMD) systems

Question Bank

Unit -1

Part A

1. What is the innovative characteristic of cloud computing?
2. Which are the technologies on which cloud computing relies?
3. Provide a brief characterization of a distributed system.
4. Define cloud computing and identify its core features.
5. What are the major distributed computing technologies that led to cloud computing?
6. What is virtualization?
7. What is the difference between parallel and distributed computing?
8. List the major categories of parallel computing systems.
9. What is a distributed system? What are the components that characterize it?

Part B

10. Briefly summarize the Cloud Computing Reference Model.
11. What is the major advantage and disadvantage of cloud computing?
12. Briefly summarize the history of cloud Computing.
13. Describe the different levels of parallelism that can be obtained in a computing system.
14. Describe the difference between parallel and distributed computing, advantage and disadvantage of distributed systems

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – II – Cloud Computing – SBS1207

Unit-II

Introduction - The cloud reference model - Types of clouds - Economics of the cloud Virtualization: Introduction - Characteristics of virtualized environments - Taxonomy of virtualization techniques - Virtualization and cloud computing - Pros and cons of virtualization - Technology example: VMware: full virtualization.

TYPES OF CLOUD

- Private cloud
- Public cloud
- Hybrid cloud
- Community cloud

Public cloud:

Public cloud are managed by third parties which provide cloud services over the internet to public, these services are available as pay-as-you-go billing mode.

They offer solutions for minimizing IT infrastructure costs and act as a good option for handling peak loads on the local infrastructure. They are a goto option for small enterprises, which are able to start their businesses without large upfront investments by completely relying on public infrastructure for their IT needs.

A fundamental characteristic of public clouds is **multitenancy**. A public cloud is meant to serve multiple users, not a single customer. A user requires a virtual computing environment that is separated, and most likely isolated, from other users.

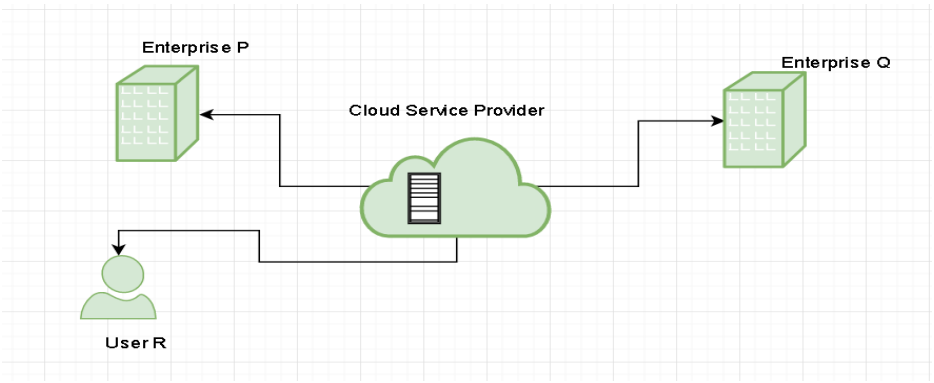


Fig 11: Public Cloud

Private cloud:

Private clouds are distributed systems that work on a private infrastructure and providing the users with dynamic provisioning of computing resources. Instead of a pay-as-you-go model as in public clouds, there could be other schemes in that take into account the usage of the cloud and proportionally billing the different departments or sections of an enterprise.

The advantages of using a private cloud are:

1. **Customer information protection:** In private cloud security concerns are less since customer data and other sensitive information does not flow out of a private infrastructure.
2. **Infrastructure ensuring SLAs:** Private cloud provides specific operations such as appropriate clustering, data replication, system monitoring and maintenance, and disaster recovery, and other uptime services.
3. **Compliance with standard procedures and operations:** Specific procedures have to be put in place when deploying and executing applications according to third-party compliance standards.This is not possible in case of public cloud.

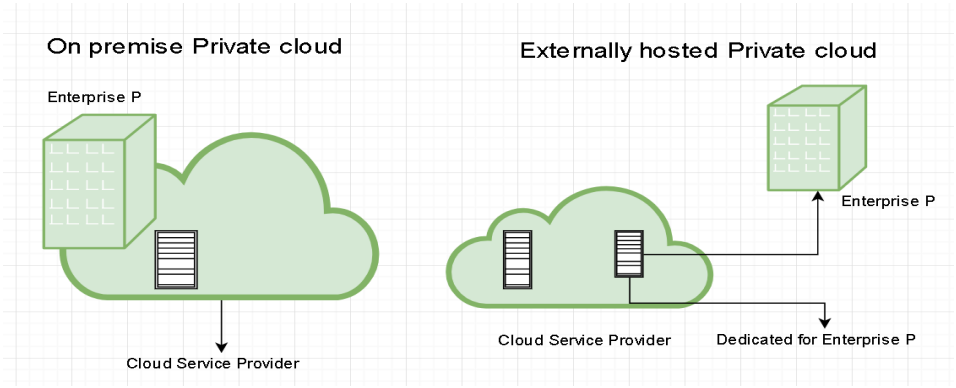


Fig 12: Private Cloud

Hybrid cloud:

Hybrid cloud is a heterogeneous distributed system resulted by combining facilities of public cloud and private cloud. For this reason they are also called **heterogeneous clouds.**

A major drawback of private deployments is the inability to scale on demand and to efficiently address peak loads. Here public clouds are needed. Hence, a hybrid cloud takes advantages of both public and private cloud.

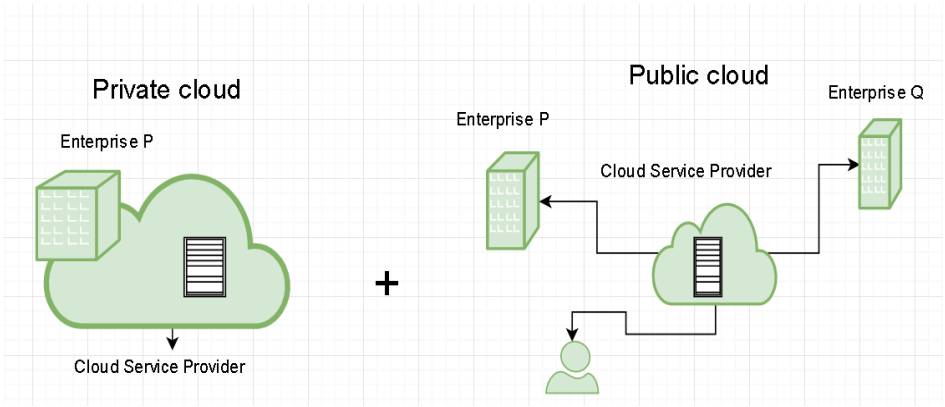


Fig 13: Hybrid Cloud

Community cloud:

Community clouds are distributed systems created by integrating the services of different clouds to address the specific needs of an industry, a community, or a business sector.

In community cloud, the infrastructure is shared between organization which have shared concerns or tasks. The cloud may be managed by an organization or a third party.

Sectors that use community clouds are:

1. **Media industry:** Media companies are looking for quick, simple, low-cost way for increasing efficiency of content generation.Most media productions involve an extended ecosystem of partners. In particular, the creation of digital content is the outcome of a collaborative process that includes movement of large data, massive compute-intensive rendering tasks, and complex workflow executions.
2. **Healthcare industry:** In healthcare industry community clouds are used to share information and knowledge on the global level with sensitive data in the private infrastructure.
3. **Energy and core industry:** In these sectors, the community cloud is used to cluster set of solution which collectively addresses management, deployment, and orchestration of services and operations.
4. **Scientific research:** In this organization with common interests of science share large distributed infrastructure for scientific computing.

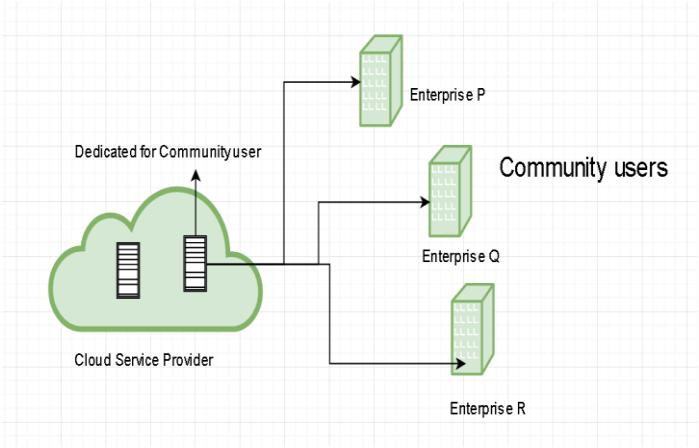


Fig 14: Community cloud

ECONOMICS OF CLOUD COMPUTING

- Economics of Cloud Computing is based on the PAY AS YOU GO method.
- Users/Customers must have to pay only for their way of usage of the cloud services.
- It is definitely beneficial for the users. So that Cloud is economically very convenient for all.
- Another side is to eliminate some indirect cost which is generated by assets such as license

of software and their support.

- In the cloud, users can use software application on subscription basis without any cost because the property of the software providing service remains to the cloud provider.

Economical background of cloud is more useful for developers in the following ways:

- Pay as you go model offered by cloud providers.
- Scalable and Simple.

Cloud Computing Allows:

Reduces the capital costs of infrastructure.

Removes the maintenance cost.

Removes the administrative cost.

What is Capital Cost ?

The cost occurred in the purchasing infrastructure or the assets that are important in the production of goods. It takes a long time to generate profit.

In the case of start-ups, there is no extra budget for the infrastructure and its maintenance. So cloud can minimizes expenses of any small organization in terms of economy. It leads to the developers can only focus on the development logic and not on the maintenance of the infrastructure.

There are **three different Pricing** Strategies which are introduced by the Cloud Computing:

- Tiered Pricing,
- Per-unit Pricing, and
- Subscription based Pricing.

Tired Pricing:

Cloud Services are offered in various tiers. Each tier offers fix service agreements at specific cost. Amazon EC2 uses this kind of pricing.

Per-unit Pricing:

The model is based upon the unit specific service concept. Data transfer and memory allocation includes in this model for specific units. GoGrid uses this kind of pricing in terms of RAM/hour.

Subscription based Pricing:

In this model users are paying periodic subscription fees for the usage of software.

So these models give more flexible solutions to the cloud economy.

CHARACTERISTICS OF VIRTUALIZATION

- **Partitioning:**
  In virtualization, many applications and operating systems (OSes) are supported in a single physical system by *partitioning* (separating) the available resources.

- **Isolation:**
  Each virtual machine is isolated from its host physical system and other virtualized machines. Because of this isolation, if one virtual-instance crashes, it doesn't affect the other virtual machines. In addition, data isn't shared between one virtual container and another.
- **Encapsulation:**
  A virtual machine can be represented (and even stored) as a single file, so you can identify it easily based on the service it provides. In essence, the encapsulated process could be a business service. This encapsulated virtual machine can be presented to an application as a complete entity. Therefore, encapsulation can protect each application so that it doesn't interfere with another application.

TAXONOMY OF VIRTUALIZATION TECHNIQUES

Five Types of Virtualization

1. Hardware Virtualization

Normally, a server devotes complete control of it's hardware resources (cpu, RAM, and storage) to the actions of a single operating system.
When you virtualize your hardware, it means that a program called a **hypervisor** manages the hardware's resources and divides them among different isolated operating systems, referred to as "virtual machines."

Fig 15: Hardware Virtualization

Advantage
It prevents a single application crash running on the server.

Portability - With modern hypervisors an IT team can migrate running virtual machines from one server to another - allowing them to do updates and maintenance without ever having downtime for the application.

1. Desktop Virtualization

Desktop Virtualization separates the desktop environment from the physical device that is used to access it.

Advantage

The desktops are accessed remotely

This creates a lot of flexibility for employees to work from home

2. Application virtualization

The user will interact with the native application on the client device using tablets or smart phones.

Advantage

Efficiency- In many situations where remote applications are useful, applications are redundant across employees.

Fig 16: Application Virtualization

3. Storage Virtualization

Storage virtualization improves storage flexibility by creating a unified virtual pool of storage from physical storage devices in a network.

It presents all physical storage in a cluster as a single shared group - visible to all servers.

Fig 17: Storage Virtualization

4. Network Virtualization

Network virtualization pools resources from all physical networking equipment and presents it to virtual machines and applications as a single virtual network.

Advantage

It reduces the provisioning time for new network architectures.

PROS AND CONS OF VIRTUALIZATION

The Advantages of Virtualization

1. It is cheaper.

Virtualization doesn't require actual hardware components to be used or installed; IT infrastructures find it to be a cheaper system to implement.

2. It keeps costs predictable.

Because third-party providers typically provide virtualization options, individuals and corporations can have predictable costs for their information technology needs.

For example: the cost of a Dell PowerEdge T330 Tower Server, at the time of writing, is $1,279 direct from the manufacturer. In comparison, services provided by Bluehost Web Hosting can be a slow as $2.95 per month.

3. It reduces the workload.

Most virtualization providers automatically update their hardware and software that will be utilized. Instead of sending people to do these updates locally, they are installed by the third-party provider.

This allows local IT professionals to focus on other tasks and saves even more money for individuals or corporations.

4. It offers a better uptime.

Uptime has improved dramatically. Some providers offer an uptime that is 99.9999%. Even budget-friendly providers offer uptime at 99.99% today.

5. It allows for faster deployment of resources.

Resource provisioning is fast and simple when virtualization is being used.

There is no longer a need to set up physical machines, create local networks, or install other information technology components.

As long as there is at least one point of access to the virtual environment, it can be spread to the rest of the organization.

6. **It promotes digital entrepreneurship.**

Before virtualization occurred on a large scale, digital entrepreneurship was virtually impossible for the average person.

7. **It provides energy savings.**

For most individuals and corporations, virtualization is an energy-efficient system. Because there aren't local hardware or software options being utilized, energy consumption rates can be lowered.

Instead of paying for the cooling costs of a data centre and the operational costs of equipment, funds can be used for other operational expenditures over time to improve virtualization's overall ROI.

The Disadvantages of Virtualization

1. **It can have a high cost of implementation.**

The cost for the average individual or business when virtualization is being considered will be quite low.

For the providers of a virtualization environment, however, the implementation costs can be quite high.

Hardware and software are required at some point and that means devices must be developed, manufactured, or purchased for implementation.

2. **It still has limitations.**

Not every application or server is going to work within an environment of virtualization.

That means an individual or corporation may require a hybrid system to function properly.

This still saves time and money in the long run, but since not every vendor supports virtualization and some may stop supporting it after initially starting it, there is always a level of uncertainty when fully implementing this type of system.

3. **It creates a security risk.**

Information is our modern currency. If you have it, you can make money. If you don't have it, you'll be ignored.

Because data is crucial to the success of a business, it is targeted frequently.

The average cost of a data security breach in 2017, according to a report published by the Ponemon Institute, was $3.62 million

4. **It creates an availability issue.**

The primary concern that many have with virtualization is what will happen to their work should their assets not be available.

If an organization cannot connect to their data for an extended period of time, they will struggle to compete in their industry. And, since availability is controlled by third-party providers, the ability to stay connected in not in one's control with virtualization.

5. **It creates a scalability issue.**

Although you can grow a business or opportunity quickly because of virtualization, you may not be able to become as large as you'd like.

You may also be required to be larger than you want to be when first starting out. Because many entities share the same resources, growth creates lag within a virtualization network. One large presence can take resources away from several smaller businesses and there would be nothing anyone could do about it.

6. **It requires several links in a chain that must work together cohesively.**

If you have local equipment, then you are in full control of what you can do. With virtualization, you lose that control because several links must work together to perform the same task.

7. **It takes time.**

Although you save time during the implementation phases of virtualization, it costs users time over the long-run when compared to local systems.

That is because there are extra steps that must be followed to generate the desired result.

TYPES OF CLOUDS

There are four different cloud models that you can subscribe according to business needs:

1. **Private Cloud:**

Computing resources are deployed for one particular organization.

This method is more used for intra-business interactions.

The computing resources can be governed, owned and operated by the same organization.

2. **Community Cloud:**

Computing resources are provided for a community and organizations.

3. **Public Cloud:**

Computing resource is owned, governed and operated by government, an academic or business organization.

4. **Hybrid Cloud:**

This type of cloud can be used for both type of interactions - B2B (Business to Business) or B2C (Business to Consumer).

This deployment method is called hybrid cloud as the computing resources are bound together by different clouds.

Full Virtualization

using Binary Translation VMware can virtualize any x86 operating system using a combination of binary translation and direct execution techniques.

Meanwhile, user level code is directly executed on the processor for high performance virtualization.

Each virtual machine monitor provides each Virtual Machine with all the services of the physical system, including a virtual BIOS, virtual devices and virtualized memory management.

This combination of binary translation and direct execution provides Full Virtualization as the guest OS is fully abstracted (completely decoupled) from the underlying hardware by the virtualization layer.

The guest OS is not aware it is being virtualized and requires no modification.

Full virtualization is the only option that requires no hardware assist or operating system assist to virtualize sensitive and privileged instructions.

The hypervisor translates all operating system instructions on the fly and caches the results for future use, while user level instructions run unmodified at native speed.

Full virtualization offers the best isolation and security for virtual machines, and simplifies migration and portability as the same guest OS instance can run virtualized or on native hardware.

VMware's virtualization products and Microsoft Virtual Server are examples of full virtualization

Question Bank

Part A

1. List the various types of clouds.
2. Give an example of the public cloud.
3. How do you estimate Economics of Cloud Computing?
4. List types of virtualization.
5. Write about Advantage of any two virtualizations.
6. Define Hypervisor.
7. What is virtualization and what are its benefits?
8. What are the disadvantages of virtualization?

Part B

1. Classify the various types of clouds
2. Discuss classification or taxonomy of virtualization at different levels.
3. What is the major advantage of disadvantage of virtualization technique?
4. Describe hypervisor, type of hypervisor with neat diagram.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – III – Cloud Computing – SBS1207

Unit-III

Anatomy of the Aneka container - Introducing parallelism for single-machine computation - Programming applications with threads - Multithreading with aneka - Programming applications with aneka threads.Cloud computing economics: Cloud infrastructure - Economics of private clouds - Software productivity in the cloud - Economies of scale: public vs. private clouds.

Anatomy of the Aneka container

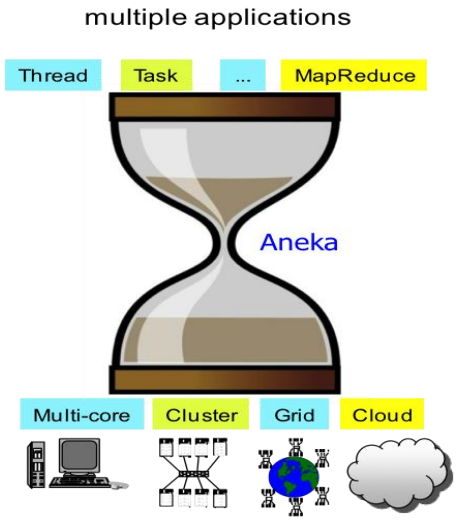


Fig 18 : Anatomy of the Aneka container

Aneka Architecture

Aneka is a platform and a framework for developing distributed applications on the Cloud.

It harnesses the spare CPU cycles of a heterogeneous network of desktop PCs and servers or datacenters on demand.

Aneka provides developers with a rich set of APIs for transparently exploiting such resources and expressing the business logic of applications by using the preferred programming abstractions.

System administrators can leverage on a collection of tools to monitor and control the deployed infrastructure. This can be a public cloud available to anyone through the Internet, or a private cloud constituted by a set of nodes with restricted access.

The Aneka based computing cloud is a collection of physical and virtualized resources connected through a network, which are either the Internet or a private intranet.

Each of these resources hosts an instance of the Aneka Container representing the runtime environment where the distributed applications are executed.

The container provides the basic management features of the single node and leverages all the other operations on the services that it is hosting.

The services are broken up into

- Fabric,
- Foundation, and

- Execution services.

Fabric services directly interact with the node through the Platform Abstraction Layer (PAL) and perform hardware profiling and dynamic resource provisioning.

Foundation services identify the core system of the Aneka middleware, providing a set of basic features to enable Aneka containers to perform specialized and specific sets of tasks.

Execution services directly deal with the scheduling and execution of applications in the Cloud.

One of the **key features of** Aneka is the ability of providing different ways for expressing distributed applications by offering different programming models;

Execution services are mostly concerned with providing the middleware with an implementation for these models.

Additional services such as persistence and security are transversal to the entire stack of services that are hosted by the Container.

At the application level, a set of different components and tools are provided to:

1) Simplify the development of applications (SDK);

2) Porting existing applications to the Cloud; and

3) Monitoring and managing the Aneka Cloud.

A common deployment of Aneka is presented at the side. An Aneka based Cloud is constituted by a set of interconnected resources that are dynamically modified according to the user needs by using resource virtualization or by harnessing the spare CPU cycles of desktop machines.

If the deployment identifies a private Cloud all the resources are in house, for example within the enterprise. This deployment is extended by adding publicly available resources on demand or by interacting with other Aneka public clouds providing computing resources connected over the Internet.

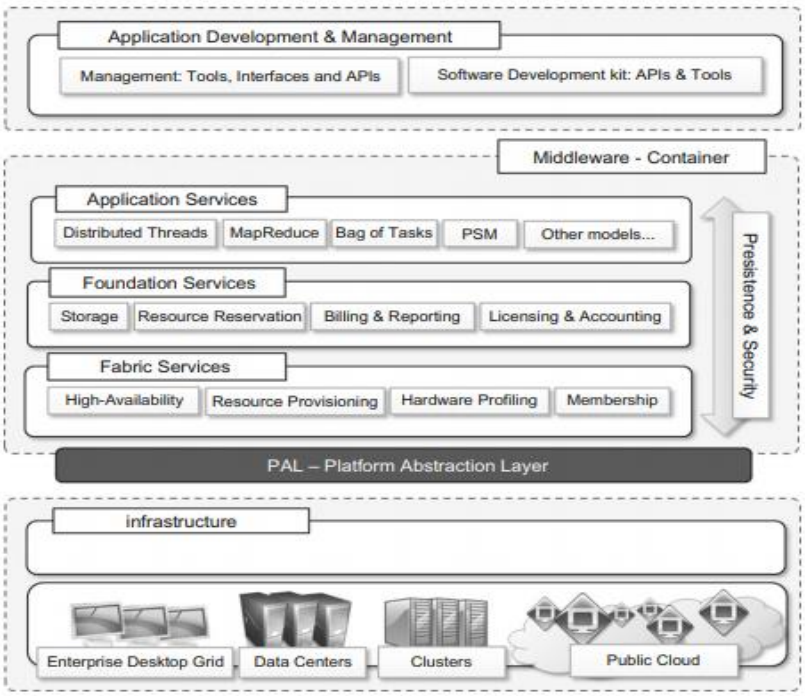


Fig 19: Services

Parallelism for single machine computation

Parallelism has been a technique for improving the performance of computers since the early 1960's, when Burroughs Corporation designed the D825, the first MIMD multiprocessor ever produced.

From there on, a variety of parallel strategies have been developed.
In particular, multiprocessing, which is the use of multiple processing units within a single machine, has gained a good deal of interest and gave birth to several parallel architectures.

One of the most important distinctions is made in terms of the symmetry of processing units.

Asymmetric multi-processing involves the concurrent use of different processing units that are specialized to perform different functions.

Symmetric multiprocessing features the use of similar or identical processing units to share the computation load.

Other examples, are non uniform memory access (NUMA) and clustered multi processing, which, respectively, define a specific architecture for accessing a shared memory between processors and the use of multiple computers joined together as a single virtual computer.

Symmetric and asymmetric multiprocessing are the techniques used to increase the performance of commodity computer hardware.

The introduction of graphical processing units(GPUs), which are de facto processors, is an application of asymmetric processing, whereas multi core technology is the latest evolution of symmetric multiprocessing.

Multiprocessor and especially multi core technologies are now of fundamental importance because of the physical constraint imposed on frequency scaling, which has been the common practice for performance gain in recent years.

It became no longer possible to increase the frequency of the processor clock without paying in terms of power consumption and cooling, and this condition became unsustainable in May 2004, when Intel officially cancelled the development of two new microprocessors in favor of multi core development.

This date is generally considered the end of the frequency-scaling era and the beginning of multi core technology. Other issues also determined the end of frequency scaling, such as the continuously increasing gap between processor and memory speeds and the difficulty of increasing the instruction-level parallelism in order to keep a single high-performance core busy.

Multicore systems are composed of a single processor that features multiple processing cores that share the memory.

Each core has generally its own L1 cache, and the L2 cache is common to all the cores, which connect to it by means of a shared bus.

Dual- and quad-core configurations are quite popular nowadays and constitute the standard hardware configuration for commodity computers.

Architectures with multiple cores are also available but are not designed for the commodity market.

Multicore technology has been used not only as a support for processor design but also in other devices, such as GPUs and network devices, thus becoming a standard practice for improving performance.

multitasking and multithreading can be implemented on top of computer hardware that is constituted of a single processor and a single core, as was the common practice before the introduction of multi core technology.

In this case, the operating system gives the illusion of concurrent execution by interleaving the execution of instructions of different processes, and of different threads within the same process.

This is also the case in multiprocessor/multicore systems, since the number of threads or processes is higher than the number of processors or cores.

Nowadays, almost all the commonly used operating systems support multitasking and multithreading.

Moreover, all the mainstream programming languages incorporate the abstractions of process and thread within their APIs, whereas direct support of multiple processors and cores for developers is very limited and often reduced and confined to specific libraries, which are available for a subset of the programming languages such as C/C++.

Programming Applications with Threads

- Modern applications perform multiple operations at the same time. Developers organize programs in terms of threads in order to express intra process concurrency.
- The use of threads might be implicit or explicit .
- Implicit threading happens when the underlying APIs use internal threads to perform specific tasks supporting the execution of applications such as graphical user interface (GUI) rendering, or garbage collection in the case of virtual machine-based languages.
- Explicit threading, is characterized by the use of threads within a program by application developers, who use this abstraction to introduce parallelism.

Thread:

- A thread identifies a single control flow, which is a logical sequence of instructions, within a process.
- By logical sequence of instructions, we mean a sequence of instructions that have been designed to be executed one after the other one.

- More commonly, a thread identifies a kind of yarn, that is used for sewing, and the feeling of continuity that is expressed by the interlocked fibers of that yarn is used to recall the concept that the instructions of thread express a logically continuous sequence of operations.
- Operating systems that supports multithreading identify threads as the minimal building blocks for expressing running code.
- This means that, despite their explicit use by developers, any sequence of instruction that is executed by the operating system is within the context of a thread.
- As a consequence, each process contains at least one thread but, in several cases, is composed of many threads having variable lifetimes. Threads within the same process share the memory space and the execution context; besides this, there is no substantial difference between threads belonging to different processes.
- In a multitasking environment the operating system assigns different time slices to each process and interleaves their execution.
- The process of temporarily stopping the execution of one process, saving all the information in the registers (and in general the state of the CPU in order to restore it later), and replacing it with the information related to another process is known as a context switch.
- This operation is generally considered demanding, and the use of multithreading minimizes the latency imposed by context switches, thus allowing the execution of multiple tasks in a lighter fashion.
- The state representing the execution of a thread is minimal compared to the one describing a process.

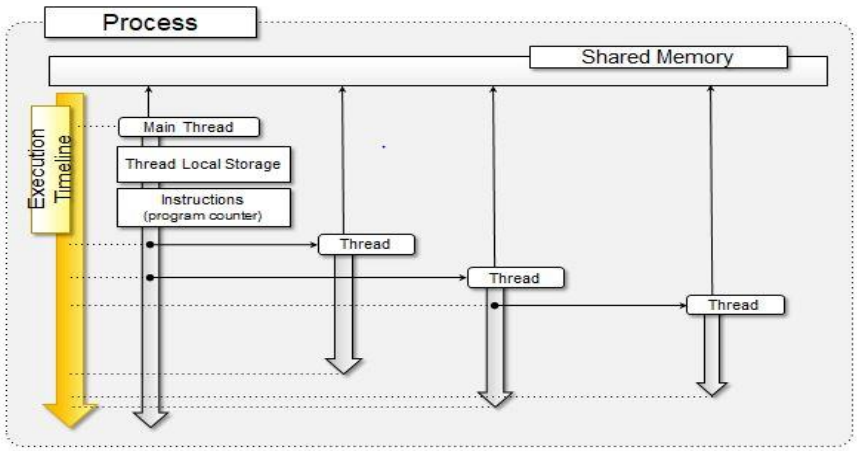


Fig20: Thread

Multithreading with Aneka

As applications become increasingly complex, there is greater demand for computational power that can be delivered by a single multi-core machine.

• Often, this demand cannot be addressed with the computing capacity of a single machine.

• It is then necessary to leverage distributed infrastructures such as Clouds.

• Decomposition techniques can be applied to partition a given application into several units of work that, rather than being executed as threads on a single node, can be submitted for execution by leveraging Clouds.

Even though a distributed facility can dramatically increase the degree of parallelism of applications, its use comes with a cost in term of application design and performance.

• For example, since the different units of work are not executing within the same process space but on different nodes both the code and the data needs to be moved to a different execution context.

• The same happens for results that need to be collected remotely and brought back to the master process.

• Moreover, if there is any communication among the different workers it is necessary to redesign the communication model eventually by leveraging the APIs provided by the middleware if any.

• In other words, the transition execution to a distributed from a single execution is process multi-threaded not transparent and application redesign and re-implementation are often required.

The amount of effort required to convert an application often depends on the facilities offered by the middleware managing the distributed infrastructure.

• Aneka, as a middleware for managing clusters, Grids, and Clouds, provides developers with advanced capabilities for implementing distributed applications.

• In particular, it takes traditional thread programming a step further. It lets you write multi-threaded applications in the traditional way, with the added twist that each of these threads can now be executed outside the parent process and on a separate machine.

• In reality, these "threads" are independent processes executing on different nodes, and do not share memory or other resources, but they allow you to write applications using the same thread constructs for concurrency and synchronization as with traditional threads.

• Aneka threads, as they are called, let you easily port existing multi-threaded compute intensive applications to distributed versions that can run faster by utilizing multiple machines simultaneously, with a minimum conversion effort.

Economics of cloud computing

Cloud computing economics depends on **four customer population metrics**:

1. Number of Unique Customer Sets
2. Customer Set Duty Cycles
3. Relative Duty Cycle Displacement
4. Customer Set Load

These above metrics allow the cloud-providers to use less IT resources and obtain maximum IT resource.

Proper balancing & handling of these above resources can save up to 30% of the IT resources. For economic planning, Booz Allen Hamilton, a leading strategy, and technology consulting firm, has a clear understanding of and models for effective Cloud Computing-based life-cycle cost and economic modeling.

This firm addresses every aspect of cost related efforts & questions such as:

- If you are migrating current systems to a cloud, how will you handle (and cost) the migration?
- Will you migrate all IT related tasks into the cloud, or it will be partially migrated?
- What IT chores should remain in the current environment?
- Will you use a public cloud, or you will need a private cloud?
- If you are migrating IT requirements into a cloud, how will you handle and budget for the short-term operations of distributed and cloud infrastructures?
- What can special-purpose computing tasks be enabled in the cloud technology that wasn't possible in the current environment?
- Can the cloud provide levels of service commensurate with existing service-level agreements?

Cloud technology provides users with strong facilities & economic incentives. The selection for implementing the private, public, community or hybrid cloud solely depends on the customer's specification for applications they want to use, the performance they need, the security they want to take & compliance requirement. Proper deployment model can save monetary value as well as time in a significant manner, provide better IT services & provide a higher level of reliability.

Achieve Cloud Economics for Operation & Services

Users get frustrated when they need to secure their resources to business requirements. Even if they're willing to pay the attributed cost, they may find that technology cannot deliver resources to address their needs. Frustrated by the unavailability of resources required to fulfill their needs, and for that, they opt for the options to obtain resources are as follows:

To obtain sufficient resources for enabling the business opportunities

Hire a senior IT manager or an IT management team to put their request. This may help in letting the business proceed forward with the full initiative. But doing this fosters disrespect for the existing cloud process.

- Request to the senior corporate manager for raising the technology and resources (in case of cloud-security, or cloud-infrastructure).

Economic Characteristics of Cloud

- **Scalability:** Access to unlimited computer resources without thinking about the economic aspects. This feature needs planning & provisioning.
- **Low Entry barrier:** Users can gain access to systems for small investments also; which allows the offer to access global resources to small ventures.
- **Flexibility:** Cloud provides high economic elasticity. Users can re-size their resources based on their need. This feature allows optimizing the system & captures all possible requirements.

- **Utility:** As cloud providers 'pay-as-you-go' model, users can match their needs & resources on an ongoing basis. This eliminates waste and added benefits of shifting risks from the client.

Cloud infrastructure:

It refers to the software along with the hardware components such as storage drive, hardware, servers, virtual software, other cloud management software, and other networking devices;

All work together to support the computing requirement of the cloud computing model.

Moreover, the cloud technology holds a software abstraction layer that virtualizes the cloud resource & presents them to users locally.

Cloud Infrastructure Management Interface (CIMI) is an open standard API that is used to manage the cloud infrastructure. It enables its users to handle the entire cloud infrastructure easily by providing a means to interact with the provider & their consumer or developer.

The hypervisor can be defined as the firmware (a permanent set of instruction or code programmed into the read-only memory & is a low-level program) that acts as a manager for the virtual machine. It is also called Virtual Machine Monitor (VMM) which creates & runs the virtual machine. It provides the guest OS with a virtual operating platform to manages the execution of other applications.

There are two types of the hypervisor. These are:

- Native Hypervisor
- Hosted Hypervisor

In cloud technology, virtualized resources are kept & maintained by the service provider or the department of IT; these resources comprise of servers, memory, network switches, firewalls, load-balancers & storage. In the cloud computing architecture, the cloud infrastructure referred to the back-end components of the cloud.

Management Software firstly helps to configure the infrastructure then maintaining it. The **Deployment software**, on the other hand, is used to deploy & combine all applications on the cloud.

Network, as we all know is the key part of cloud technology allowing users to connect to the cloud via the internet. Multiple copies of data are kept stored in the cloud. This is because, if any storage resource fails - then the data can be extracted from another one. So, **storage** is another essential component of cloud infrastructure.

Server helps to handle & compute all cloud resources & offer services like allocation, de-allocation, sharing and monitoring the cloud resources and in some cases used to provide security.

Restrictions or Limitations of Cloud Infrastructures

The limitations of cloud technology concerning infrastructure are:

- Scalability
- Intelligent Monitoring
- Security

SOFTWARE PRODUCTIVITY IN THE CLOUD

Cloud-Based Software's Advantage

- Better productivity
- Efficiency for employees
- Lower spending on hardware
- Reduced energy costs
- Flexibility
- Scalability and future focus

3 Disadvantages of Cloud-Based Productivity Software

1) Downtime

2) **Security**

3) **Cost**

Google Office Apps for Business

- Google is offering the best cloud-based productivity software names as Google Office Applications. When you talk about Cloud software, it is impossible for Google to be left behind. With Google drive, you will be able to avail of around 15 GB of storage (30GB paid) which is available for free Google online storage.

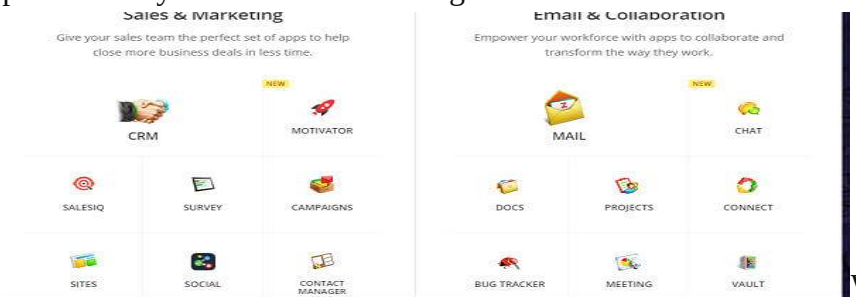
Microsoft Office 365

- Office 365, one of the well-known cloud-based productivity software from Microsoft.
- With Microsoft Office, you will be able to keep everything organized and up-to-date.
- One will be able to keep polished documents through the use of Microsoft Office. The best part of Office remains that it is accessible across devices.
- You will also get to have free training with each application of Microsoft Office.
- It is easy, smart and very creative.

- 

Many companies prefer the use of Microsoft Office due to its sheer compatibility with Windows 10.
- Microsoft Office is famous for its use because of its usage for teamwork.
- It is one of the most used Office online features which is built-in with the setup of Office blog.
- It is a perfectly able, compatible set of Docs.

Zoho Office Online Suite

- Zoho is right behind Microsoft and Google in cloud-based productivity software list.
- This name has evolved and achieved its standing in the market thanks to its many features which lets it do so.
- It helps you with your daily activities, boost sales records and increase employee productivity.
- It also has features of campaigns, connects and bug tracker available with it.
- It allows you to see show times, set up documents or survey forms for research or even personalize your contacts to the largest extent.
- 

With the right kind of Helpdesk and Human Resource available, Zoho could be the right choice of Office available.

Apple iWork for iCloud

- Apple launched iWork, a cloud-based productivity software access with the iCloud account.
- Apple will stay back and not invent its inventory of Office online products under the tagline ‘I’.
- Some of its features are Keynote for office users, Libre office project, iScript, and Automation features.

- One of the best features according to many experts is the iPhoto feature which is unique and sets it apart from the rest.
- The only downside is that Apple created its own echo system, which allows only their own device owners to enjoy the benefits of these suits.
- If you don't have free Apple / iCloud ID, you can get one free.
- 

Thus these, are some excellent Online Office applications which should be of use for various users who are looking for personalized products of Online Office.

Economics of scale: public vs. private clouds

If any organisation has the expertise to manage a large number of servers at a high level of utilisation then on-premises, customer-managed private clouds can have a total cost of ownership (TCO) advantage compared to public clouds.

For smaller environments, or any sort of variable workload demand, public cloud is a more attractive.

To **determine total cost of ownership** (TCO), an organisation consider

How large of a cloud deployment they will have,

How efficiently they can manage it, and

How much it will be utilised.

A key measurement in determining efficiency of a cloud is the number of virtual machines managed per engineer.

If an organisation is able to achieve about 400 virtual machines under management per engineer, then a private cloud can be a more attractive financial option compared to a public cloud.

If the organisation is not able to reach that efficiency level, then public cloud can be more efficient.

Question Bank

Part A

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

Part B

1. Discuss the role of dynamic provisioning in hybrid deployments.
2. Which facilities does Aneka provide for development?
3. Discuss the major features of the Aneka Application Model.
4. Discuss the major features of the Aneka Service Model.
5. Describe the features of the Aneka management tools in terms of infrastructure, platform, and applications.

References:

[1]. http://www.manjrasoft.com/aneka_architecture.html

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – IV – Cloud Computing – SBS1207

Unit-IV

Multi-entity support - Multi-schema approach - Multi-tenancy using cloud data stores - Data access control for enterprise applications.Data in the cloud: Relational databases - Cloud file systems: GFS and HDFS - BigTable, HBase - Cloud data stores: Datastore and SimpleDB.

Multi-entity support

A **multi-tenant cloud** is a **cloud computing** architecture that allows customers to share **computing** resources in a public or private **cloud**. Each **tenant's** data is isolated and remains invisible to other **tenants**. In a **multi-tenant cloud** system, users have individualized space for storing their projects and data.

Benefits of multitenancy

Better use of resources: One machine reserved for one tenant isn't efficient, as that one tenant is not likely to use all of the machine's computing power. By sharing machines among multiple tenants, use of available resources is maximized.
Lower costs: With multiple customers sharing resources, a cloud vendor can offer their services to many customers at a much lower cost than if each customer required their own dedicated infrastructure.

Drawbacks of multitenancy

Possible security risks and compliance issues: Some companies may not be able to store data within shared infrastructure, no matter how secure, due to regulatory requirements. Additionally, security problems or corrupted data from one tenant could spread to other tenants on the same machine, although this is extremely rare and shouldn't occur if the cloud vendor has configured their infrastructure correctly. These security risks are somewhat mitigated by the fact that cloud vendors typically are able to invest more in their security than individual businesses can.
The "noisy neighbor" effect: If one tenant is using an inordinate amount of computing power, this could slow down performance for the other tenants. Again, this should not occur if the cloud vendor has set up their infrastructure correctly.

Multitenancy in public cloud computing

This is similar to the way many public cloud providers implement multitenancy. Most cloud providers define multitenancy as a shared software instance. They store metadata* about each tenant and use this data to alter the software instance at runtime to fit each tenant's needs. The tenants are

isolated from each other via permissions. Even though they all share the same software instance, they each use and experience the software differently.

Multitenancy in private cloud computing

Private cloud computing uses multitenant architecture in much the same way that public cloud computing does. The difference is that the other tenants are not from external organizations. In public cloud computing, Company A shares infrastructure with Company B. In private cloud computing, different teams within Company A share infrastructure with each other.

Multitenancy is often used in cloud computing, to offer shared tenancy on public cloud providers like Amazon Web Services and Microsoft Azure. Additionally, multitenancy is a key part of another cloud model, software as a service, and so is deployed by many software as a service companies.

Development of Multi-Tenant Architecture

The concept of multitenancy actually dates back to the 1960s, when companies rented time on mainframes, which were rare and expensive. Back then it was called time sharing. Multiple customers could access the same apps at the same time, a feat only mainframes could do.

Starting in the 1990s, application service providers (ASPs) hosted applications on behalf of their customers and like mainframes, the same apps were made available to multiple customers.

In the modern era, multitenancy is part and parcel with software-as-a-service model like Salesforce, Office 365, Zoho, Box, Zendesk, Slack, and many more applications on demand.

As mentioned above, cloud providers do offer multitenancy, in a technique of shared use of computing resources. However, this shared use of resources should not be confused with virtualization, a closely related concept. In a multitenancy environment, multiple customers share the same application, in the same operating environment, on the same hardware, with the same storage mechanism. In virtualization, every application runs on a separate virtual machine with its own OS.

Single-Tenant vs. Multi-Tenant

Single-tenancy is largely seen as the "deluxe" option, in that a client operates in a solely dedicated environment. In a multi-tenant environment, each customer shares the software application along with a single database, so multiple people from the same company can access the database. Still, even in multi-tenant, each tenant is isolated from other tenants.

Advantages of Single-Tenant Hosting

Single-tenant hosting gives clients more control because there is no sharing of resources. This manifests in a number of ways:

- **Greater customization:** Since they have only client, single tenants can customize the software for their needs, whereas multi-tenant tends to be one-size-fits-all.
- **Greater isolation from security risks:** You control the environment and (hopefully) what goes in and out of it.
- **Faster recovery:** Restoring one client is faster and easier than many.

- **Better control:** Single-tenant can be choosier about accepting software changes and updates and decide what add-ons they want to use.
- **Avoiding "noisy neighbor" syndrome:** Since you are sharing resources in multi-tenant scenarios, someone else who is really heavily using the system might slow you down.

Disadvantages of Single-Tenant Hosting

Though some companies prefer it, there are downsides to single-tenancy.

- **Cost:** There is no cost sharing for things like balancing, services, system monitoring, and deployment.
- **Client responsibility:** Clients are responsible for software updates, patches, backup, restore, and disaster recovery.
- **Less efficient:** Single-tenant systems can be less efficient if they don't run at full capacity or if they are over-provisioned.

Advantages of Multi-Tenant Hosting

The chief advantage of multi-tenant hosting is that it is less expensive. The resource pooling greatly reduces the cost since you only pay for what you need. And since multi-tenant is part of a SaaS provider, you are not paying for on-premises hardware. Functions like system monitoring and servicing the deployment become shared among all of the customers, which makes it less expensive as the cost is spread around.

There are other advantages as well.

- **Simplified hosting:** It's not your hardware to manage any more, reducing a lot of time and expense.
- **Better protection of systems:** With less interaction with the outside world, exposure to malicious software is reduced.
- **Upgrading software is no longer your problem:** You always get the latest version of software pushed out to you by the provider.

Disadvantages of Multi-Tenant Hosting

Despite its cost advantage, multi-tenant environments have some downsides.

- **They have their own security risks:** For starters, you need strict authentication and access controls to make sure the right people get access. Second, data corruption could possibly spread from one user to all, though precautions guard against this.
- **Downtime:** Outages can be nationwide, and often make the news when they happen. SaaS providers tend to build enough redundancy into the system to minimize this.
- **Noise neighbors:** As mentioned earlier, someone else on your CPU might be consuming cycles and slowing you down. Capacity is supposed to be elastic and expand as needed but that's not always the case.

Multi-Tenant Databases

As we said above, in a multi-tenant environment, multiple customers share the same application, in the same operating environment, on the same hardware, with the same storage mechanism and

database. This is how Salesforce and every other SaaS operator runs. Every tenant is a customer/user who has common access and specific privileges in the software instance.

The database, however, is another matter. There are three ways to architect your database in a multi-tenant system.

A single, shared database schema

A schema is a layout for database tables that relate to each other. In the first approach, one database is used with tenant tables all linked to the database. The tables handle relations and version control or updates, such as handling two people attempting to manipulate the same table or data entry. This is the fastest way to operate, since only one database is being used, assuming it scales.

Single database, multiple schemas

Multiple schemas inside a single database is a popular method to have sub-databases, so you can divide up your data without having to set up multiple databases. Each schema is isolated from the others and operates differently, which can be useful in situations where different data has different regulation, like international data.

Multiple databases

This takes the multi-schema approach one step further because now you have the data in multiple databases. Sales or customers can be divided up by region, for example. So the upside is you get the best isolation of data. Of course, that also adds to the complexity of management, maintenance and scalability that would come with deploying multiple databases.

Multi-Tenant Architecture Examples

In a virtual system, one system may have 20 instances running 20 operating systems, each with its own application and database. In a multi-tenant architecture, all of the instances share the OS, app, and most importantly, the database.

Does this sound familiar? That's not only how SaaS works but how Infrastructure as a Service (IaaS) and Platform as a Service (PaaS) operate, since both can be used to write highly scalable multitenant apps that are outward-facing, meaning customers and partners can use them.

That's how 50 people from the same company can work on Salesforce CRM. Similarly, a SAP system is composed of a database backend and Web application servers that host Web services in a highly scalable manner. The Web services that make up the SaaS app are exposed to different customers via different domain names. Scaling is achieved by starting more services.

Most of the example conform to these three ways to structure a multi-tenant application:

- **URL-based SaaS.** This is the easiest one to do since it uses a single domain and database. You would have specific URLs, like subdomain1.maindomain.com, subdomain2.maindomain.com, etc. Data management and security is handled at the application level. Some SaaS operates this way, especially those that put a Web app interface between the user and the database.

- **Multi-tenant SaaS.** This is more complex because of multiple databases and/or schemas, and restrictions are done at the database level. This is how many SaaS apps operate and often allow more direct interaction with the database.
- **Virtualization-based SaaS, like containers.** This is the most complex setup because there is so much interaction between the containers as well as the apps and databases. That's why there is so much emphasis on orchestrators like Docker and Kubernetes.

Choosing Between Multi-Tenant and Single Tenant

Choosing between single- and multi-tenant comes down to a choice of on-premises vs. cloud. There is no single-tenant version of Salesforce and, in contrast, databases like Oracle are single-tenant so as to have full access to resources.

Security of data is clearly a concern, but that falls primarily on the shoulders of SaaS providers. They are the ones responsible for monitoring tenants and making sure there is no data bleed from one customer to another, and they do a good job of it.

The client's primary responsibility for securing the data falls to their client device. All of the major SaaS providers do offer two-factor authentication to secure logins. After that it's up to you to maintain the security of the endpoint device.

A major concern between single- and multi-tenant systems is data ownership. Larry Ellison once called Salesforce a "roach motel of clouds," meaning data goes in but doesn't come out. Obviously, he was overstating the matter. It is certainly possible to get your CRM data out of Salesforce, but more likely in a CSV format than a SQL database.

Multi-tenancy is at the heart of cloud computing. It is designed to help scale up thousands of users both within an enterprise and externally as companies interact and do business. Whether it's your Salesforce account or an app you built on AWS for customers, multitenancy can scale through public and private clouds and provide true economies of scale.

Multi-tenancy using cloud data stores

The cloud storage has both financial and security advantages. Financial advantages in the case that the virtual resources are cheaper than dedicated physical resources connected to a personal computer or network.

There are four main types of cloud storage:

Personal Cloud Storage: It is also called mobile cloud storage which is a subset of public cloud storage that applies to storing an individual's data in the cloud and provides access to data from anywhere. . Apple's iCloud is an example of personal cloud storage.

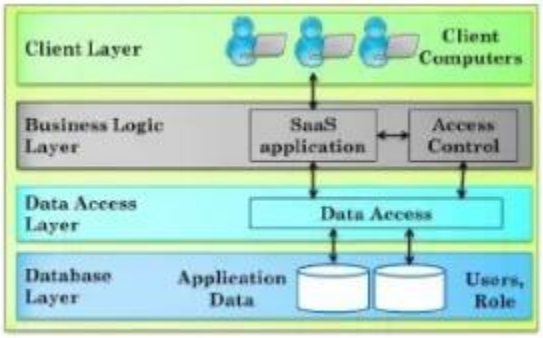
Public Cloud Storage: Public cloud storage is where the storage service providers and enterprises are separate and there aren't any cloud resources stored in the enterprise's data center. The enterprise data management is done by cloud storage provider.

Private Cloud Storage: A form of cloud storage where the data storage is done within the enterprise. This helps to resolve the potential risk for performance and security concerns while still providing the advantages of cloud storage.

Hybrid Cloud Storage: Hybrid cloud storage is a combination of private and public cloud storage where some data that is critical resides in the enterprise's private cloud while other data is stored and accessible from a public storage provider.

Tenant Data Management in Multi-Tenant Cloud Systems

Multi-tenancy enables the client service providers to serve multiple clients by a single application instance, isolating each tenant's data. The benefits of this method include increased utilization of available hardware resources and improved ease of maintenance and deployment. These results are in lower overall application costs. In a multi-tenant architecture, a software application is designed virtually to partition data. Each tenant works in a virtual application instance in which some tenants may be divided into multiple subtenants, each having its own users. As the number of tenants increases, a scalable architecture is needed



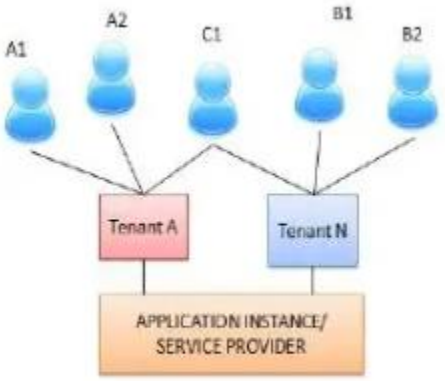


Fig 1: Tenant Organization

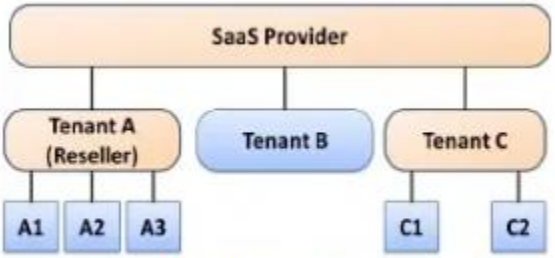


Fig 3 Tenant Tree

The business logic layer provides security by the access control component which should be isolated from the offered services. The database layer stores the application data. To enforce scalability of the database layer a logical hierarchical representation of the tenants and subtenants, and making a mapping to the physical storage is followed. The data access layer provides load balancing between the different data stores and stores the decision support for managing the data store.Tenants and subtenants in a multi-tenant organization can be structured hierarchically in the form of a tenant tree. At the top level of the tree is the SaaS provider, which is the root node of the tenant tree.

Various tenants using the applications are located at the next level, and is the child nodes of SaaS provider. All tenants share the same parent or root node. Some tenants may be even divided into multiple subtenants.

The representation of tenants is done by logical hierarchical representation. The data could be split as in the logical representation, by creating a data stores for tenants or merging smaller databases. For small applications with a less number of tenants, can share a single data store. There are 3different models for load balancing

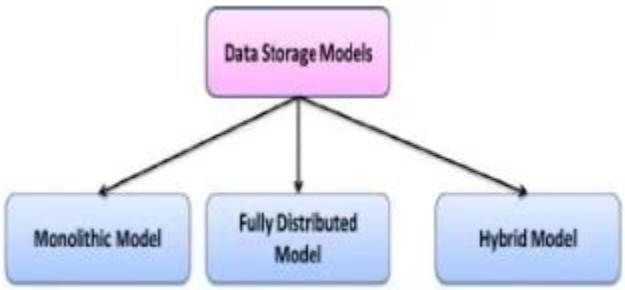


Fig 4 Physical Storage

In the monolithic model, all data is stored in a single data store. In the fully distributed model every tenants and subtenants has its own data store. Whereas the hybrid model is a mixture of both monolithic and fully distributed models. For new SaaS applications with limited number of tenants, the service provider could use the monolithic model for storing data, and move to a fully distributed model or hybrid as the amount of data and the number of tenants increases. The monolithic model is easier to implement at a lower costs, as it requires only a single database instance. The fully distributed model is more expensive but it offers more scalability and flexibility.

4.1 MULTI-ENTITY SUPPORT

Long before ASPs and SaaS, large globally distributed organizations often needed their applications to support multiple organizational units, or 'enti- ties,' in a segregated manner.

For example, consider a bank with many branches needing to transition from separate branch specific installations of its core banking software to a centralized deployment where the same software would run on data from all branches.

The software designed to operate at the branch level clearly could not be used directly on data from all branches: For example branch-level users should see data related only to their branch and branch-wise accounting should consider transactions segregated by branch. If there was a need to enhance the system, say by introducing a new field, such a change would need to apply across all branches; at the same time, sometimes branch specific extensions would need to be supported as well. These

requirements are almost exactly the same as for multi-tenancy!

In a multi-entity scenario there are also additional needs, such as where a subset of users needed to be given access to data from all branches, or a sub- set of branches, depending on their position in an organizational hierarchy.

Similarly, some global processing would also need to be supported, such as inter-branch reconciliation or enterprise-level analytics, without which the benefits of centralization of data might not be realized.

Figure 4.1 depicts the changes that need to be made in an application to support basic multi-entity features, so that users only access data belong- ing to their own units. Each database table is appended with a column (OU_ID) which marks the organizational unit each data record belongs to.

Each database query needs to be appended with a condition that ensures that data is filtered depending on the organizational unit of the currently logged-in user, which itself needs to be set in a variable, such as current_user_OU, during each transaction.

An exactly similar mechanism can be used to sup- port multi-tenancy, with OU_ID now representing the customer to whom data records belong. Note that the application runs on a single schema containing data from all organizational units; we shall refer to this as the *single SCHEMA* model.

Many early implementations of SaaS products utilized the single schema model, especially those that built their SaaS applications from scratch. One advantage of the single schema structure is that upgrading functionality of the

Other fields					OU_ID
					North
					North
					North
					South
					South

SELECT … FROM T WHERE OU_ID=:current_user_OU

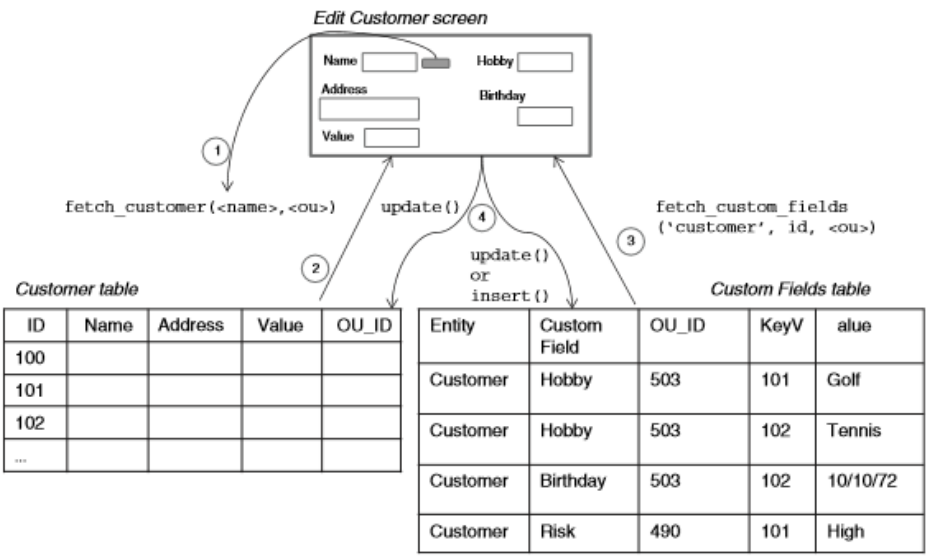
FIGURE 4.1. Multi-entity implementation

Application, say by adding a field, can be done at once for all customers.

At the same time, there are **disadvantages**:

Re-engineering an existing applica- tion using the single schema approach entails significant re-structuring of application code. For a complex software product, often having millions of lines of code, this cost can be prohibitive. Further, while modifications to the data model can be done for all customers at once, it becomes difficult to support customer specific extensions, such as custom fields, using a single schema structure. Meta-data describing such customizations, as well as the data in such extra fields has to be maintained separately. Further, it remains the responsibility of the application code to interpret such meta-data for, say, displaying and handling custom fields on the screen. Additionally, any queries that require, say, filtering or sorting on these custom fields become very complex to handle. Some of these issues can be seen more clearly through the example in Figure 4.2 that depicts a multi- tenant architecture using a single schema model which also supports custom fields:

In the single schema model of Figure 4 .2, a Custom Fields table stores meta-Information and data values for *ALL* tables in the application. Mechanisms for handling custom fields in single schema architecture are usually variants of this scheme. Consider a screen that is used to retrieve and update records in the Customer table. First the record from the main table is retrieved by name, suitably filtered by the OUattribute of the logged in user. Next, custom fields along with their values are retrieved from the Custom Fields table, for *this* particular record *AND* the OUof the logged in user. For example, in OU503, there are two custom fields as displayed on the screen, but only one in OU



ID	Name	Address	Value	OU_ID
100				
101				
102				
...				

Entity	Custom Field	OU_ID	KeyV	alue
Customer	Hobby	503	101	Golf
Customer	Hobby	503	102	Tennis
Customer	Birthday	503	102	10/10/72
Customer	Risk	490	101	High

FIGURE 4.2. Multi-tenancy using a single schema

490, and none otherwise. Furthermore, some records may have missing values for these fields, so while saving the record care must be taken to appropriately either insert or update records in the Custom

Fields table.

The above example is a simple case; more complex requirements also need to be handled, for example where a list of records is to be displayed with the ability to sort and filter on custom fields. It should be clear from this example that the single schema approach to multi-tenancy, while seemingly having the advantage of being able to upgrade the data model in one shot for all customers, has many complicating disadvantages in addition to the fact that major re-engineering of legacy applications is needed to move to this model.

4.2 MULTI-SCHEMA APPROACH

Instead of insisting on a single schema, it is sometimes easier to modify even an existing application to use multiple schemas, as are supported by most relational databases. In this model, the application computes which OU the logged in user belongs to, and then connects to the appropriate database schema. Such architecture is

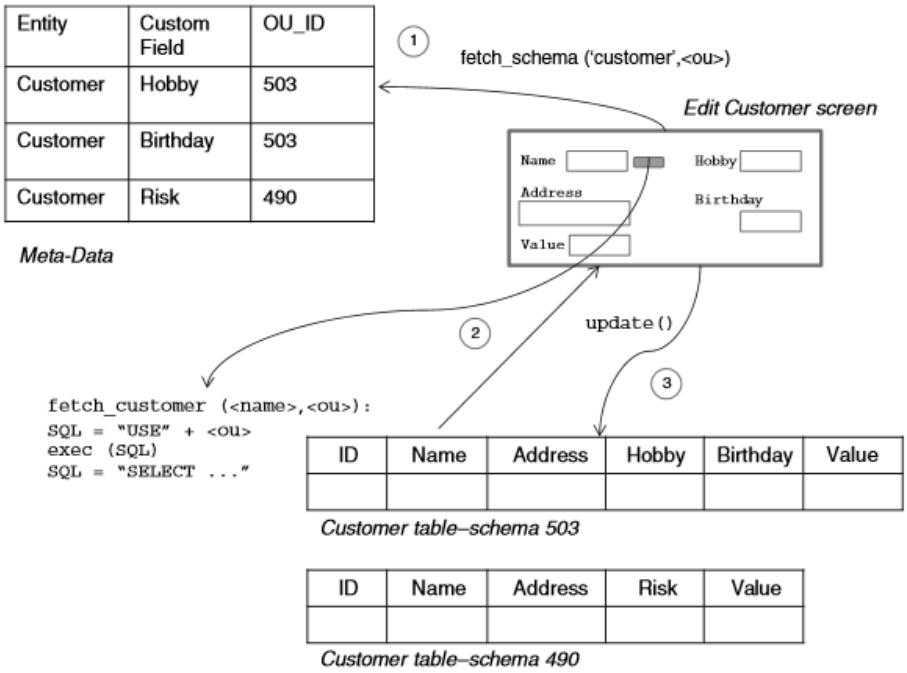


FIGURE 4.3. Multi-tenancy using multiple schemas

In the multiple schema approach a separate database schema is maintained for each customer, so each schema can implement customer-specific cus- tomizations directly. Meta-data describing customizations to the core schema is also maintained in a separate table, but unlike the Custom Fields table of Figure 4.2, this is pure meta-data and does not contain field values in individ- ual records. As a

result, the application design is simpler, and in case a legacy application needs to be re-engineered for multi-tenancy, it is likely that the modifications will be fewer and easier to accomplish.

Consider implementing the Edit Customer screen as discussed earlier using a multiple schema approach: The application renders the appropri- ate fields on the screen using information from the Meta-Data table. When making a database query, the application sets the database schema before issuing data manipulation (i.e. SQL) statements so as to access the appropriate schema.

We have described a rather simple implementation to illustrate the concept of using multiple schemas for multi-tenancy. In practice, web-application servers need to have schema names configured during deployment so that they can maintain database connection pools to each schema. Therefore, another level of indirection is usually required, where customer name (i.e. OU) is mapped to the actual schema name, so that customers can be added or deleted online without bringing the system down.

In the case of a multi-entity scenario within a single organization, the number of users was relatively small, probably in the thousands at most. For a SaaS application, the number of users will be orders of magnitude larger. Thus additional factors need to be considered for a multi-tenant SaaS deployment, such as how many applications server and database instances are needed, and how a large set of users are efficiently and dynamically mapped to OUs so as to be connected to the appropriate application server and database instance.

4.3 MULTI-TENANCY USING CLOUD DATA STORES

As discussed in the previous chapter, cloud data stores exhibit non-relational storage models. Furthermore, each of these data stores are built to be multi- tenant from scratch since effectively a single instance of such a large-scale distributed data store caters to multiple applications created by cloud users. For example, each user of the Google App Engine can create a fixed number of applications, and each of these *APPEARS* to have a separate data store;

Here we focus on a different problem: As a *user* (application developer) of a cloud platform , how does one create one's own multi-tenant application? In the case of Amazon EC2 the answer is straightforward; since this is an infrastructure cloud it gives users direct access to (virtual) servers where one can recreate exactly the same multi-tenant architectures discussed earlier using standard application servers and database systems.

However the situation is different using a PaaS platform such as Google's App Engine with its Datastore, Amazon's SimpleDB or even Azure's data ser- vices. For example, a single App Engine application has *one* data store name space, or schema (so, if we create one 'Customer' model, then we cannot have another by the same name in the same application). Thus, it appears at first that we are constrained to use the inefficient single schema approach.

However, an interesting feature of the Google Datastore is that entities are essentially *SCHEMA-LESS*. Thus, it is up to the *LANGUAGE* API provided to define how the data store is used. In particular, the Model

class in the Python API to App Engine is object-oriented as well as dynamic. Further, as Python is a completely interpretive language, fresh classes can be defined at runtime, along with their corresponding data store 'kinds.'

Figure 4.4 shows one possible implementation of multi-tenancy using multiple schemas with Google App Engine, in Python. Separate classes are instantiated for each schema, at runtime. This approach is similar to simulat- ing multiple schemas in a relational database by having table names that are schema dependent.

A similar strategy can be used with Amazon's SimpleDB, where *DOMAINS*, which play the role of tables in relational parlance and are the equivalent of

```
# Normal schema definition (not used)
#Class Customer(db.Model):
#  ID    = db.IntegerProperty()
#  Name = db.StringProperty()
  ... ...
# Dynamic OU specific classes for 'Customer'
for OU in OUList:
  #Gets ALL fields from meta-data
  schema=fetch_schema('Customer' OU)
  # Create OU specific class at run-time
  OUclass=type('Customer'+OU,(db.Model,), schema)
```

ID	*Name*	*Address*	*Hobby*	*Birthday*	*Value*

Customer 503

ID	Name	Address	Risk	Value

Customer 490

FIGURE 4.4. Multi-tenancy using Google Datastore

'kind' in the Google Datastore, can be created dynamically from any of the provided language APIs.

4.5 DATA ACCESS CONTROL FOR ENTERPRISE APPLICATIONS

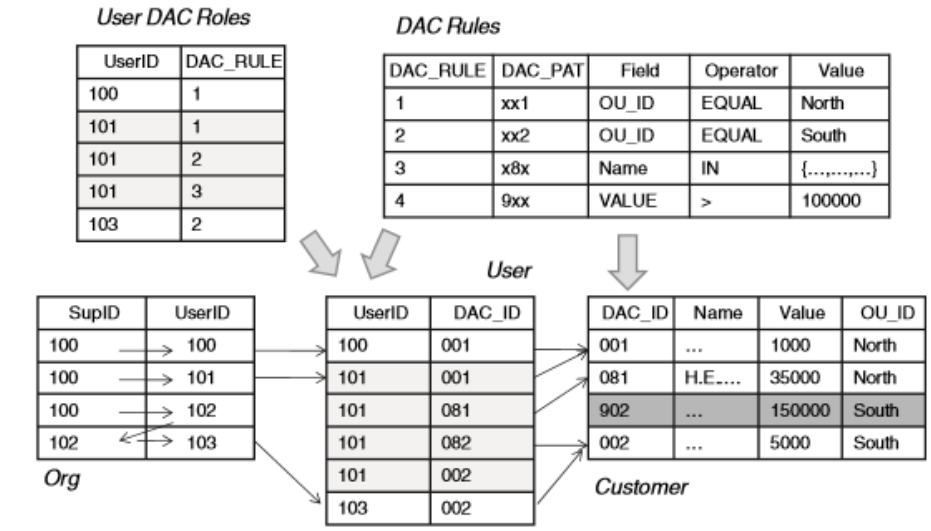
So far we have covered the typical strategies used to achieve multi-tenancy from the perspective of enabling a single application code base, running in a single instance, to work with data of multiple customers, thereby bringing down costs of management across a potentially large number of customers.

For the most part, multi-tenancy as discussed above appears to be of use primarily in a software as a service model. There are also certain cases where multi-tenancy can be useful within the enterprise as

well. We have already seen that supporting multiple entities, such as bank branches, is essentially a multi-tenancy requirement. Similar needs can arise if a workgroup level application needs to be rolled out to many independent teams, who usually do not need to share data. Customizations of the application schema may also be needed in such scenarios, to support variations in business processes. Similar requirements also arise in supporting multiple *LEGAL* entities each of which could be operating in different regulatory environments.

As we mentioned earlier, in a multi-entity scenario a subset of users may need to be given access to data from all branches, or a subset of branches, depending on their position in an organizational unit hierarchy. More gen- erally, access to data may need to be controlled based on the *VALUES* of any field of a table, such as high-value transactions being visible only to some users, or special customer names being invisible without explicit permission. Such requirements are referred to as *DATA ACCESS control* needs, which while common, are less often handled in a reusable and generic manner. Data access control (or DAC) is a generalization of multi-tenancy in that the latter can often be implemented using DAC. In Figure 4.5 we illustrate how data access control can be implemented in a generic manner within a single schema to support fairly general rules for controlling access to records based on *field VALUES*.

Each application table, such as Customer, is augmented with an additional field DAC_ID. The DAC Rules table lists patterns based on value ranges of arbitrary fields using which the values of the DAC_IDin each Customer record are filled through a batch process. Users are assigned privileges to access records satisfying one or more such DAC rules as specified in the User DAC Roles table. This information is expanded, via a batch process, to data



Using RDBMS

```
SELECT … FROM CUSTOMER AS C, USER AS U, ORG AS O1 OUTER JOIN ORG AS O2 WHERE
  O1.SupID = :user AND                    /*  current user
  O1.UserID = O2.SupID AND                /*  current user and supervisors
  O2.UserID=   U.UserID AND               /*  DAC_IDs of these users
  U.DAC_ID = C.DAC_ID                     /*  filter Customer table by these DAC_IDs
```

Using GQL queries on Google Datastore:

```
USER_LIST = fetch_team_for_user(:user)    /* execute self join on ORG in memory
DAC_LIST  = SELECT  DAC_ID  from USER WHERE UserID in USER_LIST
SELECT … FROM  CUSTOMER  WHERE  DAC_ID  IN DAC_LIST
```

FIGURE 4.5. Data access control

in the User table where there is a record for each value of DAC_IDthat a user can access.

For example, the user 101 has access to three DAC rules, which translate to five records in the User table. This calculation involves computing the complete set of mutually exclusive and unique DAC range combinations based on the DAC Rules and thereafter which subset of these a particular user has access to based on the User DAC Roles information; note that this computation is independent of the actual DAC_IDvalues in the Customer or other application tables.

It is straightforward to limit access to records of the Customer table to only those a particular user is permitted, as specified in the User table using a join. In the illustration of Figure 4.5, we introduce an additional complication where users are also given access to the DAC permissions of all their direct reports, as specified in the Org table.

In a traditional relational database, SQL queries on the Customer database can be modified to support data access control by introducing a generic join,

including a self-join on the Org table to find all direct reports of a user, which is then joined to the User table and the Customer table. However, in a cloud database, such as Google Datastore or Amazon's SimpleDB, *joins ARE not supported*. Therefore the same functionality must be implemented in code as shown in the figure: The self-join on Org is done in memory giving a list of reportees, including the user; this is used as a filter to get the permissible DAC_IDs from the User table. Finally this list is used to filter the application query on the Customer table.

It is important to note that when adding or modifying Customer records the DAC_ID needs to be recomputed based on the DAC Rules; this computation also needs to be optimized, especially if there are a large number of DAC Rules. Adding new DAC Rules or modifying existing ones will also require re-computation and updates to the DAC_IDvalues in the Customer table. Care also needs to be taken when filling the DAC Rules table to ensure that DAC ranges on the same field are always non-overlapping.

We thought it fit to cover data access control here, as part of our treatment of multi-tenancy, first because these requirements are closely related, but also to bring out the complexities of real enterprise applications even for incorporating a generic requirement such as data access control. Beyond the example itself the lesson to be learnt is that migrating applications to a multi- tenancy model, especially using cloud databases, is not a trivial task.

The emergence of cloud platforms has given rise to new paradigms for dealing with distributed data in the cloud, parallel computing using very large computing clusters as well as rapid application development tools for specialized domains: Cloud-based data stores differ significantly from tradi- tional relational databases, with different query and consistency semantics as well as performance behavior. The MapReduce programming paradigm makes large-scale analytics tasks easy to define. MapReduce implementations allow massive computing clusters to be used while tolerating faults that are inevitable at such scales. Similarly, but in a very different context, Dev 2.0 plat- forms allow simple business applications to be developed by end-users using always-on hosted platforms in the cloud, obviating the need for traditional development and thereby increasing business agility.

4.6 Data in the cloud

Since the 80s relational database technology has been the ‘default’ data storage and retrieval mechanism used in the vast majority of enterprise applications. The origins of relational databases, beginning with System R [5] and Ingres
[60] in the 70s, focused on introducing this new paradigm as a general purpose replacement for hierarchical and network databases, for the most common business computing tasks at the time, viz. transaction processing.

In the process of creating a planetary scale web search service, Google in particular has developed a massively parallel and fault tolerant distributed file system (GFS) along with a data organization (BigTable) and program- ming paradigm (MapReduce) that is markedly different from the traditional relational model. Such ‘cloud data strategies’ are particularly well suited for large-volume massively parallel text processing, as well as possibly other tasks, such as enterprise analytics. The public cloud computing offerings from Google (i.e. App Engine) as well as those from other vendors have made similar data models (Google’s Datastore, Amazon’s SimpleDB) and program- ming paradigms (Hadoop on Amazon’s EC2) available to users as part of their cloud platforms.

At the same time there have been new advances in building specialized database organizations optimized for analytical data processing, in particular column-oriented databases such as Vertica. It is instructive to note that the BigTable-based data organization underlying cloud databases exhibits some similarities to column-oriented databases. These concurrent trends along with the ease of access to cloud platforms are witnessing a resurgence of interest in non-relational data organizations and an exploration of how these can best be leveraged for enterprise applications.

In this chapter we examine the structure of Google App Engine's Data store and its underlying technologies, Google's distributed file system (GFS) and BigTable abstraction, as well as the open source project Hadoop's HBase and HDFS (clones of BigTable and GFS respectively).

4.7 RELATIONAL DATABASES

Before we delve into cloud data structures we first review traditional relational database systems and how they store data. Users (including application pro- grams) interact with an RDBMS via SQL; the database 'front-end' or parser transforms queries into memory and disk level operations to optimize execu- tion time. Data records are stored on pages of contiguous disk blocks, which are managed by the disk-space-management layer.

Pages are fetched from disk into memory buffers as they are requested, in many ways similar to the file and buffer management functions of the operating system, using pre-fetching and page replacement policies. However, database systems usually do not rely on the file system layer of the OS and instead manage disk space themselves. This is primarily so that the database can have full control of when to retain a page in memory and when to release it. The database needs be able to adjust page replacement policy when needed and pre-fetch pages from disk based on expected access patterns that can be very different from file operations. Finally, the operating system files used by databases need to span multiple disks so as to handle the large storage requirements of a database, by efficiently exploiting parallel I/O systems such as RAID disk arrays or multi-processor clusters.

The storage indexing layer of the database system is responsible for locating records and their organization on disk pages. Relational records (tabular rows) are stored on disk pages and accessed through indexes on specified columns, which can be B^+-tree indexes, hash indexes, or bitmap indexes [46]. Normally rows are stored on pages contiguously, also called a 'row-store', and indexed using B^+-trees. An index can be *PRIMARY*, in which case rows of the table are physically stored in as close as possible to sorted order based on the column specified by the index. Clearly only one such primary index is possible per table; the remaining indexes are *SECONDARY*, and maintain pointers to the actual row locations on disk.

While B^+-tree indexes on a row-store are optimal for write oriented work- loads, such as the case in transaction processing applications, these are not the best for applications where reads dominate; in the latter case bitmap indexes, cross-table indexes and materialized views are used for efficient access to records and their attributes. Further, a row-oriented storage of records on disk may also not be optimal for

read-dominated workloads, especially analytical applications. Recently column-oriented storage [61] has been proposed as a more efficient mechanism suited for analytical workloads, where an aggrega- tion of measures columns (e.g. Sales) need to be performed based on values of dimension columns (e.g. Month). Figure 4.6 illustrates the difference between row-oriented and column-oriented storage. Notice that in a column store, projections of the table are stored sorted by dimension values, which are themselves compressed (as bitmaps, for example) for ease of comparison as well as reduced storage. Notice also that the column store needs additional 'join indexes' that map the sort orders of different projections so as to be able to recover the original row when required. When the cardinality of dimension values are small these join indexes can also be efficiently compressed

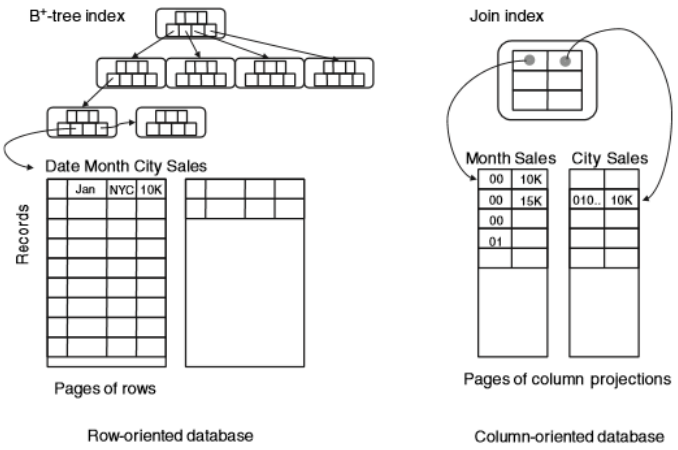


FIGURE 4.6. Row vs. column storage

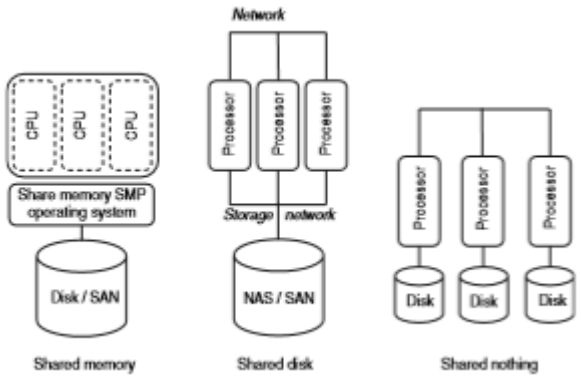


FIGURE 4.7. Parallel database architectures

(which we have not shown in the figure). When we cover cloud data stores later in this chapter we will see some of the similarities between their meta- data indexes and B^+-tree indexes, as well as between their data organization structures and column oriented databases.

Over the years database systems have evolved towards exploiting the parallel computing capabilities of

multi-processor servers as well as harnessing the aggregate computing power of clusters of servers connected by a high-speed network. Figure 4.7 illustrates three parallel/distributed database architectures: The shared memory architecture is for machines with many CPUs (and with each having possibly many processing 'cores') while the memory address space is shared and managed by a symmetric multi-processing operating sys- tem that schedules processes in parallel exploiting all the processors. The shared-nothing architecture assumes a cluster of independent servers each with its own disk, connected by a network. A shared-disk architecture is somewhere in between with the cluster of servers sharing storage through high-speed network storage, such as a NAS (network attached storage) or a SAN (storage area network) interconnected via standard Ethernet, or faster Fiber Channel or Infiniband connections. Parallel database systems capable of exploiting any of these parallel architectures have existed since the 80s. These systems parallelize SQL queries to execute efficiently and exploit multiple processors [46]. In the case of shared-nothing architectures, tables are par- titioned and distributed across processing nodes with the SQL optimizer handling distributed joins as best possible. Each of the traditional transaction- processing databases, Oracle, DB2 and SQL Server support parallelism in various ways, as do specialized systems designed for data warehousing such as Vertica, Netezza and Teradata.

Traditional relational databases are designed to support high-volume trans- action processing involving many, possibly concurrent, record level insertions and updates. Supporting concurrent access while ensuring that conflicting actions by simultaneous users do not result in inconsistency is the responsi- bility of the transaction management layer of the database system that ensures 'isolation' between different transactions through a variety of locking strate- gies. In the case of parallel and distributed architectures, locking strategies are further complicated since they involve communication between processors via the well-known 'two-phase' commit protocol [46].

It is important to note that the parallel database systems developed as exten- sions to traditional relational databases were designed either for specially constructed parallel architectures, such as Netezza, or for closely coupled clusters of at most a few dozen processors. At this scale, the chances of system failure due to faults in any of the components could be sufficiently compensated for by transferring control to a 'hot-standby' system in the case of transaction processing or by restarting the computations in the case of data warehousing applications. As we shall see below, a different approach is required to exploit a parallelism at a significantly larger scale.

4.8 CLOUD FILE SYSTEMS: GFS AND HDFS

The Google File System (GFS) [26] is designed to manage relatively large files using a very large distributed cluster of commodity servers connected by a high-speed network. It is therefore designed to (a) expect and tolerate hardware failures, even during the reading or writing of an individual file (since files are expected to be very large) and (b) support parallel reads, writes and appends by multiple client programs. A common use case that is efficiently supported is that of many 'producers'

appending to the same file in parallel, which is also being simultaneously read by many parallel 'consumers'.

As a result they also do not scale as well as data organizations built on GFS-like platforms such as the Google Datastore. The Hadoop Dis- tributed File System (HDFS) is an open source implementation of the GFS architecture that is also available on the Amazon EC2 cloud platform; we refer to both GFS and HDFS as 'cloud file systems.'

The architecture of cloud file systems is illustrated in Figure 10.3. Large files are broken up into 'chunks' (GFS) or 'blocks' (HDFS), which are themselves large (64MB being typical). These chunks are stored on commodity (Linux) servers called Chunk Servers (GFS) or Data Nodes (HDFS); further each chunk is replicated at least three times, both on a different physical rack as well as a different network segment in anticipation of possible failures of these components apart from server failures.

When a client program ('cloud application') needs to read/write a file, it sends the full path and offset to the Master (GFS) which sends back meta-data

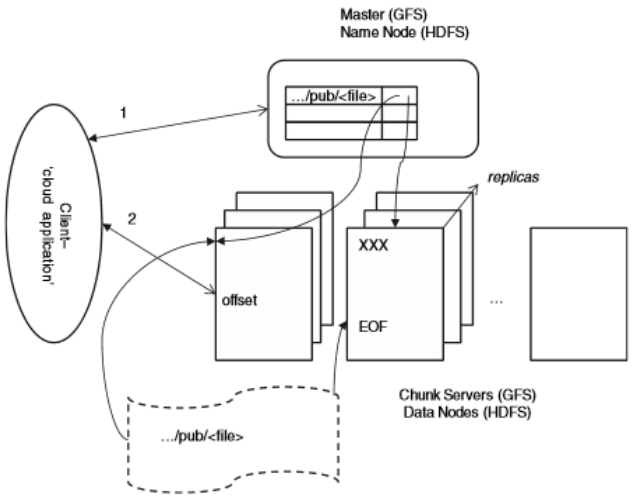


FIGURE 4.8. Cloud file systems

for one (in the case of read) or all (in the case of write) of the replicas of the chunk where this data is to be found. The client caches such meta-data so that it need not contact the Master each time. Thereafter the client directly reads data from the designated chunk server; this data is not cached since most reads are large and caching would only complicate writes.

In case of a write, in particular an append, the client sends only the data to be appended to all the chunk servers; when they all acknowledge receiving this data it informs a designated 'primary' chunk server, whose identity it receives (and also caches) from the Master. The primary chunk server appends its copy of data into the chunk at an offset of its choice; note that this may be beyond the EOF to account for multiple writers who may be appending to this file simultaneously. The primary then

forwards the request to all other replicas which in turn write the data at the same offset if possible or return a failure. In case of a failure the primary rewrites the data at possibly another offset and retries the process.

The Master maintains regular contact with each chunk server through heartbeat messages and in case it detects a failure its meta-data is updated to reflect this, and if required assigns a new primary for the chunks being served by a failed chunk server. Since clients cache meta-data, occasionally they will try to connect to failed chunk servers, in which case they update their meta-data from the master and retry.

In [26] it is shown that this architecture efficiently supports multiple par- allel readers and writers. It also supports writing (appending) and reading the same file by parallel sets of writers and readers while maintaining a consistent view, i.e. each reader always sees the same data regardless of the replica it happens to read from. Finally, note that computational processes (the 'client' applications above) run on the same set of servers that files are stored on. As a result, distributed programming systems, such as MapReduce, can often schedule tasks so that their data is found locally as far as possible, as illustrated by the Clustera system [16].

4.9 BIGTABLE, HBASE

BigTable [9] is a distributed *structured* storage system built on GFS; Hadoop's HBase is a similar open source system that uses HDFS. A BigTable is essentially a sparse, distributed, persistent, multidimensional sorted 'map.'[1] Data in a

[1] In the programming language sense, i.e. a dictionary of key-value pairs.

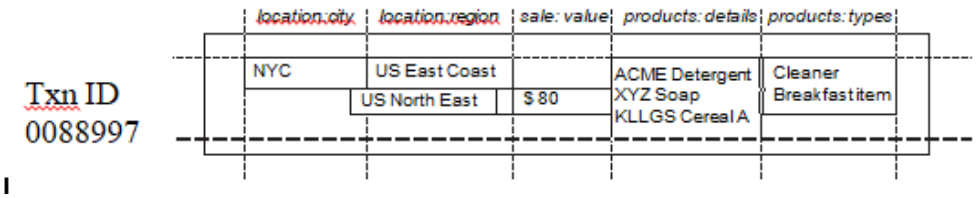


FIGURE 4.9. Data organization in a BigTable

BigTable is accessed by a row key, column key and a timestamp. Each col- umn can store arbitrary name–value pairs of the form *COLUMN-FAMILY:LABEL*, *string*. The set of possible column-families for a table is fixed when it is cre- ated whereas columns, i.e. labels within the column family, can be created dynamically at any time. Column families are stored close together in the distributed file system; thus the BigTable model shares elements of column- oriented databases. Further, each Bigtable cell (row, column) can contain multiple versions of the data that are stored in decreasing timestamp order. We illustrate these features below through an example.

Figure 4.9 illustrates the BigTable data structure: Each row stores infor- mation about a specific sale transaction and the row key is a transaction identifier. The 'location' column family stores

columns relating to where the sale occurred, whereas the 'product' column family stores the actual prod- ucts sold and their classification. Note that there are two values for region having different timestamps, possibly because of a reorganization of sales regions. Notice also that in this example the data happens to be stored in a de-normalized fashion, as compared to how one would possibly store it in a relational structure; for example the fact that XYZ Soap is a Cleaner is not maintained.

Since data in each column family is stored together, using this data orga- nization results in efficient data access patterns depending on the nature of analysis: For example, only the *LOCATION* column family may be read for tra- ditional data-cube based analysis of sales, whereas only the *product* column family is needed for say, market-basket analysis. Thus, the BigTable structure can be used in a manner similar to a column-oriented database.

Figure 4.10illustrates how BigTable tables are stored on a distributed file system such as GFS or HDFS. (In the discussion below we shall use BigTable terminology; equivalent HBase terms are as shown in Figure 4.11) Each table is split into different row ranges, called tablets. Each tablet is managed by a tablet server that stores each column family for the given row range in a separate distributed file, called an SSTable. Additionally, a single Metadata table is managed by a meta-data server that is used to locate the tablets of any user table in response to a read or write request. The Metadata table itself can be large and is also split into tablets, with the root tablet being special in that it points to the locations of other meta-data tablets. It is instructive to notice how this multi-layer structure is in many ways similar to a distributed B^+-tree index on the row keys of all tables.

BigTable and HBase rely on the underlying distributed file systems GFS and HDFS respectively and therefore also inherit some of the properties of these systems. In particular large parallel reads and inserts are efficiently sup- ported, even simultaneously on the same table, unlike a traditional relational database. In particular, reading all rows for a small number of column fami- lies from a large table, such as in aggregation queries, is efficient in a manner similar to column-oriented databases. Random writes translate to data inserts since multiple versions of each cell are maintained, but are less efficient since cell versions are stored in descending order and such inserts require more work than simple file appends. Similarly, the consistency properties of large parallel inserts are stronger than that for parallel random writes, as is pointed out in [26]. Further, writes can even fail if a few replicas are unable to write even if other replicas are successfully updated.

4.10 CLOUD DATA STORES: DATASTORE AND SIMPLEDB

The Google and Amazon cloud services do not directly offer BigTable and Dynamo to cloud users. Using Hadoop's HDFS and HBase, which are avail- able as Amazon AMIs, users can set up their own BigTable-like stores on Amazon's EC2. However, as we have seen in Chapter 5, Google and Amazon both offer simple key-value pair database stores, viz. Google App Engine's Datastore and Amazon's SimpleDB. Here we describe how the Google Datas- tore is implemented using an underlying BigTable infrastructure, as illustrated in Figure 10.7.

All entities (objects) in a Datastore reside in one BigTable table, the Entities table having *one* column family. In addition to the single Entities table there are index tables that are used to support efficient queries.

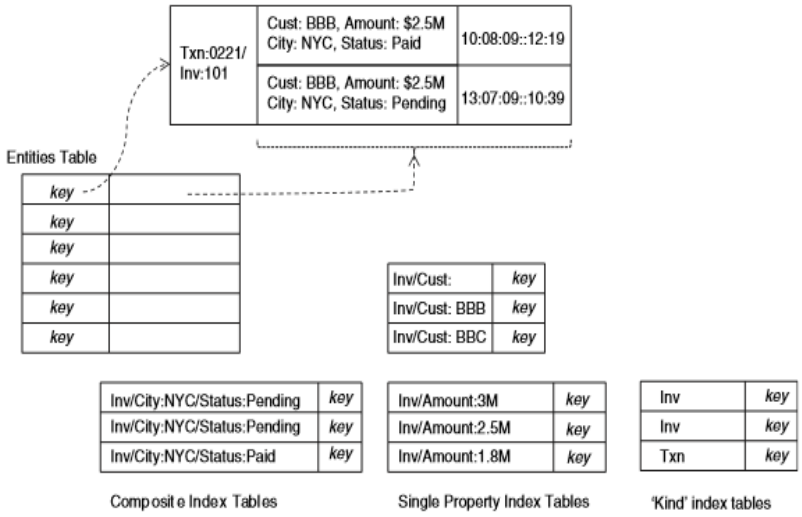


FIGURE 10.7. Google Datastore using BigTable

For the purposes of this discussion, it is useful to think of a BigTable as an array that has been horizontally partitioned (also called 'sharded') across disks, and *sorted* lexicographically by key values. In addition to single key lookup, this structure also enables highly efficient execution of *prefix* and *RANGE* queries on key values, e.g. all keys having prefix 'Txn,' or in the range 'Amount:1M' to 'Amount:3M.' From this feature derive the key structures of entity and index tables that implement Datastore.

Recall from Chapter 5 that Datastore allows entities to be 'grouped' (entity groups) for the purposes of transactions and also efficiency. Entity keys are lexicographic by group ancestry, so if the entity Inv:101 has parent Txn:0221, then its key in the entities table would look like 'Txn:0221/Inv:101,' which is lexicographically near that of its parent 'Txn:0221.' As a result these enti- ties end up being stored close together on disk, whereas another group, say 'Txn:9999' and 'Txn:9999/Inv:875' may be far apart from these. Notice that this is very different from a relational model where the location of records from Txn and Inv tables on disk would be unrelated to any foreign key relationships between such records.

Index tables support the variety of queries included in the Datastore API. Some indexes, such as single property indexes and 'kind' indexes are auto- matically created when a 'kind' is defined, whereas others such as composite indexes need to be created by developers. Index tables use values of entity attributes as keys, e.g. the index entry 'Inv/Cust:BBB' allows efficient lookup of the record with the WHERE clause 'Cust = BBB'. Similarly, the index on Amount enables efficient range queries such as 'Amount ≥ 2M AND Amount ≥ 3M'. 'Kind' indexes

support queries of the form SELECT ALL Invoices, retrieving all entries of a given 'kind.' Composite indexes support more complex queries such as retrieving all invoices 'WHERE City=NYC AND Sta- tus=Pending.' In cases where composite indexes are not present (because developers did not provide them), single property indexes are used in such queries with the results being merged in memory. A query is executed by choosing indexes with highest selectivity first, which is efficient unless the selectivity of ALL of the query parameters is low (e.g. 'Status=Pending'). Notice that index operations are efficient precisely because of the efficiency of prefix
and range scans in BigTable.

The entities table stores multiple versions of each entity, primarily in order to support transactions spanning updates of different entities in the same group. Only one of the versions of each entity is tagged as 'committed,' and this is updated only when a transaction succeeds on all the entities in the group; journal entries consisting of previous entity versions are used to rollback if needed.

Note that many App Engine applications are small, so it is likely that a large number of tables within the App Engine Datastore have only a single tablet; in this case the structure of BigTable allows many tablets possibly from different applications to share GFS files (SSTables), each of which may have only one or just a few chunks.

Notice also that this mapping of the Datastore API onto BigTable does *not* exploit the column-oriented storage of BigTable, since a single column family is used. Thus while BigTable (and HBase) are potentially useful in large analytical tasks for the same reason that column-oriented databases are, Datastore as an application of BigTable does not share this property. Datastore is much more suited to transactional key-value pair updates, in much the same manner as Amazon's SimpleDB is, with the difference that its consistency properties are stronger (as compared to the 'eventual' consistency of SimpleDB), at the cost of a fixed overhead even for small writes.

While it is widely speculated that SimpleDB is similarly based on an under- lying Dynamo infrastructure, and its eventual consistency model substantiates this claim, there is no published information that confirms this. Nevertheless, as both Datastore and SimpleDB share many similarities, we expect that our discussion of Datastore may also shed some light on how SimpleDB might be implemented on Dynamo. While SimpleDB objects are easily mapped to Dynamo objects, indexes are probably maintained differently since Dynamo does not share any underlying sorted array structure similar to BigTable that can make queries efficient.

Question Bank

Part A

1. Define multitenant.
2. Disadvantage of multitenant
3. Define DAC.
4. Compare relational database with parallel database.
5. Write about cloud file system GFS and HDFS
6. What is bigtable?

Part B

1. Describe Multi-entity implementation in detail.
2. Explain in detail Multi-tenancy using a single schema with appropriate table.
3. Explain in detail Multi-tenancy using multiple schemas with appropriate table.
4. Compare and contrast Multi-tenancy using multiple schemas with Multi-tenancy using a single schema.
5. Describe multi-tenancy using cloud data stores
6. Explain Data access control (DAC)for enterprise applications.
7. Explain in detail Cloud file systems: GFS and HDFS
8. Describe Cloud data stores: datastore and simpledb

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – V – Cloud Computing – SBS1207

Unit - V

Amazon web services: Compute services - Storage services - Communication services - Additional services. Google AppEngine: Architecture and core concepts - Application life cycle - Cost model – Observations. Microsoft azure: Azure core concepts - SQL azure - Windows azure platform appliance.

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 5.1 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 struc- tured 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

com- plementary services.

4.1.0 Compute services

Compute services constitute the fundamental element of cloud computing systems. The fundamen- tal service in this space is Amazon EC2, which delivers an IaaS solution that has served as a refer- ence 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 con- figured 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.

4.1.0.1 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-xxxxxx* and

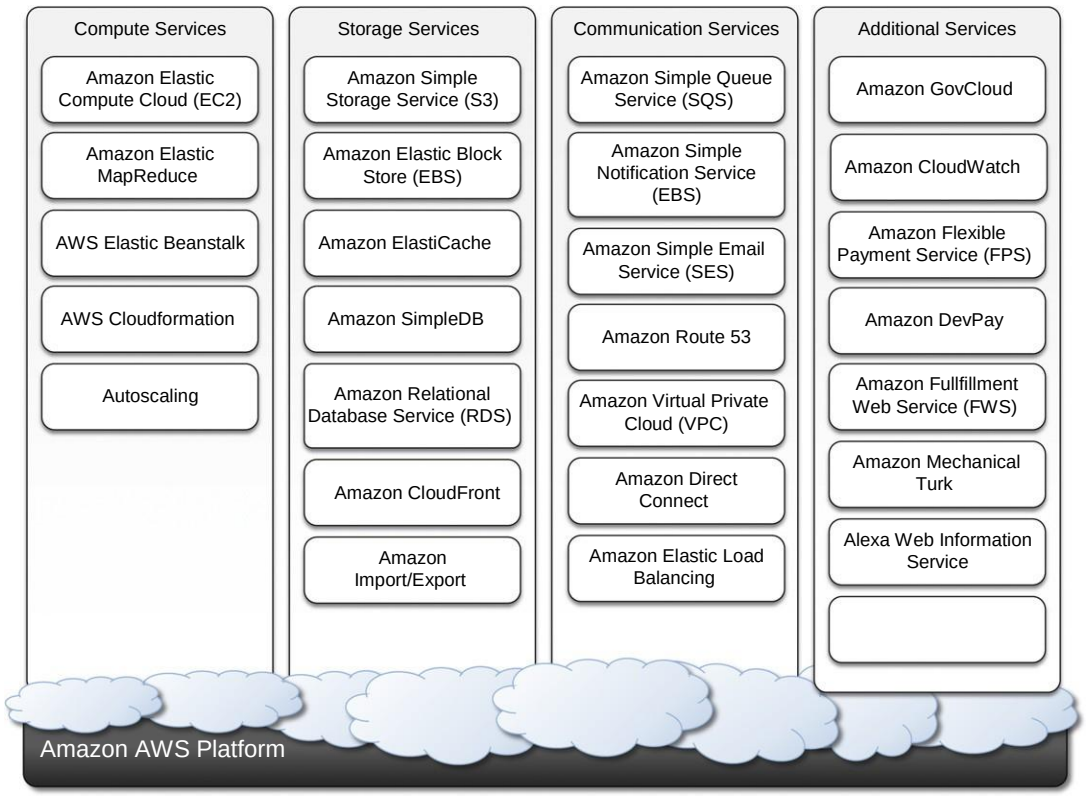


FIGURE 5.1 Amazon Web Services ecosystem.

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 prac- tice 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.

4.1.0.2 EC2 instances

EC2 instances represent virtual machines. They are created using AMI as templates, which are spe- cialized 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 ECUis 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 fre- quency 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 5.2 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 vola- tile than normal instances; whereas for normal instances EC2 will try as much as possible to keep

Table 5.2 Amazon EC2 (On-Demand) Instances Characteristics

Instance Type	ECU	Platform	Memory	Disk Storage	Price (U.S. East) (USD/hour)
Standard instances					
Small	1(1 3 1)	32 bit	1.7 GB	160 GB	$0.085 Linux $0.12 Windows
Large	4(2 3 2)	64 bit	7.5 GB	850 GB	$0.340 Linux $0.48 Windows
Extra Large	8(4 3 2)	64 bit	15 GB	1,690 GB	$0.680 Linux $0.96 Windows
Micro instances					
Micro	,5 2	32/64 bit	613 MB	EBS Only	$0.020 Linux $0.03 Windows
High-Memory instances					
Extra Large	6.5(2 3 3.25)	64 bit	17.1 GB	420 GB	$0.500 Linux $0.62 Windows
Double Extra Large	13(4 3 3.25)	64 bit	34.2 GB	850 GB	$1.000 Linux $1.24 Windows
Quadruple Extra Large	26(8 3 3.25)	64 bit	68.4 GB	1,690 GB	$2.000 Linux $2.48 Windows
High-CPU instances					
Medium	5(2 3 2.5)	32 bit	1.7 GB	350 GB	$0.170 Linux $0.29 Windows
Extra Large	20(8 3 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.

4.1.0.3 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, attach- ing 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 com- municating 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 dif- ferently: 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 con- trol 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.

4.1.0.4 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 conve- niently package their Web application into a WAR file and use Beanstalk to automate its deploy- ment 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 underly- ing 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 infrastruc- ture 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 com- posed 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 clus- ter (Small, High-Memory, High-CPU, Cluster Compute, and Cluster GPU). On top of these ser- vices, basic Web applications allowing users to quickly run data-intensive applications without writing code are offered.

4.1.1 Storage services

AWS provides a collection of services for data storage and information management. The core ser- vice 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 rep- resent 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.

4.1.1.1 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 distrib- uted 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 tempo- rary access token.

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 con- figuration 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* proto- col. 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.

4.1.1.2 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 differ- ent 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 occu- pied 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.

4.1.1.3 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 administra- tive 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 5.3 provides an overview of the pricing options.

The prices indicated in Table 5.3 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.

4.1.1.4 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 5.3 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 3 1)	64 bit	1.3 GB	Moderate	$0.095
Large	4(2 3 2)	64 bit	7.1 GB	High	$0.380
Extra Large	8(4 3 2)	64 bit	14.6 GB	High	$0.760
High-Memory instances					
Extra Large	6.5(2 3 3.25)	64 bit	16.7 GB	High	$0.560
Double Extra Large	13(4 3 3.25)	64 bit	33.8 GB	High	$1.120
Quadruple Extra Large	26(8 3 3.25)	64 bit	68 GB	High	$2.240
High-CPU instances					
Extra Large	26(8 3 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 solu- tion is implemented by keeping in standby synchronized copies of the services in different avail- ability 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 cut- ting down the response time of the service.

The available options and the relative pricing of the service during ~~20~~11 2012 are shown in Table 5.4. The table shows the costing details of the on-demand instances. There is also the possi- bility of using reserved instances for long terms (one to three years) by paying up-front at dis- counted 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 infra- structure and require a scalable database solution.

Amazon SimpleDB

Amazon SimpleDB is a lightweight, highly scalable, and flexible data storage solution for applica- tions 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 5.4 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 3 1)	64 bit	1.7 GB	Moderate	$0.11
Large	4(2 3 2)	64 bit	7.5 GB	High	$0.44
Extra Large	8(4 3 2)	64 bit	15 GB	High	$0.88
High-Memory instances					
Extra Large	6.5(2 3 3.25)	64 bit	17.1 GB	High	$0.65
Double Extra Large	13(4 3 3.25)	64 bit	34 GB	High	$1.30
Quadruple Extra Large	26(8 3 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: 5, !5, ,, -, ,5, .5, *like*, *not like*, *between*, *is null*, *is not null*, and *every()*. Here is a simple example on how to query data:

select [x] from domain_name where every(attribute_name) 5 ‘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 rea- sonable, 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 5.5 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 semistruc- tured collections of small objects and not for being a long-term storage option for large objects.

4.1.1.5 *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 5.5 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 pos- sible 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 dis- tributed 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 5.6 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 distribu- tion of very popular content that is frequently downloaded, potentially from the entire globe and not only the Amazon network.

4.1.2 Communication services

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

4.1.2.1 Virtual networking

Virtual networking comprises a collection of services that allow AWS users to control the connec- tivity to and between compute and storage services. *Amazon Virtual Private Cloud (VPC)* and *Amazon Direct Connect* provide connectivity solutions in terms of infrastructure; *Route 53* facili- tates connectivity in terms of naming.

Table 5.6 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	$0.120/GB	$0.148/GB	$0.200/GB
Next 100 TB/month	$0.060/GB	$0.060/GB	$0.080/GB	$0.127/GB	$0.180/GB
Next 350 TB/month	$0.040/GB	$0.040/GB	$0.060/GB	$0.106/GB	$0.160/GB
Next 524 TB/month	$0.030/GB	$0.030/GB		$0.085/GB	$0.140/GB
Next 4 PB/month	$0.025/GB	$0.025/GB		$0.075/GB	$0.130/GB
Greater than 5 PB/month	$0.020/GB	$0.020/GB		$0.065/GB	$0.125/GB

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 pri- vate 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).

4.1.2.2 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 mes- sage 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 infra- structure. 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 specify- ing email headers and Multipurpose Internet Mail Extension (MIME) types. Emails are queued for delivery, and the users are notified of any failed delivery. SES also provides a wide range of statis- tics 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.

4.1.3 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 devel- opers understand and optimize the behavior of their application hosted on AWS. CloudWatch col- lects 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.

4.1.4 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.

5.2 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.

4.2.1 Architecture and core concepts

AppEngine is a platform for developing scalable applications accessible through the Web (see Figure 5.2). The platform is logically divided into four major components: infrastructure, the run- time environment, the underlying storage, and the set of scalable services that can be used to develop applications.

4.2.1.1 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 pro- cesses the request, evaluates their load, and, if necessary, allocates additional resources (i.e., ser- vers) 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 sta- tistics on which the billing is calculated.

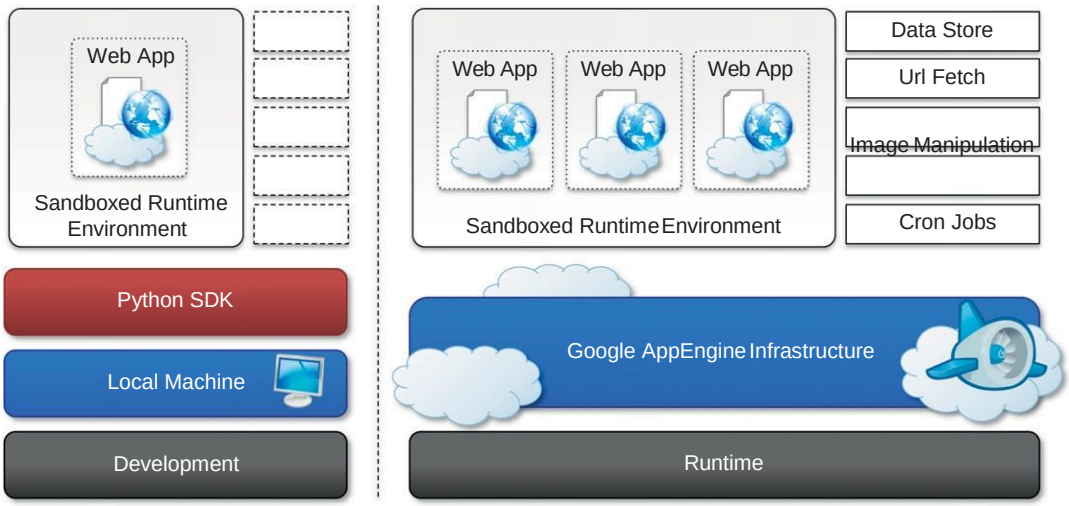


FIGURE 5.2 Google AppEngine platform architecture.

4.2.1.2 *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 environ- ment 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 applica- tion 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 appli- cation 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 implementa- tions 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 run- time 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 appli- cations 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.

4.2.1.3 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 con- tent, and users can specify how dynamic content should be served when uploading their applica- tions 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 data- base 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 applica- tions 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 proper- ties of the same name might contain values of different types. Moreover, properties can store differ- ent 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 con- text 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.

4.2.1.4 Application SERVICES

Applications hosted on AppEngine take the most from the services made available through the run- time 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 gen- erated 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 ref- erence 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 fol- lowing up with users about operations performed by the application. Email can also be used to trig- ger 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 interac- tions, XMPP can be conveniently used to connect the Web application with chat bots or to imple- ment 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 implementa- tion 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.

4.2.1.5 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 per- formed 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 con-figurable 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 per- formed 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 han- dler 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.

4.2.2 Application life cycle

AppEngine provides support for almost all the phases characterizing the life cycle of an applica-tion: 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.

4.2.2.1 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 possi- ble 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 avail- able 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 con- sole 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 pre- view 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 develop- ment 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 reimplemen- ted 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.

4.2.2.2 Application deployment and management

Once the application has been developed and tested, it can be deployed on AppEngine with a sim- ple 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 appli- cations. 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.

4.2.3 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.

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 iden- tify 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, subse- quent 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 use- ful feedback to users.

Resources and services quotas are organized into free default quotas and billing-enabled default quo- tas. 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.

4.2.4 Observations

AppEngine, a framework for developing scalable Web applications, leverages Google's infrastruc- ture. 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 applica- tions 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 con- forms with the AppEngine application model.

5.3 Microsoft Azure

Microsoft Windows Azure is a cloud operating system built on top of Microsoft datacenters' infra- structure and provides developers with a collection of services for building applications with cloud technology. Services range from compute, storage, and networking to application connectivity,

access control, and business intelligence. Any application that is built on the Microsoft technology can be scaled using the Azure platform, which integrates the scalability features into the common Microsoft technologies such as Microsoft Windows Server 2008, SQL Server, and ASP.NET.

Figure 5.3 provides an overview of services provided by Azure. These services can be managed and controlled through the *Windows Azure Management Portal*, which acts as an administrative console for all the services offered by the Azure platform. In this section, we present the core fea- tures of the major services available with Azure.

4.3.1 Azure core concepts

The Windows Azure platform is made up of a foundation layer and a set of developer services that can be used to build scalable applications. These services cover compute, storage, networking, and identity management, which are tied together by middleware called *AppFabric*. This scalable com- puting environment is hosted within Microsoft datacenters and accessible through the Windows Azure Management Portal. Alternatively, developers can recreate a Windows Azure environment (with limited capabilities) on their own machines for development and testing purposes. In this sec- tion, we provide an overview of the Azure middleware and its services.

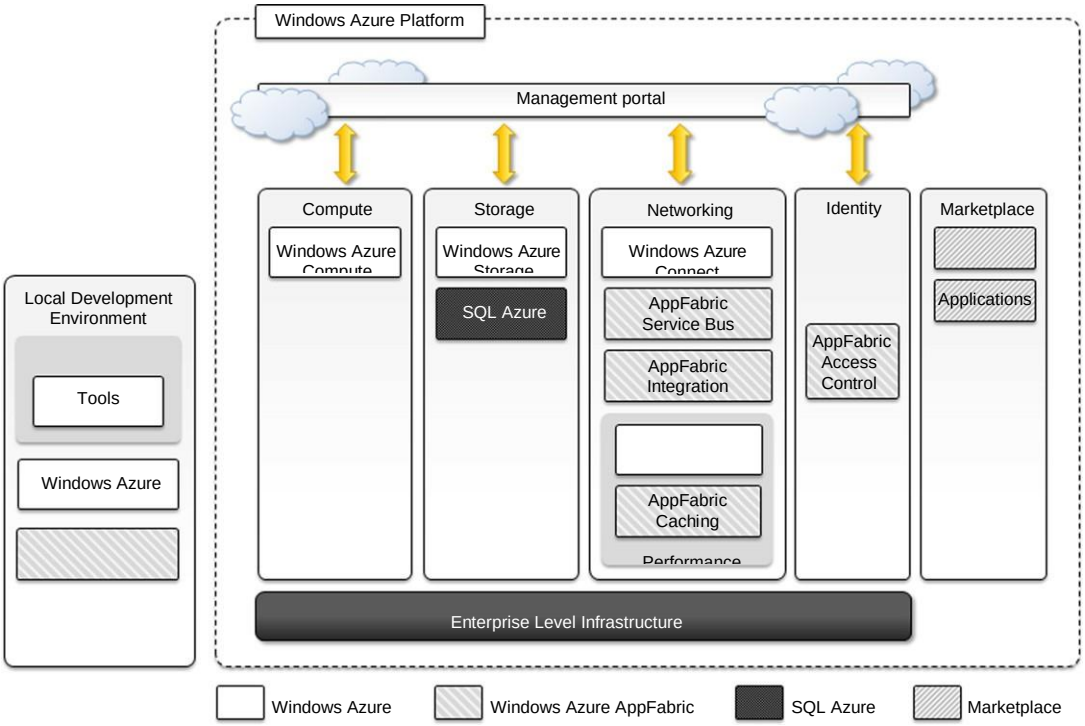


FIGURE 5.3 Microsoft Windows Azure Platform Architecture.

4.3.1.1 Compute SERVICES

Compute services are the core components of Microsoft Windows Azure, and they are delivered by means of the abstraction of *roles*. A role is a runtime environment that is customized for a specific compute task. Roles are managed by the Azure operating system and instantiated on demand in order to address surges in application demand. Currently, there are three different roles: *Web role*, *Worker role*, and *Virtual Machine (VM) role*.

Web role

The *Web role* is designed to implement scalable Web applications. Web roles represent the units of deployment of Web applications within the Azure infrastructure. They are hosted on the IIS 7 Web Server, which is a component of the infrastructure that supports Azure. When Azure detects peak loads in the request made to a given application, it instantiates multiple Web roles for that applica- tion and distributes the load among them by means of a load balancer.

Since version 3.5, the .NET technology natively supports Web roles; developers can directly develop their applications in Visual Studio, test them locally, and upload to Azure. It is possible to develop ASP.NET (*ASP.NET Web Role* and *ASP.NET MVC 2 Web Role*) and WCF (*WCF Service Web Role*) applications. Since IIS 7 also supports the PHP runtime environment by

means of the FastCGI module, Web roles can be used to run and scale PHP Web applications on Azure (*CGI Web Role*). Other Web technologies that are not integrated with IIS can still be hosted on Azure (i.e., Java Server Pages on Apache Tomcat), but there is no advantage to using a Web role over a Worker role.

Worker role

Worker roles are designed to host general compute services on Azure. They can be used to quickly provide compute power or to host services that do not communicate with the external world through HTTP. A common practice for Worker roles is to use them to provide background proces- sing for Web applications developed with Web roles.

Developing a worker role is like a developing a service. Compared to a Web role whose compu- tation is triggered by the interaction with an HTTP client (i.e., a browser), a Worker role runs con- tinuously from the creation of its instance until it is shut down. The Azure SDK provides developers with convenient APIs and libraries that allow connecting the role with the service pro- vided by the runtime and easily controlling its startup as well as being notified of changes in the hosting environment. As with Web roles, the .NET technology provides complete support for Worker roles, but any technology that runs on a Windows Server stack can be used to implement its core logic. For example, Worker roles can be used to host Tomcat and serve JSP-based applications.

Virtual machine role

The *Virtual Machine role* allows developers to fully control the computing stack of their compute service by defining a custom image of the Windows Server 2008 R2 operating system and all the service stack required by their applications. The Virtual Machine role is based on the Windows Hyper-V virtualization technology (see Section 3.6.3), which is natively integrated in the Windows server technology at the base of Azure. Developers can image a Windows server installation com- plete with all the required applications and components, save it into a Virtual Hard Disk (VHD)file, and upload it to Windows Azure to create compute instances on demand. Different types of instances are available, and Table 5.7 provides an overview of the options offered during 2011 2012.

Compared to the Worker and Web roles, the VM role provides finer control of the compute ser- vice and resource that are deployed on the Azure Cloud. An additional administrative effort is required for configuration, installation, and management of services.

Table 5.7 Windows Azure Compute Instances Characteristics, 2011—2012

Compute Instance Type	CPU	Memory	Instance Storage	I/O Performance	Hourly Cost (USD)
Extra Small	1.0 GHz	768 MB	20 GB	Low	$0.04
Small	1.6 GHz	1.75 GB	225 GB	Moderate	$0.12
Medium	2 3 1.6 GHz	3.5 GB	490 GB	High	$0.24
Large	4 3 1.6 GHz	7 GB	1,000 GB	High	$0.48
Extra Large	8 3 1.6 GHz	14 GB	2,040 GB	High	$0.96

4.3.1.2 Storage SERVICES

Compute resources are equipped with local storage in the form of a directory on the local file sys- tem that can be used to temporarily store information that is useful for the current execution cycle of a role. If the role is restarted and activated on a different physical machine, this information is lost.

Windows Azure provides different types of storage solutions that complement compute services with a more durable and redundant option compared to local storage. Compared to local storage, these services can be accessed by multiple clients at the same time and from everywhere, thus becoming a general solution for storage.

Blobs

Azure allows storing large amount of data in the form of binary large objects (BLOBs) by means of the *blobs* service. This service is optimal to store large text or binary files. Two types of blobs are available:

- *Block blobs.* Block blobs are composed of blocks and are optimized for sequential access; therefore they are appropriate for media streaming. Currently, blocks are of 4 MB, and a single block blob can reach 200 GB in dimension.
- *Page blobs.* Page blobs are made of pages that are identified by an offset from the beginning of the blob. A page blob can be split into multiple pages or constituted of a single page. This type of blob is optimized for random access and can be used to host data different from streaming. Currently, the maximum dimension of a page blob can be 1 TB.

Blobs storage provides users with the ability to describe the data by adding metadata. It is also possible to take snapshots of a blob for backup purposes. Moreover, to optimize its distribution, blobs storage can leverage the Windows Azure CDN so that blobs are kept close to users requesting them and can be served efficiently.

Azure drive

Page blobs can be used to store an entire file system in the form of a single *Virtual Hard Drive (VHD)* file. This can then be mounted as a part of the NTFS file system by Azure compute

resources, thus providing persistent and durable storage. A page blob mounted as part of an NTFS tree is called an *Azure Drive*.

Tables

Tables constitute a semistructured storage solution, allowing users to store information in the form of entities with a collection of properties. Entities are stored as rows in the table and are identified by a key, which also constitutes the unique index built for the table. Users can insert, update, delete, and select a subset of the rows stored in the table. Unlike SQL tables, there are no schema enforcing constraints on the properties of entities and there is no facility for representing relation- ships among entities. For this reason, tables are more similar to spreadsheets rather than SQL tables.

The service is designed to handle large amounts of data and queries returning huge result sets. This capability is supported by partial result sets and table partitions. A partial result set is returned together with a continuation token, allowing the client to resume the query for large result sets. Table partitions allow tables to be divided among several servers for load-balancing purposes. A partition is identified by a key, which is represented by three of the columns of the table.

Currently, a table can contain up to 100 TB of data, and rows can have up to 255 properties, with a maximum of 1 MB for each row. The maximum dimension of a row key and partition keys is 1 KB.

Queues

Queue storage allows applications to communicate by exchanging messages through durable queues, thus avoiding lost or unprocessed messages. Applications enter messages into a queue, and other applications can read them in a first-in, first-out (FIFO) style.

To ensure that messages get processed, when an application reads a message it is marked as invisible; hence it will not be available to other clients. Once the application has completed proces- sing the message, it needs to explicitly delete the message from the queue. This two-phase process ensures that messages get processed before they are removed from the queue, and the client failures do not prevent messages from being processed. At the same time, this is also a reason that the queue does not enforce a strict FIFO model: Messages that are read by applications that crash dur- ing processing are made available again after a timeout, during which other messages can be read by other clients. An alternative to reading a message is *peeking*, which allows retrieving the mes- sage but letting it stay visible in the queue. Messages that are peeked are not considered processed. All the services described are geo-replicated three times to ensure their availability in case of major disasters. *Geo-replication* involves the copying of data into a different datacenter that is hun-
dreds or thousands of miles away from the original datacenter.

4.3.1.3 Core infrastructure: AppFabric

AppFabric is a comprehensive middleware for developing, deploying, and managing applica- tions on the cloud or for integrating existing applications with cloud services. AppFabric imple- ments an optimized infrastructure supporting scaling out and high availability; sandboxing and multitenancy; state management; and dynamic address resolution and routing. On top of this infra- structure, the middleware offers a collection of services that simplify many of the common tasks in a distributed application, such as communication, authentication and authorization, and data access. These services are available through language-agnostic interfaces, thus allowing developers to build heterogeneous applications.

Access control

AppFabric provides the capability of encoding access control to resources in Web applications and services into a set of rules that are expressed outside the application code base. These rules give a great degree of flexibility in terms of the ability to secure components of the application and define access control policies for users and groups.

Access control services also integrate several authentication providers into a single coherent identity management framework. Applications can leverage Active Directory, Windows Live, Google, Facebook, and other services to authenticate users. This feature also allows easy building of hybrid systems, with some parts existing in the private premises and others deployed in the pub- lic cloud.

Service bus

Service Bus constitutes the messaging and connectivity infrastructure provided with AppFabric for building distributed and disconnected applications in the Azure Cloud and between the private pre- mises and the Azure Cloud. Service Bus allows applications to interact with different protocols and patterns over a reliable communication channel that guarantees delivery.

The service is designed to allow transparent network traversal and to simplify the development of loosely coupled applications, without renouncing security and reliability and letting developers focus on the logic of the interaction rather than the details of its implementation. Service Bus allows services to be available by simple URLs, which are untied from their deployment location. It is possible to support publish-subscribe models, full-duplex communications point to point as well as in a peer-to-peer environment, unicast and multicast message delivery in one-way commu- nications, and asynchronous messaging to decouple application components.

In order to leverage these features, applications need to be connected to the bus, which provides these services. A connection is the Service Bus element that is priced by Azure on a pay-as-you-go basis. Users are billed on a connections-per-month basis, and they can buy advance "connection packs," which have a discounted price, if they can estimate their needs in advance.

Azure cache

Windows Azure provides a set of durable storage solutions that allow applications to persist their data. These solutions are based on disk storage, which might constitute a bottleneck for the applica- tions that need to gracefully scale along the clients' requests and dataset size dimensions.

Azure Cache is a service that allows developers to quickly access data persisted on Windows Azure storage or in SQL Azure. The service implements a distributed in-memory cache of which the size can be dynamically adjusted by applications according to their needs. It is possible to store any .NET managed object as well as many common data formats (table rows, XML, and binary data) and control its access by applications. Azure Cache is delivered as a service, and it can be

easily integrated with applications. This is a particularly true for ASP.NET applications, which already integrate providers for session state and page output caching based on Azure Cache.

The service is priced according the size of cache allocated by applications per month, despite their effective use of the cache. Currently, several cache sizes are available, ranging

from 128 MB ($45/month) to 4 GB ($325/month).

4.3.1.4 Other SERVICES

Compute, storage, and middleware services constitute the core components of the Windows Azure platform. Besides these, other services and components simplify the development and integration of applications with the Azure Cloud. An important area for these services is applications connectiv- ity, including virtual networking and content delivery.

Windows Azure virtual network

Networking services for applications are offered under the name *Windows Azure Virtual Network*, which includes *Windows Azure Connect* and *Windows Azure Traffic Manager*.

Windows Azure Connect allows easy setup of IP-based network connectivity among machines hosted on the private premises and the roles deployed on the Azure Cloud. This service is particu- larly useful in the case of VM roles, where machines hosted in the Azure Cloud become part of the private network of the enterprise and can be managed with the same tools used in the private premises.

Windows Azure Traffic Manager provides load-balancing features for services listening to the HTTP or HTTPS ports and hosted on multiple roles. It allows developers to choose from three dif- ferent load-balancing strategies: Performance, Round-Robin, and Failover.

Currently, the two services are still in beta phase and are available for free only by invitation.

Windows Azure content delivery network

Windows Azure Content Delivery Network (CDN) is the content delivery network solution that improves the content delivery capabilities of Windows Azure Storage and several other Microsoft services, such as *Microsoft Windows Update* and *Bing* maps. The service allows serving of Web objects (images, static HTML, CSS, and scripts) as well as streaming content by using a network of 24 locations distributed across the world.

4.3.2 SQL Azure

SQL Azure is a relational database service hosted on Windows Azure and built on the SQL Server technologies. The service extends the capabilities of SQL Server to the cloud and provides develo- pers with a scalable, highly available, and fault-tolerant relational database. SQL Azure is accessi- ble from either the Windows Azure Cloud or any other location that has access to the Azure Cloud. It is fully compatible with the interface exposed by SQL Server, so applications built for SQL Server can transparently migrate to SQL Azure. Moreover, the service is fully manageable using REST APIs, allowing developers to control databases deployed in the Azure Cloud as well as the firewall rules set up for their accessibility.

Figure 5.4 shows the architecture of SQL Azure. Access to SQL Azure is based on the Tabular Data Stream (TDS) protocol, which is the communication protocol underlying all the different

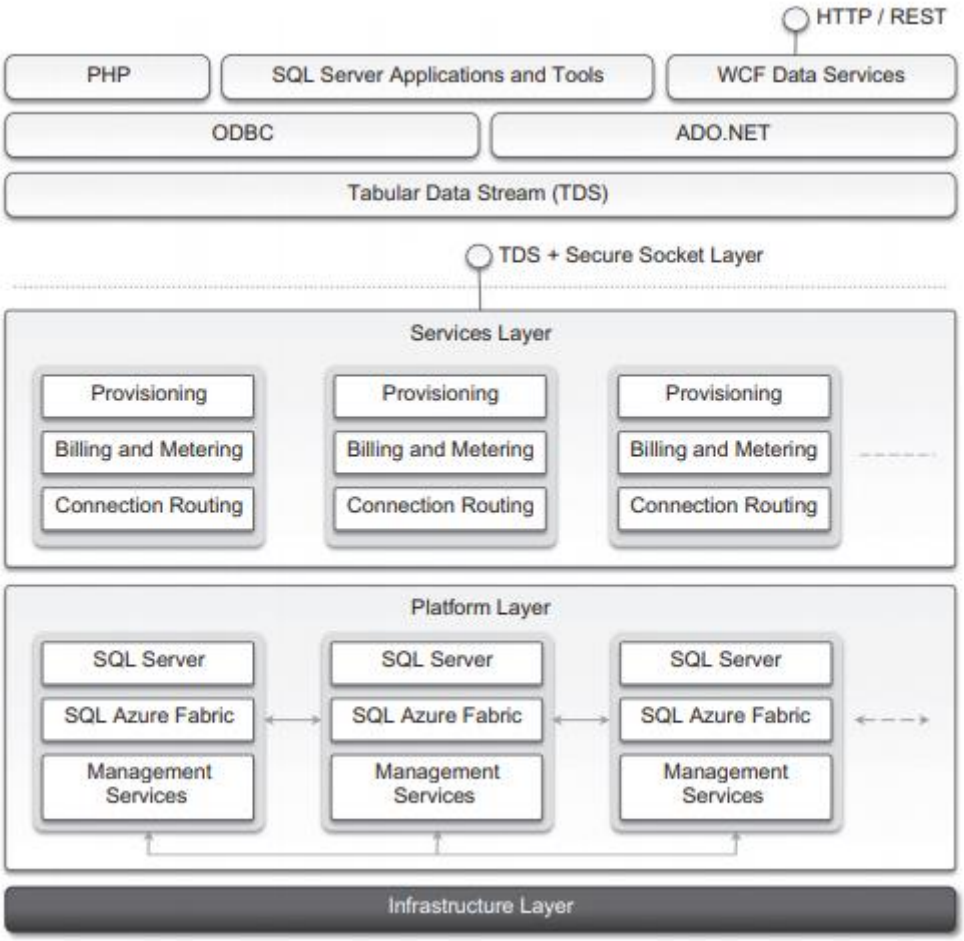


FIGURE 5.4 SQL Azure architecture.

interfaces used by applications to connect to a SQL Server-based installation such as ODBC and ADO.NET. On the SQL Azure side, access to data is mediated by the service layer, which provides provisioning, billing, and connection-routing services. These services are logically part of server instances, which are managed by SQL Azure Fabric. This is the distributed database middleware that constitutes the infrastructure of SQL Azure and that is deployed on Microsoft datacenters.

Developers have to sign up for a Windows Azure account in order to use SQL Azure. Once the account is activated, they can either use the Windows Azure Management Portal or the REST APIs to create servers and logins and to configure access to servers. SQL Azure servers are abstractions that closely resemble physical SQL Servers: They have a fully qualified domain name under the *database.windows.net* (i.e., *server-name.database.windows.net*) domain name. This simplifies the management tasks and the interaction with SQL Azure from client applications. SQL Azure ensures that multiple copies of each server are maintained within the Azure Cloud and that these copies are kept synchronized when client applications insert, update, and delete data on them.

Currently, the SQL Azure service is billed according to space usage and the type of edition. Currently, two different editions are available: Web Edition and Business Edition. The former is suited for small Web applications and supports databases with a maximum size of 1 GB or 5 GB. The latter is suited for independent software vendors, line-of-business applications, and enterprise applications and supports databases with a maximum size from 10 GB to 50 GB, in increments of 10 GB. Moreover, a bandwidth fee applies for any data transfer trespassing the Windows Azure Cloud or the region where the database is located. A monthly fee per

user/database is also charged and is based on the peak size the database reaches during the month.

4.3.3 Windows Azure platform appliance

The Windows Azure platform can also be deployed as an appliance on third-party data centers and constitutes the cloud infrastructure governing the physical servers of the datacenter. The Windows Azure Platform Appliance includes Windows Azure, SQL Azure, and Microsoft- specified configuration of network, storage, and server hardware. The appliance is a solution that targets governments and service providers who want to have their own cloud computing infrastructure.

As introduced earlier, Azure already provides a development environment that allows building applications for Azure in their own premises. The local development environment is not intended to be production middleware, but it is designed for developing and testing the functionalities of applications that will eventually be deployed on Azure. The Azure appliance is instead a full-featured implementation of Windows Azure. Its goal is to replicate Azure on a third-party infrastructure and make available its services beyond the boundaries of the Microsoft Cloud. The appliance addresses two major scenarios: institutions that have very large computing needs (such as government agencies) and institutions that cannot afford to transfer their data outside their premises.

4.3.4 Observations

Windows Azure is Microsoft's solution for developing cloud computing applications. Azure is an implementation of the PaaS layer and provides the developer with a collection of services and scal- able middleware hosted on Microsoft datacenters that address compute, storage, networking, and identity management needs of applications. The services Azure offers can be used either individu- ally or all together for building both applications that integrate cloud features and elastic computing systems completely hosted in the cloud.

The core components of the platform are composed of compute services, storage services, and middleware. Compute services are based on the abstraction of roles, which identify a sandboxed environment where developers can build their distributed and scalable components. These roles are useful for Web applications, back-end processing, and virtual computing. Storage services include solutions for static and dynamic content, which is organized in the form of tables with fewer con- straints than those imposed by the relational model. These and other services are implemented and made available through AppFabric, which constitutes the distributed and scalable middleware of Azure.

SQL Azure is another important element of Windows Azure and provides support for relational data in the cloud. SQL Azure is an extension of the capabilities of SQL Server adapted for the cloud environment and designed for dynamic scaling.

The platform is mostly based on the .NET technology and Windows systems, even though other technologies and systems can be supported. For this reason, Azure constitutes the solution of choice for migrating to the cloud applications that are already based on the .NET technology.

Summary

This chapter introduced some cloud platforms that are widely used in industry for building real commercial applications: Amazon Web Services, Google AppEngine, and Microsoft Windows Azure.

Amazon Web Services (AWS) provides solutions for building infrastructure in the Amazon Cloud. Amazon EC2 and Amazon S3 represent AWS's core value offering. The former allows developers to create virtual servers and customize their computing stack as required. The latter is a storage solution that allows users to store documents of any size. These core services are then complemented by a wide collection of services, covering networking, data management, con- tent distribution, computing middleware, and communication, which make AWS a complete solu- tion for developing entire cloud computing systems on top of the Amazon infrastructure.

Google AppEngine is a distributed and scalable platform for building Web applications in the Cloud. AppEngine is a scalable runtime that offers developers a collection of services for simplifying the development of Web applications. These services are designed with scalability in mind and constitute functional blocks that can be reused to define applications. Developers can build their applications in either Java or Python, first locally using the AppEngine SDK. Once the applications have been completed and fully tested, they can deploy the application on AppEngine.

Windows Azure is the cloud operating system deployed on Microsoft datacenters for building dynamically scalable applications. Azure's core components are represented by compute services expressed in terms of roles, storage services, and the AppFabric, the middleware that ties together all these services and constitutes the infrastructure of Azure. A role is a sandboxed runtime envi- ronment specialized for a specific development scenario: Web applications, background processing, and virtual computing. Developers define their Azure applications in terms of roles and then deploy these roles on Azure. Storage services represent a natural complement to roles. Besides storage for static data and semistructured data, Windows Azure also provides storage for relational data by means of the SQL Azure service.

AppEngine and Windows Azure are PaaS solutions. AWS extends its services across all three layers of the Cloud Computing Reference Model, although it is well known for its IaaS offerings, represented by EC2 and S3.

Question Bank

Part - A

1. What is AWS? What types of services does it provide?
2. Describe Amazon EC2 and its basic features.
3. What is a bucket? What type of storage does it provide?
4. What are the differences between Amazon SimpleDB and Amazon RDS?
5. What type of problems does the Amazon Virtual Private Cloud address?
6. Introduce and present the services provided by AWS to support connectivity among applications.
7. What is the Amazon CloudWatch?
8. What type of service is AppEngine?

.

9. What are the development technologies currently supported by AppEngine?
10. What is DataStore? What type of data can be stored in it?
11. Discuss the compute services offered by AppEngine.
12. What is Windows Azure?
13. Describe the architecture of Windows Azure.
14. What is a role? What types of roles can be used?
15. What is AppFabric, and which services does it provide?
16. What is SQL Azure?
17. What is the Windows Azure Platform Appliance? For which kinds of scenarios was this appliance designed?

Part B

1. Describe the Architecture and core components of Google AppEngine.
2. Explain in detail Amazon web services with Compute services, Storage services, Communication services and Additional services.
3. Describe the Microsoft azure and its core concepts.
4. Illustrate the architecture of SQL Azure.
5. Discuss the storage services provided by Windows Azure.