UNIT-I

(1) Introduction to cloud computing:

→ The term cloud refers to a Server that are accessed over the Internet which is present at remote location.

→ cloud can provide services over public and private networks i.e, WAN, LAN (or) VPN

→ Applications such as e-mail, web conferencing, customer relationship management (CRM) execute on cloud.

→ cloud computing means storing, Managing and accessing the data and programs on remote Servers that are hosted on the Internet instead of computer's hard drive (or) local server.

→ cloud computing is also referred to as Internet-based computing.

→ cloud computing involves provisioning of computing, Networking and storage resources on demand and providing these resources as metered services to the users; in a "Pay as you go" Model.

company - 10 systems we need to see we maintain Hardware

- for that we need to see where we don't need to be maintain Infrastructure

If we take 100- System and 50 employess will left Resource computers will get waste; All the same time we have, Some services - will have service

According to National Institute of Standards and Technology defines cloud computing as:

Definition:-

cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing Resources (e.g., Networks, Servers, storage, Applications and Services) that can be rapidly provisioned and released with minimal management effect (or) service provided interaction

(or)

cloud computing is the on-demand availability of computer system resources (especially) data storage/cloud storage and computing power) without direct active management by the user.

⇒ Advantages of cloud computing:-

1. Lower cost computer for users
2. Lower IT Infrastructure cost
3. fewer Maintenance cost.
4. Lower software cost.
5. Instant software updates.
6. Increased computing Power
7. unlimited storage capacity
8. cloud computing offers on-demand self-Service
9. It offers load balancing that makes it more

10. One can Manipulate and configure the applications online at any time

=> Disadvantages of cloud computing:-

1. Require a constant internet connection
2. Stored data might not be Secure
3. No control on Resources
4. Depends on cloud providers
5. Interoperability

(2) characteristics of cloud computing

NIST identifies five essential characteristics of cloud computing:

(i) On Demand self Service:- cloud computing allows the users to use web services and resource on demand, without requiring interactions with the cloud service provider (CSP). one can logon to a website at anytime & use them

(ii) Broad Network Access:- The computing services are generally provided over standard networks and heterogeneous devices/platforms such as workstations, laptops, tablets and smartphones.

(iii) Resource pooling:- The computing and storage resources provided by cloud service providers are Pooled to serve Multiple users using Multi-tenancy.

Multi-tenant aspects of the cloud allow multiple users to be served by the same physical hardware

(iv) Rapid Elasticity :- cloud computing resources can be provided rapidly and elastically. cloud resources can be rapidly scaled up (or) down based on demand two types of scaling options exist:

a. Horizontal scaling (Scaling Out):- It involves launching and providing additional Server resources

b. Vertical Scaling (scaling up):- It involves changing the computing capacity assigned to the Server resources while keeping the number of Server resources constant.

(v) Measured Service :- cloud computing resources are provided to users on a "pay-per-use Model". The usage of the cloud resources can be Monitored, controlled, and reported, providing transparency for both the provider and consumer.

In addition to these five essential characteristics of cloud computing, other common characteristics that again highlight 'savings' in cost include:

(i) performance :- cloud computing Provides improved performance for applications since the resources available to the applications can be scaled up or down based on the dynamic application workloads.

(ii) Reduced costs:- cloud computing Provides cost

and storage resources as required can be provisio-
-ned dynamically, and upfront investment in purchase
of computing assests to cover worst case requirements
is avoid. This saves significant cost for organizations
and individuals

Eg:- e-commerce applications

(iii) Outsourced Management:- cloud computing allows the users (individuals, large organizations, small and medium enterprises and governments) to outsource the IT infrastructure requirements to external cloud providers

(iv) Reliability:- Applications deployed in cloud computing environments generally have a higher reliability since the underlying IT infrastructure in professionally managed by the cloud service. cloud service providers specify and guarantee the reliability and availability levels for their cloud resources in the form of service level agreements (SLAs).

(v) Multi-tenancy:- The multi-tenanted approach of the cloud allows multiple users to make use of the same shared resources

- Modern applications such as e-commerce, Business-to-Business, Banking and financial, Retail and social Networking applications that are deployed in cloud computing Environments are multi-tenated applications
- multi-tenancy can be of different forms:

*. Virtual multi-tenancy:- In virtual multi-tenancy, computing and storage resources are shared among

*. organic multi-tenancy:- In organic multi-tenancy every component in the system architecture is shared among multiple tenants, including hardware, OS, database servers, application servers, load balancers, etc

3. Cloud Models:

3.1 - Service Models

3.2 - Deployment Models

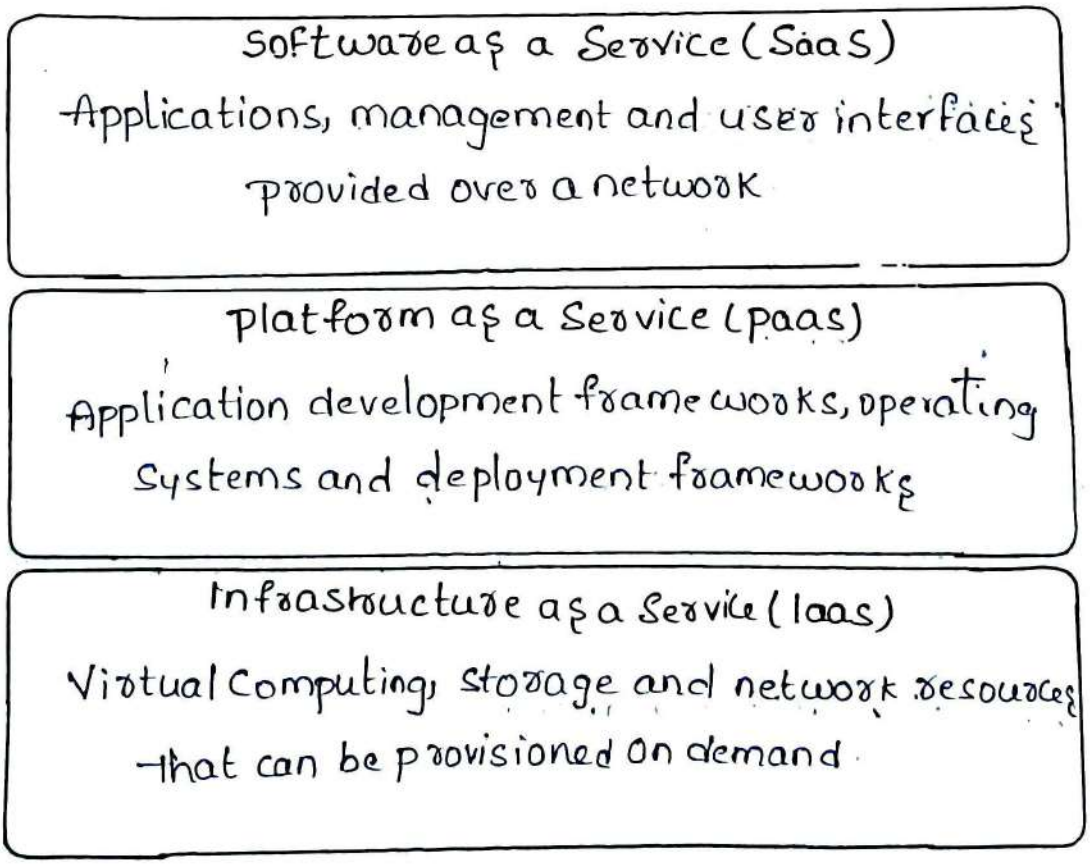
3.1 Service Models:

Cloud computing services are offered to users in different forms. NIST defines three cloud service models as follow

a. Infrastructure as a Service (Iaas)

b. Platform as a Service (Paas)

c. software as a service (Saas)



a. Infrastructure as a Service (Iaas):-

→ It provides us infrastructure

→ Iaas is also known as Hardware as a Service (Haas)

→ It is a type of cloud computing service used by System administrators/Network architects.

→ It simply Provides the underlying os (operating System) security, Networking, and Servers for developing the applications

→ It provides access to fundamental resource such as Machines, Virtual Machines, Virtual-storage etc.

→ on demand, over the internet and on a pay as you go basis

→ we can scale up and shrink the resources as per requirement

b. Platform-as-a Service (Paas):-

→ paas cloud computing Platform is created for the programmer to develop, test, run, and manage the applications

→ The cloud service provider manages the underlying cloud infrastructure including Servers, Network, Operating systems and storage

→ The users, themselves, are responsible for developing, deploying, configuring and managing applications on the cloud infrastructure.

c. Software-as-a-Service (saas):-

→ saas is also known as "on-demand software"

→ It is a software in which applications are

→ users can access the applications with the help of internet connection and web browser.

→ maintainance of software and hardware done by the vendors, so it removes the cost of Hardware and software Maintainance.

Iaas

Benefits
- shift focus from IT manag-ement to core activities
- pay-per-use/ pay-per-go-peic-ing
- Dynamic scaling
- Secure acces

characteristics
- multi tenancy
- virtualized hardware
- management tools
- Disaster recovery

Adoption
- Individual users: Low
- Small & medium enterprises: medium
- Large organizations: High
- Government: High

Example
- Amazon Elastic compute cloud (EC2)
- RackSpace
- GoGrid
- Eucalyptus
- Joyent
- Terremark
- Opsource

Paas

Benefits
- Lower upfront & operations costs
- Improved scalability
- Higher perform-ance
- Secure access

characteristics
- multi-tenancy
- open integration protocols
- App development tools & SDKs
- Analytics

Adoption
- Individual users: Low
- Small & medium enterprises: medium
- Large organizations: High
- Government: Medium

Example.
- Google App Engine
- windows Azure Platform
- force.com
- Rightscale
- Heroku
- Github
- Gigaspaces

Saas

Benefits
- Lower costs
- No infrastruc-ture required
- Seamless upgrades
- Secure
- High adoption
- On-the more access

characteristics
- multi-tenancy
- on-demand software
- open integration protocols
- Social network integration

Adoption
- Individual users: High
- Small & medium enterprises: High
- Large organizations: High
- Government: Medium

Example.
- Google Apps
- sales force.com
- facebook
- Zoho
- Dropbox
- Taleo
- microsoft office 365

Benefit characteristics and

3.2 Deployment Models:- NIST also defines four cloud deployment models as follows.

Public cloud	community cloud
Available for public use or a large industry group.	Available for shared use of several organizations supporting a specific community
Private cloud	**Hybrid cloud**
Operated for exclusive use of a single organization	combines multiple clouds (public and private) that remain unique but bound together to offer Applica-tion

fig:- cloud deployment models

a. public cloud:-

* In the public cloud deployment model, cloud services are available to the general public (or) a large group of companies
* managed by third-party cloud service provider
* public clouds are best suited for users who want to use cloud infrastructure for development and testing of applications and host applications in the cloud to serve large workloads, without upfront investments in IT infrastructure.

Advantages:-

- Minimal Investment
- Infrastructure Management is not required.
- No Maintenance.
- Dynamic scalability
- No setup cost

Disadvantages:-

- Less secure because resources are shared

fig:- a. public cloud deployment model

b. private cloud:-

* Private cloud services are dedicated for a single organization.
* It is managed by Third party (or) the organi-zation internally.
* Private clouds are best suited for applications where security is very important and organiza-tions that wont to have very tight control over their data

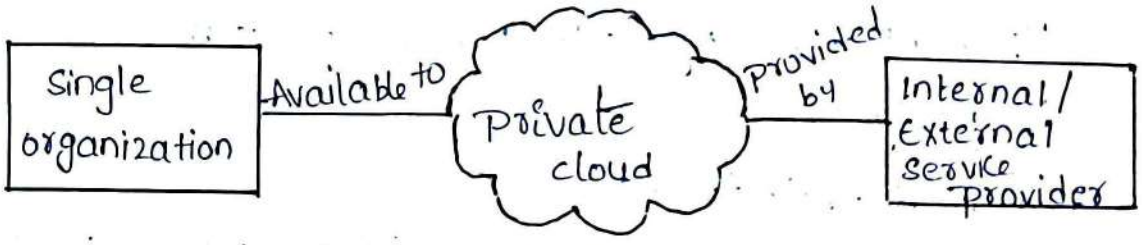


fig:- b. private cloud Deployment model

Advantages:-

- High security
- Data privacy → only authorized people can access the data
- More customizable → as companies get to customize their solution as per requirements

Disadvantages:-

- private cloud is accessible within an organization so the area of operations is limited.
- High cost → As we need to invest in Hardware and Software
- Limited Scalability

c. Hybrid cloud:-

* The hybrid cloud deployment model combines the services of multiple clouds (private (or) public)
* critical activities are performed by Private cloud & Non critical activities by public cloud.
* Hybrid clouds are best suited for organizations that want to take advantage of secured application and data hosting on a private cloud, and at the same time benefit from cost savings by hosting shared applications and data in public clouds

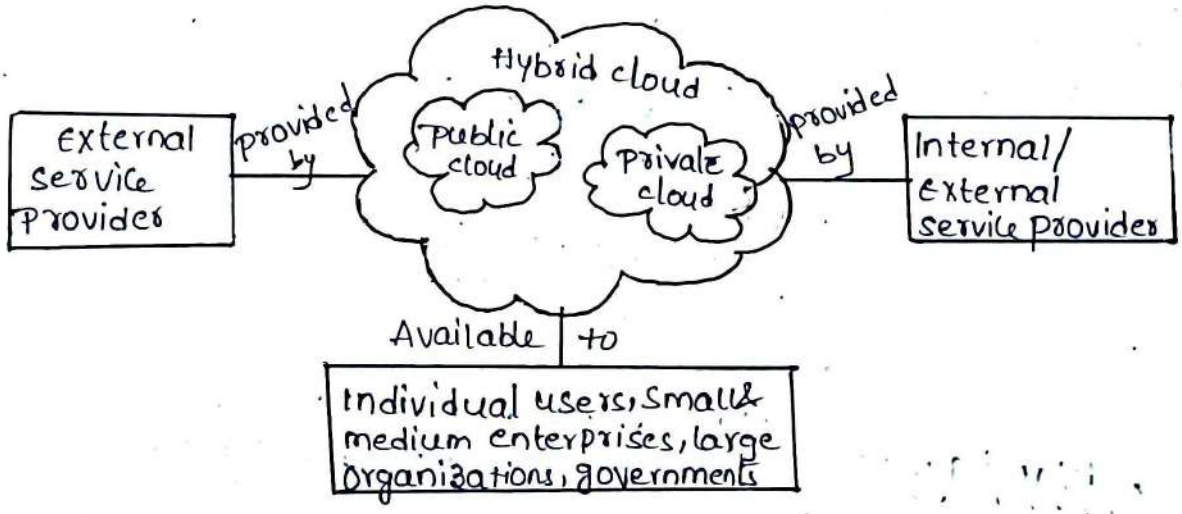


fig:-c. Hybrid cloud Deployment Model

Advantages:-

- Scalability
- security
- Low cost (as compared to Private cloud)

Disadvantages:-

- Managing is complex because there are more than 1 type of deployment model.
- Dependency on infrastructure.

d. community cloud:-

* In the community cloud deployment model, the cloud services are shared by several organizations that have the same policy and compliance considerations.
* Available for shared use of several organizations to share the information between the organization and a specific community.

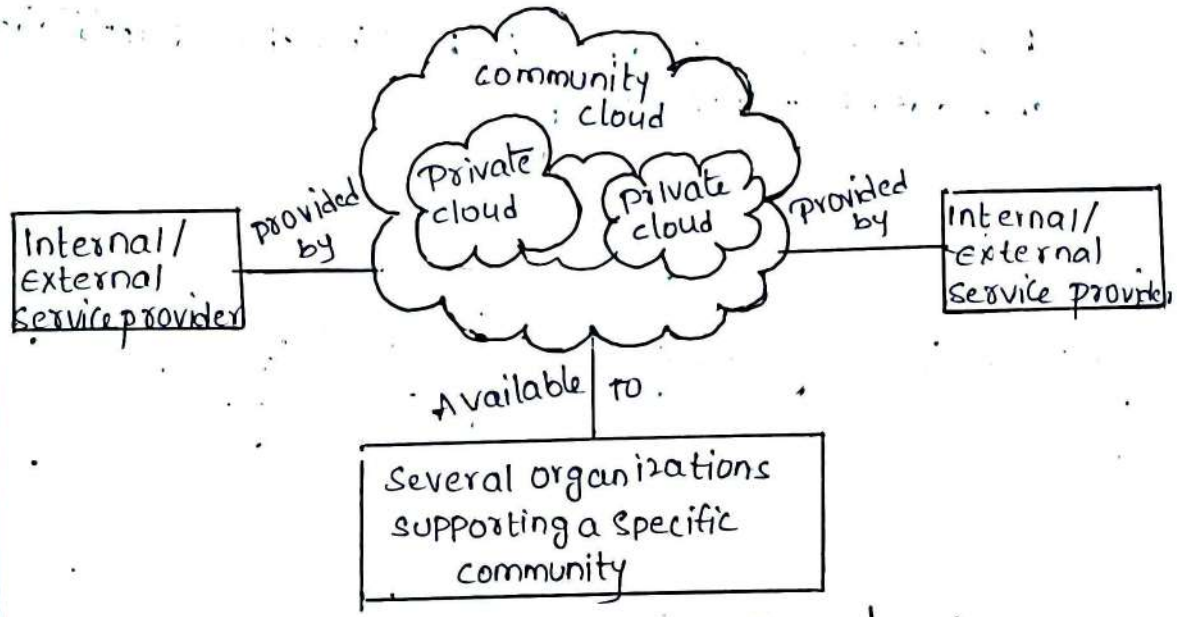


fig:- d. community cloud deployment model

Advantages:-

- cost effective
- It provides Better Security
- sharing Resources Among companies
- more secured than public cloud but less than Private cloud.

Disadvantages:-

- Data is accessable between organization
- consistent Maintenance cost
- overall cost will be increased compared to public cloud.

4. cloud services Examples

4.1 Iaas: Amazon Ec2, Google compute Engine, Azure Vms

4.2 paas: Google App Engine

4.3 saas: Salesforce.

4.1 Iaas: Amazon Ec2, Google compute Engine, Azure Vms

(1) AEC2 is an Infrastructure-as-a-Service (Iaas) offered from Amazon.com

(2) Elastic compute cloud (EC2) is a Webservice that provides computing capacity in the form of Virtual Machines that are Launched in Amazon's cloud computing Environment

(3) It is used to Launch as many (or) a few Virtual servers as you need, configure Security and Networking, and manage storage.

(4) It is Enables you to scale up(or) down to handle changes in requirements, reducing your need to forecast traffic.

(5) Amazon Provides pre-configured Amazon Machine Images (AMIs) Which are templates of cloud instances

(6) The pricing model for Ec2 instance is based on a pay-per use model

(7) AEC2 provides number of powerful features for building scalable and reliable applications such as auto-scaling and elastic load balancing.

(8) users can also create their own AMIs with custom Applications, libraries and data.

(9) Google compute Engine (GCE) is a Iaas offered from Google.

(10) It provides Virtual Machines of various computing capacities ranging from small instances (Eg: 1 virtual core with 1.38 GCE unit and 1.7GB memory) to high Memory Machine types (Eg: 8 virtual core with 22 GCE units and 52 GB Memory).

(11) Windows Azure Virtual machines is an Iaas offered from microsoft.

(12) Azure Vms Provides virtual Machines of various computing capacities ranging from small instances (1 Virtual core 1Vc) with 1.75 GB memory) to memory intensive machine types (8 virtual cores with 56GB Memory)

4.2 Paas: Google App Engine (GAE)

(1) GAE is a platform-as-a service offered from Google.

(2) It is a cloud-based service for hosting web applications and storing data.

(3) It allows users to build scalable and reliable applications that run on the same systems that power Googles own Applications

(5) Developers can develop and test their applications with GAE SDK on a local Machine and then upload it Google App Engine (GAE) with a simple click of a button.

(6) Applications hosted in GAE are easy to built, Maintain and Scale.

(7) GAE Supports applications written in Several programming languages.

(8) With GAE's Java runtime environment developers can build applications written in several programming languages.

(9) GAE also provides runtime Environment for python Languages

(10) The pricing model for GAE is based on the amount of computing resources used.

(11) GAE provides free computing resources for applications upto a certain limit. Beyond that limit, users are billed based on amount of resources used.

4.3. SaaS: Salesforce

(1) Salesforce is a cloud-based customer relationship management Software offered from software-as-a-service

(2) Users can access CRM application from anywhere through internet-enabled devices such as workstations, laptops, tablets and Smartphones.

(3) Salesforce is a popular CRM tool for support, sales and marketing teams worldwide.

(4) Salesforce Services allow businesses to use

(5) For Example: LinkedIn - It brings companies and customers together on the CRM

(6) salesforce allows customer representatives to manage customer profiles, track opportunities, optimize campaigns from lead to close and monitor the impact of campaigns.

(7) Salesforce Service cloud is a cloud-based customer service management Saas.

(8) Service cloud provides companies a call-center like view and allows creating, tracking, routing and escalating cases, also provides self service capabilities to customers.

(9) Salesforce marketing cloud is a cloud based social marketing Saas.

(10) Marketing cloud allows companies to identify sales from social media, identify the most trending information on any topic and allows companies to actively engage with customers, to create and deploy social content, Manage and execute optimized social advertisement campaigns and track the performance of social campaigns.

(11) Some of the tools included in the sales force sales, Service and Marketing clouds include:

- Accounts and contacts
- Leads
- chatter
- Analytics and

Cloud Based Services and applications

5.1 Cloud Computing for Healthcare:

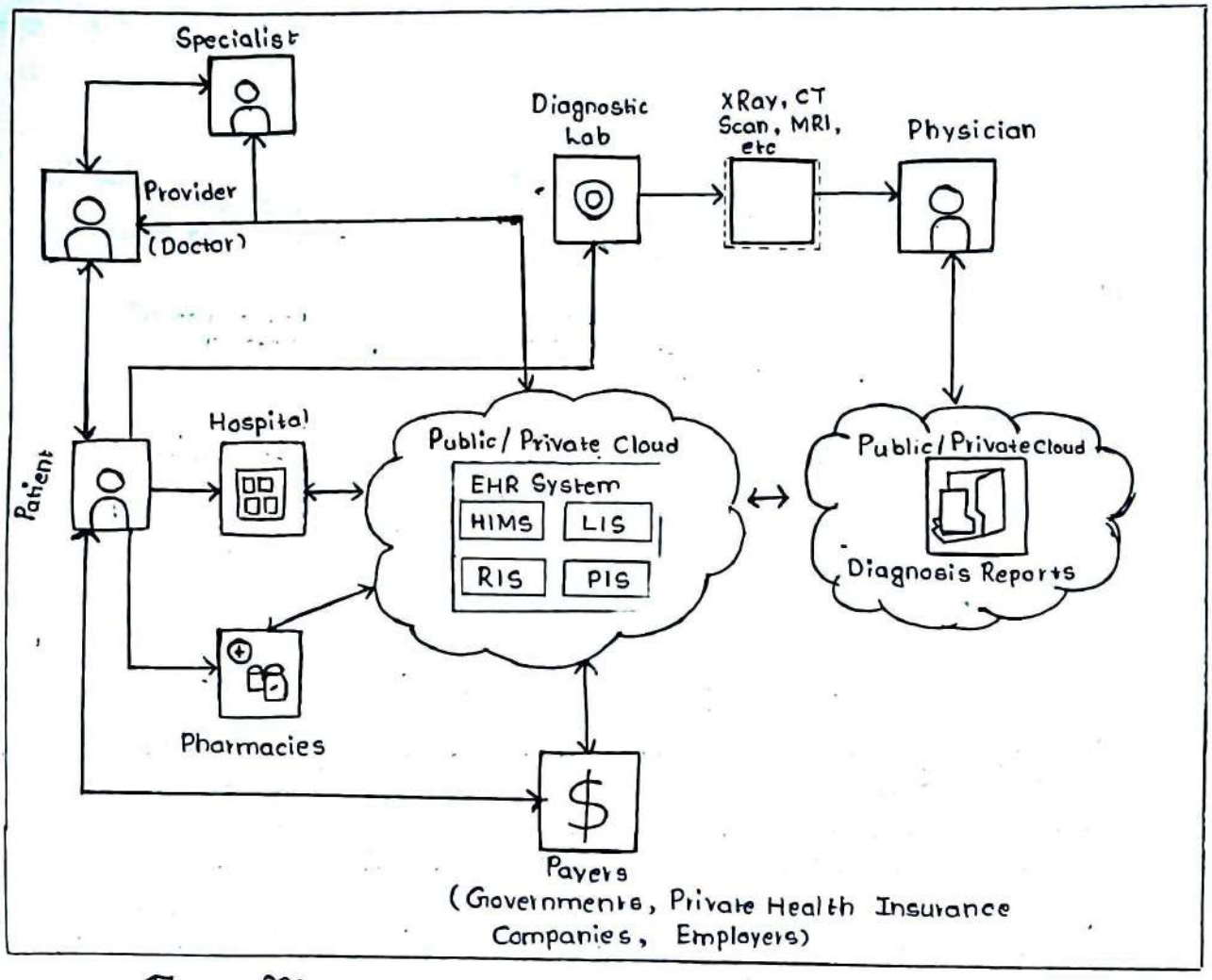


Fig: Cloud computing for Healthcare

1. The above figure shows the application of cloud computing environments to the healthcare system.
2. Hospitals and their affiliated providers can securely access patient data stored in the cloud and share the data with other hospitals and physicians.
3. Patients can access their own health information from all of their care providers and store it in a personal health record (PHR).
4. The PHR can be a vehicle for e-prescribing, a technique known to reduce paper prescriptions and allows physicians and other medical practitioners to write & send prescriptions to a participating pharmacy electronically.
5. History and information staged stored in the cloud using SaaS applications can streamline the admissions, care & discharge processes by eliminating redundant data collection and entry.
6. Health payers can increase the effectiveness & lower the

5.2 Cloud computing for energy systems

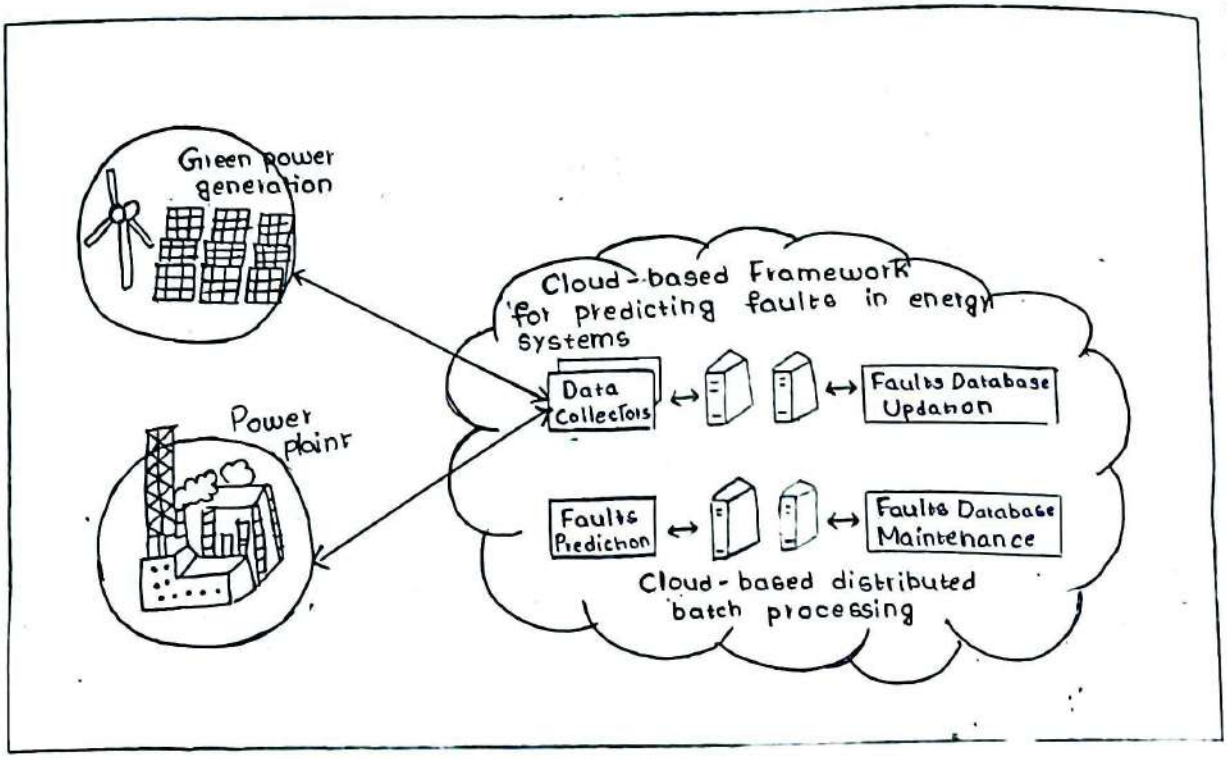


Fig.: Cloud computing for energy systems

1. Energy systems such as smart grids, power plants, wind turbine farms, etc have thousands of sensors that gather real-time maintenance data continuously for condition monitoring & failure prediction purposes.
2. These energy systems have a large no. of critical components that must function correctly so that the systems can perform their operations correctly.
3. For example, a wind turbine has a no. of critical components, e.g., bearings, turning gears, etc that must be monitored carefully as sudden change in operating conditions of the machines can result in failures.
4. In systems such as power grids, real-time information is collected using specialized electrical sensors called Phasor Measurement Units (PMU) at the substitutions.
5. The information received from PMUs must be monitored in real-time for estimating the state of the system & for predicting failures.
6. Maintenance & repair of such complex systems is not only expensive but also time consuming, therefore failures can cause huge losses for the operators & supply outage / interruption for customers.
7. Generic framework, such as cloud view can be used for maintenance data, storage, processing & analysis of machine maintenance data, collecting from a large no. of sensors embedded in individual industrial machines, in a cloud computing environment.

cloud based architectures that leverages the advantages of the parallel computing capabilities of the cloud to make local decisions with global information efficiently.

5.3 Cloud Computing for Transportation Systems

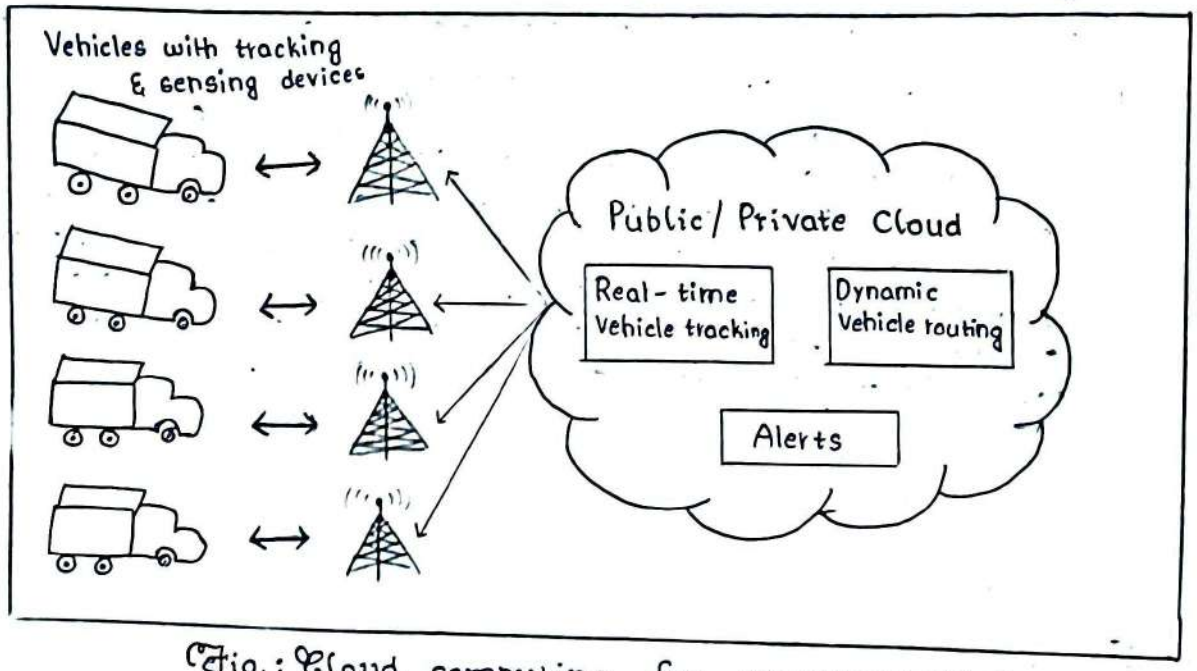


Fig.: Cloud computing for transportation systems

1. Intelligent Transportation Systems (ITS) have evolved significantly in recent years.
2. Modern ITS are driven by data collected from multiple sources which is processed to provide new services to their users.
3. By collecting large amount of data from various sources & processing the data into useful information.
4. Data-driven ITS can provide new services such as advanced route guidance, dynamic vehicle routing, anticipating customer demands for pickup & delivery problem etc.
5. Primarily, collection & organization of data from multiple sources in real-time & using the massive amounts of data for providing intelligent decisions for operations & supply chains, is a major challenge, because the size of the databases involved is very large & real-time analysis tools have not been available.
6. As a result large organizations are facing problems which cannot be overcomed apparently of analyzing terabytes of unorganized data stored on isolated & distinct geographical locations.
7. They proposed a cloud-based framework that can be used for tracking & monitoring fresh food supply which can be damaged during transit due to unrefrigerated conditions & changes in environmental

8. Spoilage of fruits & vegetables during transport and distribution not only results in losses to the distributors but also presents a hazard to food safety.

9. The analysis & interpretation of data on environmental conditions in the container & food track positioning can enable more effective routing decisions in real-time.

10. It is possible to take remedial measures such as,
   i) The food that has a limited time budget before it gets rotten can be re-routed to a closer destination.
   ii) Alerts can be raised to the driver & the distributor about the transit conditions, such as container temperature exceeding the allowed limit, humidity levels going out of the allowed limit, etc. & corrective actions can be taken before the food gets damaged.

5.4 Cloud computing for manufacturing industry

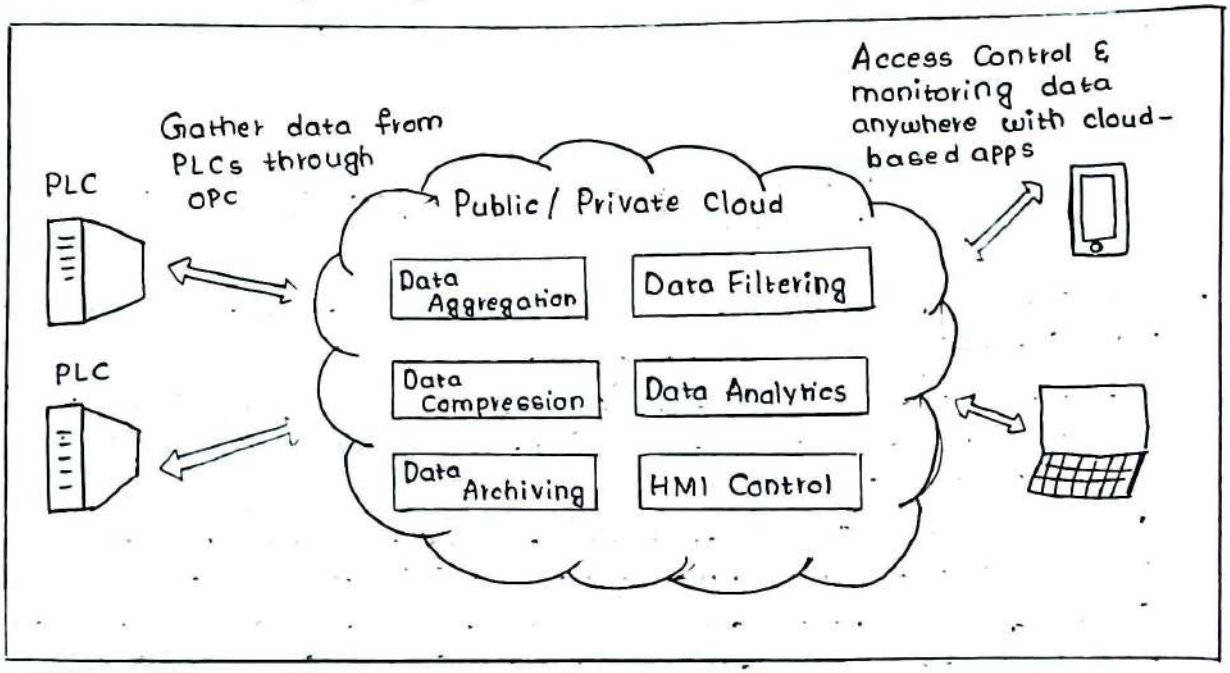


Fig.: Cloud Computing for manufacturing industry.

1. Industrial Control Systems (ICS), such as supervisory control & data acquisition (SCADA) systems, distributed control systems (DCS), & other control system configurations such as programmable logic controllers (PLC) continuously generate monitoring & control data.

2. Real-time collection, management & analysis of data on production operations generated by ICS, in cloud, can help in estimating the state of the systems, improve their personnel safety & take appropriate action in real-time to prevent sudden / unexpected machine failures.

5.5 Cloud computing for Government

1. Cloud computing plays a significant role for improving the efficiency & transparency of govt operations.
2. Cloud based e-Governance systems can improve the delivery of services to citizens, business, govt. employees & agencies etc
3. To improve the participation of all responsible parties in various govt schemes & policy formation processes.
4. Public services such as public transport reservations, vehicle registrations, issuing of driving licenses, income tax filing, electricity & water bill payments, birth (or) marriage registrations, etc can be facilitated through cloud-based applications.
5. The benefit of using cloud for public service applications is that the applications can be scaled upto serve a large no. of citizens.
6. Cloud-based applications can share common data related to citizens.
7. Utilization of govt schemes data can be collected from citizens & used in the policy formation process & improvement of schemes.

Following figure shows generic use case of cloud computing for government.

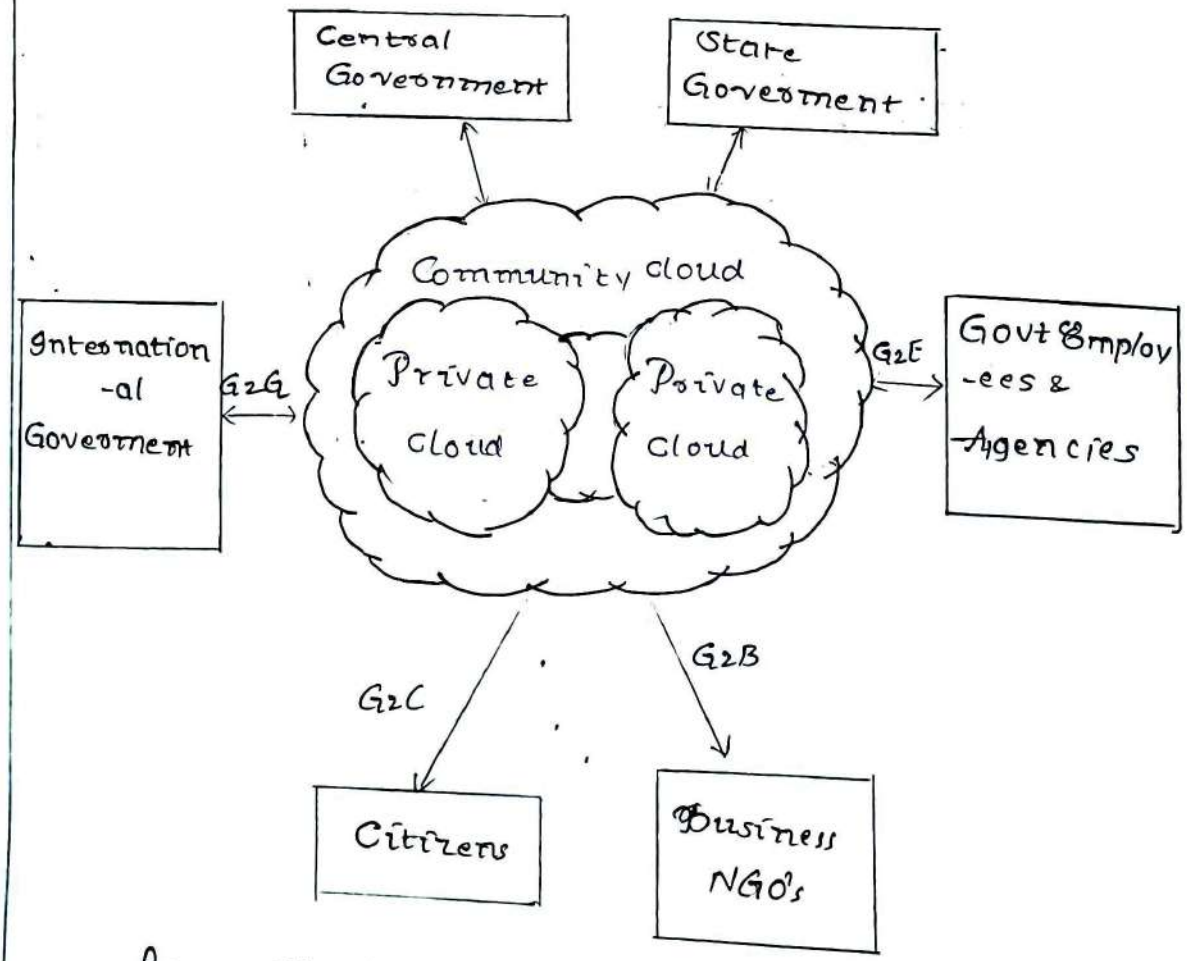
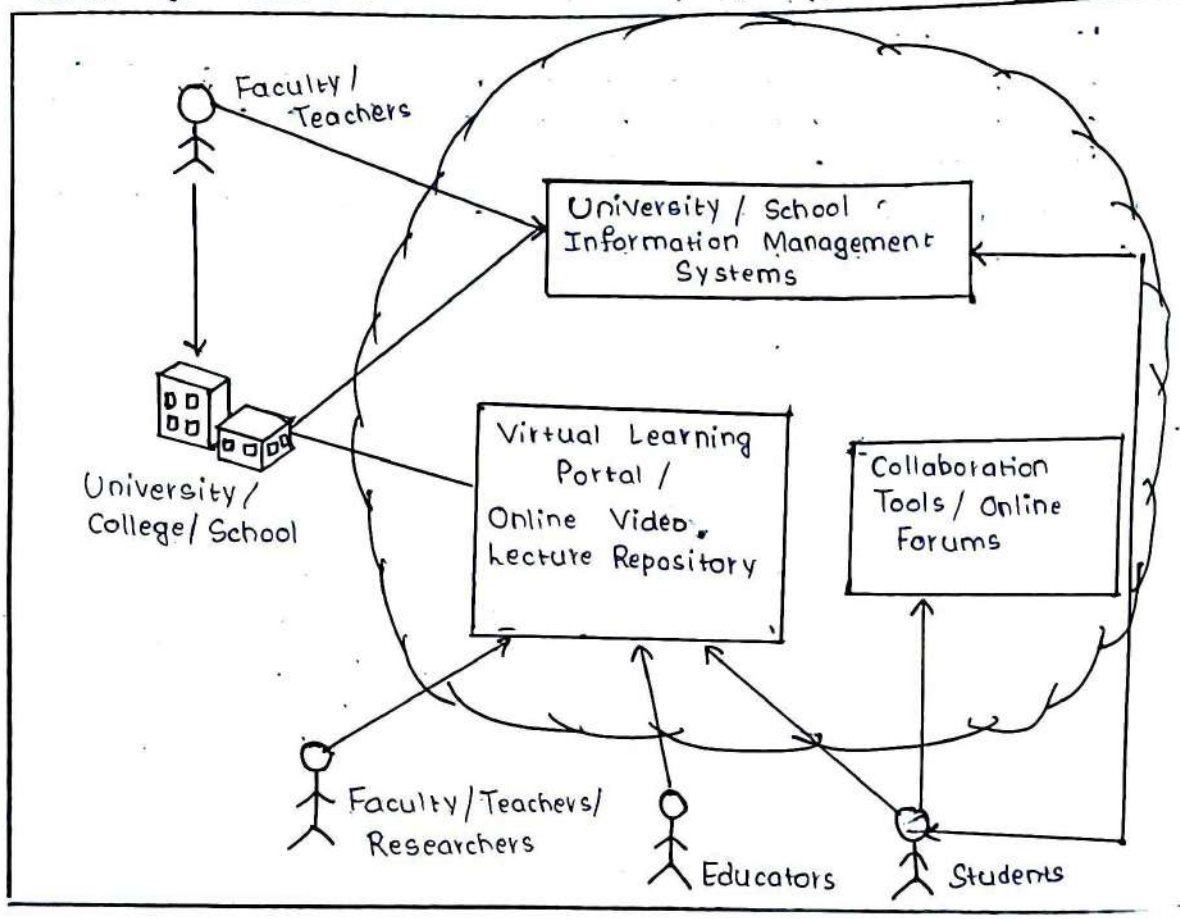


Figure: Cloud Computing for G

5.6 Cloud Computing for Education

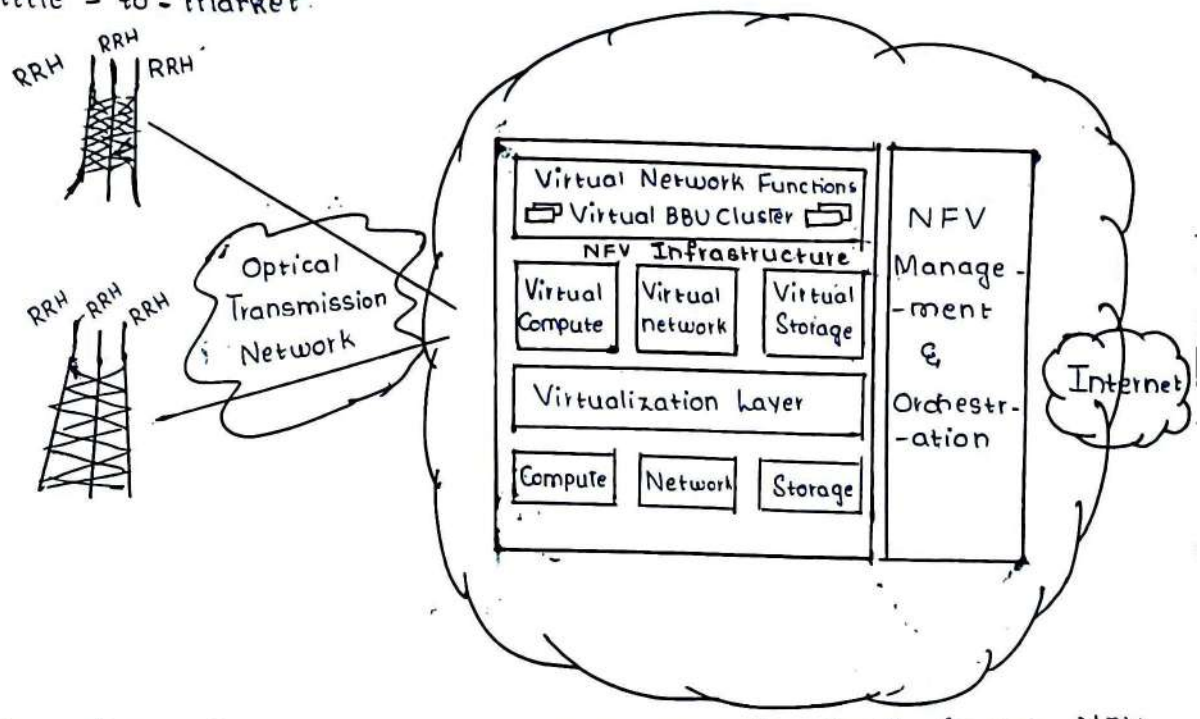
1. Cloud computing helps in improving the quality of the online education to students.
2. Cloud-based collaboration applications such as online forums, can help students discuss common problems & seek guidance from experts.
3. Universities, colleges & schools can use cloud-based the information management systems to admissions, improve administrative efficiency, offer online & distance education programs, online exams, track progress of students, collect feedback from students.
4. Cloud-based online learning systems can provide access to high quality educational matetial to students.
5. Cloud-based systems can help universities, colleges & schools in cutting down the IT infrastructure costs & provides access to educational services to a large no. of students. Below figure shows a generic use case of cloud for education.



5.7 Cloud Computing for Mobile Communication

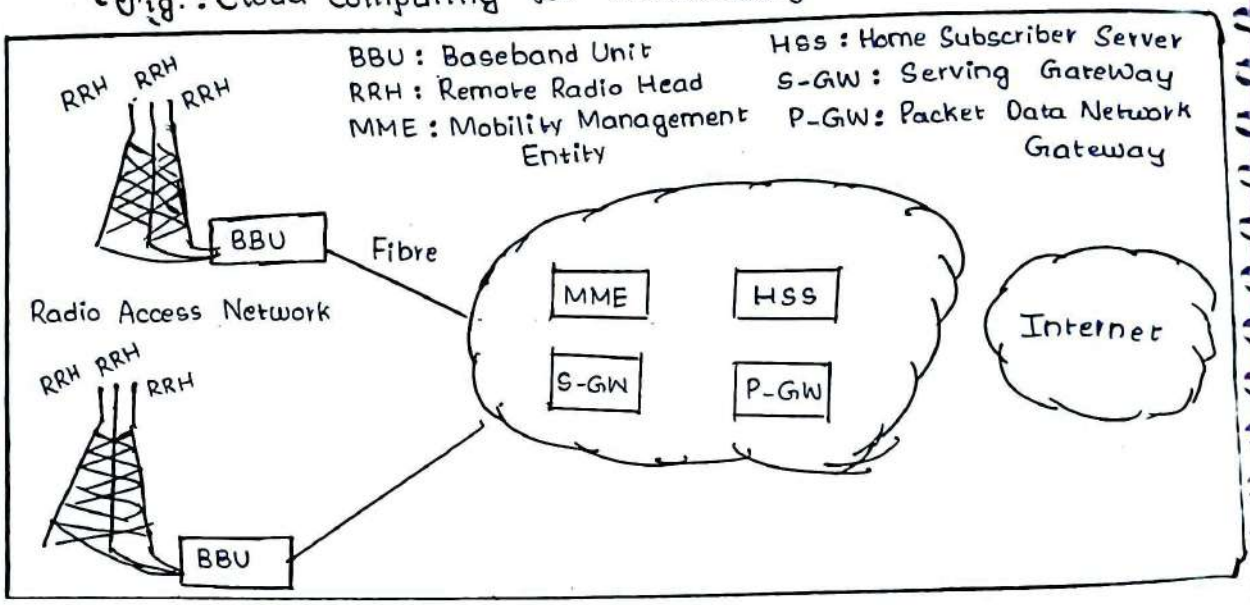
1. Mobile communication infrastructure involves multiple nlw devices for the radio access network (PAN) & the core network (CN).
2. Of variety of related hardware components & systems are used for these network devices including their cost & inflexibility.

4. Due to the increasing speed of innovation, the lifecycles of the n/w devices are becoming shorter.
5. Network Function Virtualization (NFV) is seen as a key enabling technology for the 5th generation of Mobile Communication N/w (5G) in the next decade.
6. NFV will leverage cloud computing to combine the multiple n/w devices into the cloud.
7. The NFV architecture being standardized by European Telecommunications Standards Institute (ETSI) made up of NFV infrastructure, Virtual N/w functions & NFV management & orchestration layers.
8. NFV is made up of n/w functions implemented in s/w that run on virtualized resources in the cloud.
9. NFV enables the separation of n/w functions which are implemented in s/w from those of the underlying h/w, thus, n/w functions can be easily tested & upgraded by installing new s/w while the h/w remains the same.
10. This flexibility will speed up innovation & reduce the time - to - market.



- The above figure shows a use case of cloud-based NFV architecture for cloud-based radio access n/w (C-RANs) with virtualized mobile base stations (Baseband units).
- The baseband units such as eNode B in 4G, in current mobile communication n/ws are co-located with the cell towers on-site & run on related hardware.
- With many NFV & cloud the BBUs can be virtualized only as many resources as required to meet the workload levels can be provided on-demand.
- This will result in significant power savings.

Fig.: Cloud Computing for virtualizing radio access network



- Above figure shows a usecase of cloud-based NFV architecture for mobile core nlw.
- The core nlw devices such as mobility management entity (MME), Home Subscriber Server (HSS), Serving Gateway (S-GW) & packet data nlw gateway in 4G can be implemented in slw & deployed on virtualized resources in the cloud.
- Benefits of using cloud-based NFV architecture for mobile core nlw include improved resource utilization efficiency, nlw resilience, flexibility in scaling up capacity.

6. Cloud Concepts & Technologies

6.1 Virtualization :

1. Virtualization is a technology in cloud computing.
2. Virtualization refers to the partitioning the resources of a physical system into multiple virtual resources (computing, storage, nlw & memory).
3. It is a key enabling technology of cloud computing & allows pooling of resources.
4. In cloud computing resources are pooled to serve multiple users using multi-tenancy.
5. Multi-tenant aspects of the cloud allow multiple users to be served by the same physical hardware.
6. Users are assigned virtual resources that run on top of the physical resources.
7. The following diagram shows the architecture of a virtualization technology in cloud computing.
8. The physical resources such as computing, storage, memory & nlw resources are virtualized.
9. The virtualization layer partitions the physical

10. Virtualization layer allows multiple OS instances to run currently as virtual machines on the same underlying physical resources.

* Hypervisor :

1. The virtualization layer consists of a hypervisor or a virtual Machine Monitor (VMM)
2. The hypervisor presents a virtual operating platform to a guest operating system (OS).
3. There are two types of hypervisors are shown in the following figures.

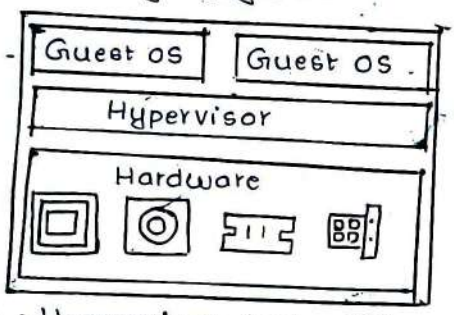


Fig. : Hypervisor design : Type-1

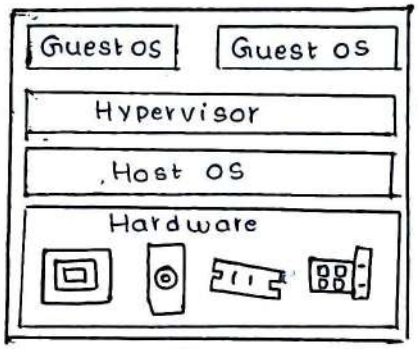


Fig. : Hypervisor design : Type - 2

4. Type-1 Hypervisor (or) the native hypervisors run directly on the host hardware & control the hardware & monitor the guest OS.
5. Type-2 Hypervisor (or) Host hypervisors run on top of a conventional (main/host) operating system & monitor the guest OS.

* Guest OS :

1. A guest OS in an OS that is installed in a virtual machine in addition to the host (or) main OS.
2. In virtualization, the guest OS can be different from the host OS.

Various terms of virtualization approaches exist :

Full Virtualization :

1. In full virtualization, the virtualization layer completely decouples the guest OS from the underlying hardware.
2. The guest OS requires no modification & is not aware that it is being virtualized.
3. Full virtualization is enabled by direct execution of user requests & binary translation of OS requests.

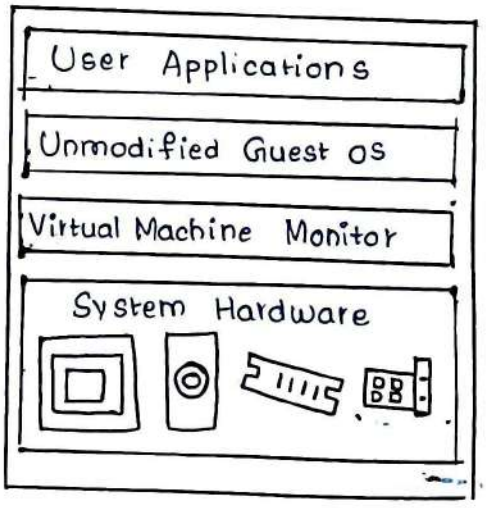


Fig. : Full Virtualization

Para-Virtualization :

1. In para-virtualization, the guest OS is modified to enable communication with the hypervisor to improve performance & efficiency.
2. The guest OS kernel is modified to replace non-virtualized instructions with hypercalls that will communicate directly with the virtualization layer hypervisor.

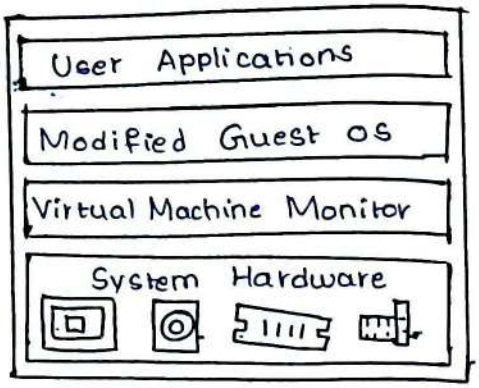


Fig.: Para Virtualization

Hardware Virtualization :

1. Hardware assisted virtualization is enabled by hardware features such as Intel's virtualization Technology VT-x and AMD's AMD-V.
2. In hardware assisted virtualization, sensitive calls are set to automatically trap to the hypervisor.
3. Thus, there is no need for either binary translation or para-virtualization.

Following table shows lists of some examples of popular hypervisors.

Hypervisor	Type
Citrix XenServer	Type-1
Oracle VM Server	Type-1
KVM	Type-1
VMWare ESX/ESXi	Type-1
Microsoft Hyper-V	Type-1
Xen Hypervisor	Type-1
VMWare Workstation	Type-2
Virtual Box	Type-2

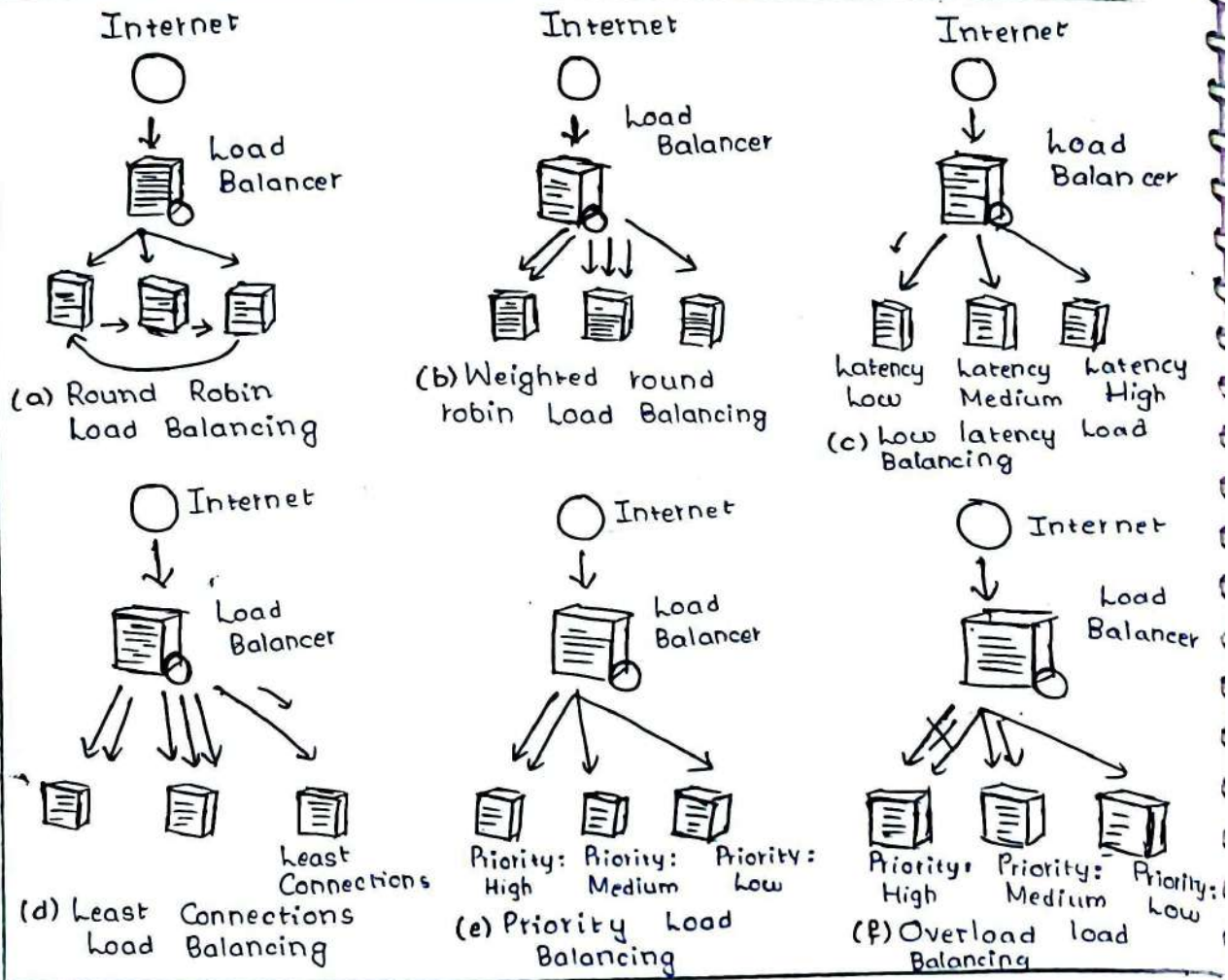
6.2 Load Balancing

1. One of the important features of cloud computing is Scalability.
2. Cloud computing resources can be scaled up on demand to meet the performance requirements of applications.
3. Load balancing distributes workloads across multiple servers to meet the application workload.
4. The goals of load balancing techniques are to achieve maximum utilization of resources, minimizing the

across multiple resources.

6. Cloud-based applications can achieve high availability & reliability with load-balancing.
7. Load balancers are used to serve the user requests, in the event of failure of one or more of the resources, load balancers can automatically reroute the user traffic to the healthy resources.
8. The routing of user requests is determined based on a load balancing algorithm Commonly used load balancing include:

- Round Robin: The servers are selected one by one to serve the incoming request in a non-hierarchal circular-fashion with no priority assigned to a specific server.
- Weighted Round Robin:
  1. In weighted round robin load balancing, servers are assigned some weights.
  2. The incoming requests are proportionally routed using a static or dynamic ratio of respective weights.
- Low Latency:
  1. In low latency load balancing the load balancer monitors the latency of each server.
  2. Each incoming request is routed to the server which has the lowest latency.
- Least connections: In least connections load balancing, the incoming requests are routed to the server with the least no. of connections.
- Priority:
  1. In priority load balancing, each server is assigned a priority.
  2. The incoming traffic is routed to the highest priority server as long as the server is available.
  3. When the highest priority server available fails, the incoming traffic is routed to a server with a lower priority.
- Overflow:
  1. Overflow load balancing is similar to priority load balancing.
  2. When the incoming requests to highest priority server overflow, the requests are routed to a lower priority server.
- The following figures depicts various load balancing approaches.



- For session based applications, an important issue to handle during load balancing is the persistence of multiple requests from a particular user session. Three commonly used persistence approaches are described below:

(a) Sticky sessions:

- All the request belonging to a user session are routed/carried out to the same server. These sessions are called sticky sessions.
- Benefit of this approach is that it makes session management simple.
- Drawback of this approach is that if a server fails all the sessions belonging to that server are lost.

(b) Session database:

- All the session information is stored externally in a separate session database, which is often replicated to avoid a single point of failure.

(c) Browser cookies:

- The session information is stored on the client side in the term of browser cookies.
- The benefit of this approach is that it makes the session management easy & has the least

- URL re-writing:
  - URL re-write engine stores the session information by modifying the URLs on the client side
  - This approach avoids overhead on the load balancer.
  - Drawback is that the output of session information can be stored is limited

Load Balancing can be implemented in s/w & h/w.

- Software based load balancers run on standard OS & like other cloud resources, load balancers are virtualized.
- Hardware based load balancers implement load balancing algorithms in Application Specific Integrated Circuits (ASICS).

Examples of Load Balancing:

Load Balancing	Type
Nginx	Software
HAProxy	Software
Pound	Software
Varish	Software
Cisco systems catalyst 6500	Hardware
Coyote point equalizer	Hardware
F5 Network BIG-IP LTM	Hardware
Barracuda load balancer	Hardware

6.3 Scalability & Elasticity:

1. Multi-tier applications such as e-commerce, social networking, business-to-business, etc can experience rapid changes in their traffic.
2. Each website has a different traffic platform which is determined by a no. of factors that are generally hard to predict before.
3. Modern web applications have multiple tiers of deploying with varying no. of servers in each tier. Capacity planning is an important task for such applications.
4. Traditional approaches for capacity planning are based on predicted outcomes demands for applications & account for worst case peak loads of applications.
5. When the workloads of applications increases, the traditional approaches can be either scaled up or scaled out.
6. Scaling up involves upgrading the h/w resources.

8. Scaling up & Scaling out approaches are based on demand forecasts at regular intervals of time.
9. Variations in workloads are rapid; traditional approaches are unable to keep track with the demand which leads to either over-provisioning or under provisioning of resources
10. Over-provisioning of resources leads to higher capital expenditures than required.
11. Under-provisioning of resources leads to traffic overloads slow response times, low throughputs & hence loss of opportunity to serve the customers.

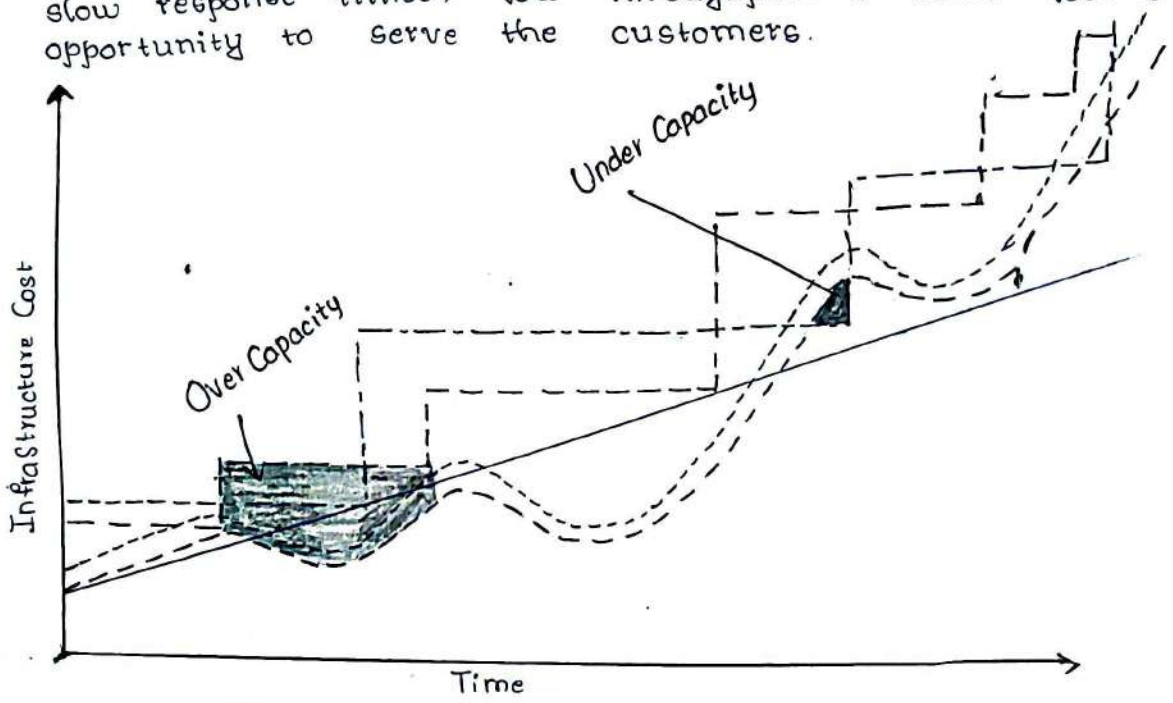


Fig.: Cost versus Capacity curves

6.4 Deployment-

- Deployment prototyping can help in making deployment architecture design choices.
- By comparing performance of alternative deployment architectures, deployment prototyping can help in choosing the best & most effective deployment architecture that can meet the application performance requirements.
- Deployment design is an alternative process that involves the following three steps:
    1. Deployment Design
    2. Performance Evaluation
    3. Deployment Refinement

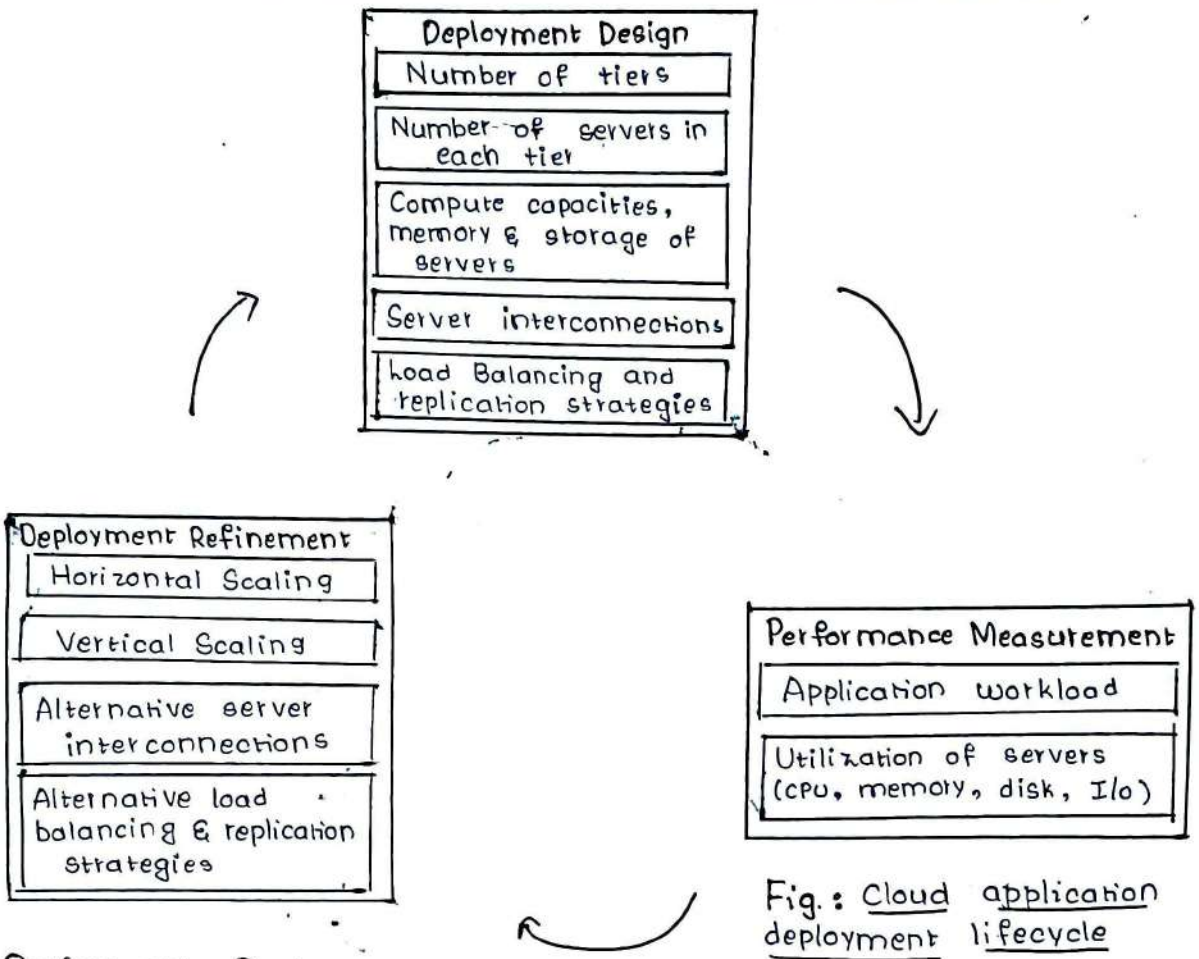


Fig.: Cloud application deployment lifecycle

1. Deployment Design

* Application deployment is created with various tiers as specified in the deployment configuration.
* The variables in this step include the no of servers in each tier, computing, memory & storage capacities of servers, server interconnection, load balancing and replication strategies.
* Deployment is created by provisioning the cloud resources as specified in the deployment configuration.
* The process of providing resources & deployment creation is automated & involves a no of steps such as launching of server instances, configuration of the servers & deployment of various tiers of application on the servers.

2. Performance Evaluation

* Once the application is deployed in the cloud, the next step in the deployment lifecycle is to verify whether the application meets the performance requirements with the deployment.
* This step involves monitoring the workload on the application & measuring various workload parameters such as response time & management.
* The utilization of servers CPU, memory, disk, I/o etc in each tier is also monitored.

3. Deployment Refinement :

* After evaluating the performance of the application, deployments are refined so that the application can meet the performance requirement.
* Various alternatives can exist in this step such as vertical scaling, horizontal scaling, alternative server interconnections, alternative load balancing & replication strategies, for instance.

Examples of popular cloud deployment management tools:

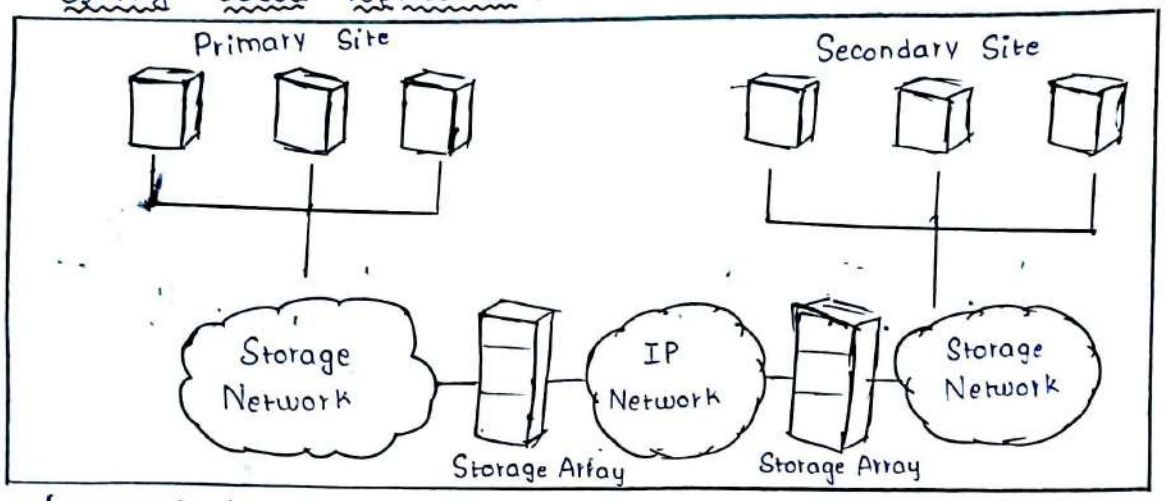
Cloud Deployment Management Tool	Features
RightScale	Design, deploy & manage cloud deployment across multiple public or private clouds.
Scalr	Provides tools to automate the management of servers, monitors servers, replaces servers that fail, provides auto scaling & backups.
Kaavo	Allows deploying applications easily across multiple clouds, managing distributed applications & automating high availability.
Cloud Stack	Allows simple & cost effective deployment management & configuration of cloud computing environment.

6.5 Replication

1. Replication is used to create & maintain multiple copies of the data in the cloud.
2. Replication of data is important for practical reasons such as business continuity & disaster recovery.
3. In the event of data loss at the primary location, organizations can continue to operate their applications from secondary data sources.
4. With real-time replication of data, organizations can achieve faster recovery from failures.
5. Traditional business continuity & disaster recovery approaches don't provide efficient, cost-effective & automated recovery of data.
6. Cloud based data replication approaches provide replication of data in multiple locations, automated recovery, low recovery point objective (RPO) & low recovery time objective (RTO).
7. Cloud enables rapid implementation of replication solutions for disaster recovery for small & medium enterprises & large organizations.

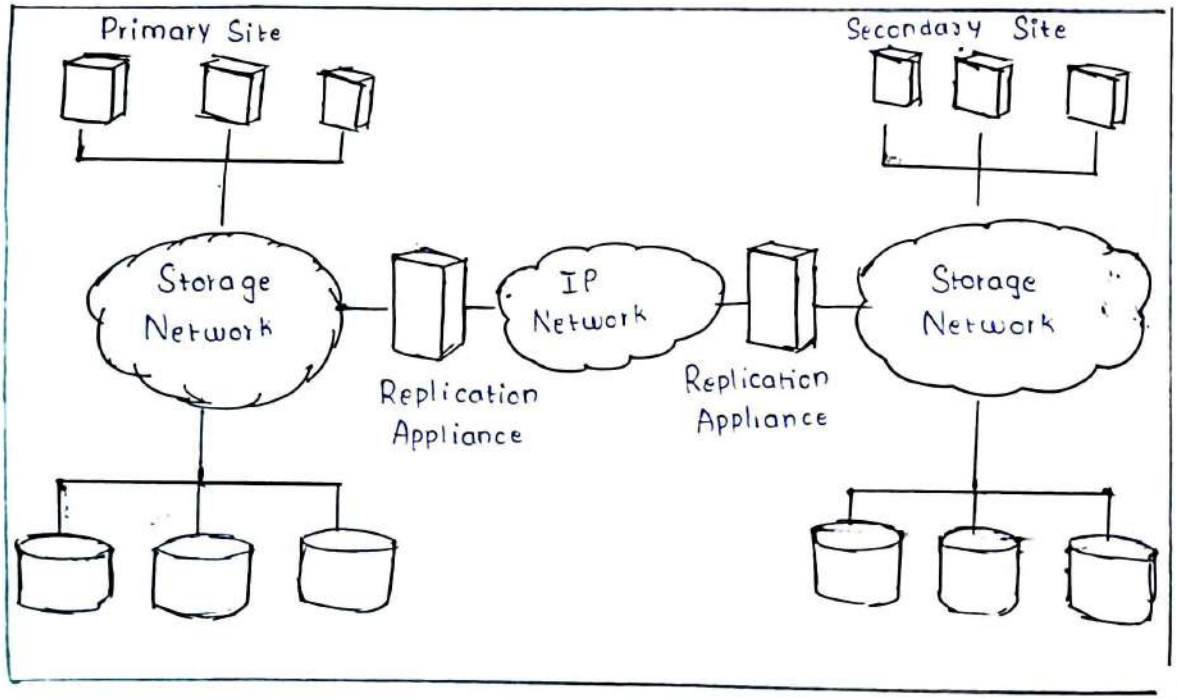
There are three replication approaches as shown below:

(a) Array - based replication:



- Array - based replication uses compatible storage arrays to automatically copy data from a local storage array to a remote storage.
- It can work in heterogeneous environments with different OS.
- Array - based application uses network attached storage (NAS) or Storage area n/w to replicate.
- A drawback of this array - based replication is that it requires similar arrays at local & remote locations, thus for the cost for setting up array - based replication are higher than the other approaches

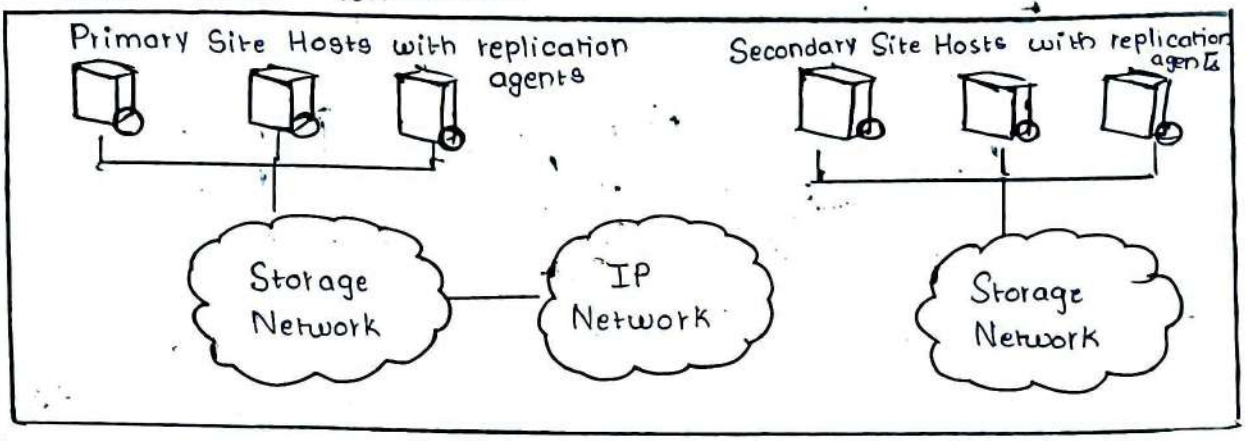
(b) Network - based Replication:



- Network - based replication uses an appliance that sits on the n/w & intercepts packets that are sent from hosts & storage arrays.

- The benefits of this approach is that it supports heterogeneous environments & requires a single point of failure.
- This approach involves higher initial costs due to the replication hardware & software.

(c) Host - based replication :



- Host based replication runs on standard servers & uses s/w to transfer data from a local to remote location.
- Host acts as the replication control mechanism.
- An agent is installed on the hosts that communicate with the agents on the other hosts.
- Host - based replication can be either block - based or file - based.
- Block - based replication typically require dedicated volumes of the same size on both the local & remote servers.
- File - based replication requires less storage as compared to block - based storage.

6.6 Monitoring

1. Cloud resources can be monitored by monitoring services provided by the cloud service providers.
2. Monitoring services allow cloud users to collect and analyze the data on various monitoring metrics.
3. A monitoring service allows, collects data on various system & application metrics from the following cloud computing instances.
4. Monitoring services provide various pre-defined metrics.
5. Users can also define their custom metrics for monitoring the cloud resources.
6. Users can define various actions based on the monitoring data.
7. Monitoring services also provide various statistics based on the monitoring data collected.
8. Monitoring of cloud resources is important because

- Below figure shows a generic architecture for a cloud monitoring service

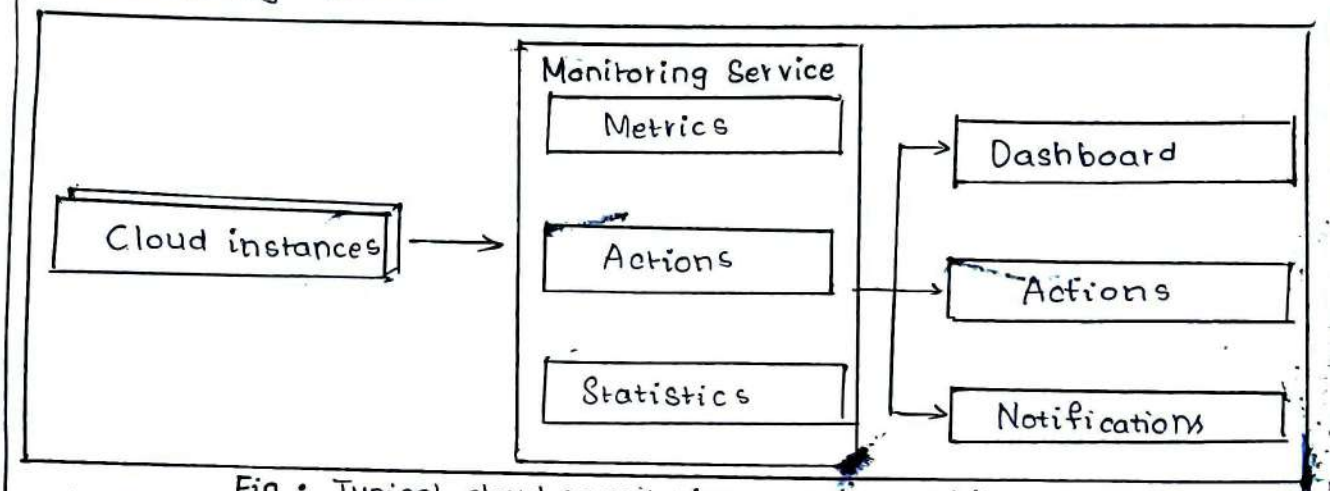


Fig.: Typical cloud monitoring service architecture

- Below table list typical monitoring metrics used for cloud computing resources.

Type	Metrics
CPU	CPU - usage, CPU - Idle
Disk	Disk - usage, Byte/sec (read/write), operations/sec
Memory	Memory - used, Memory - free, Page - cache
Interface	Packets/sec

6.7 Software Defined Networking (SDN):

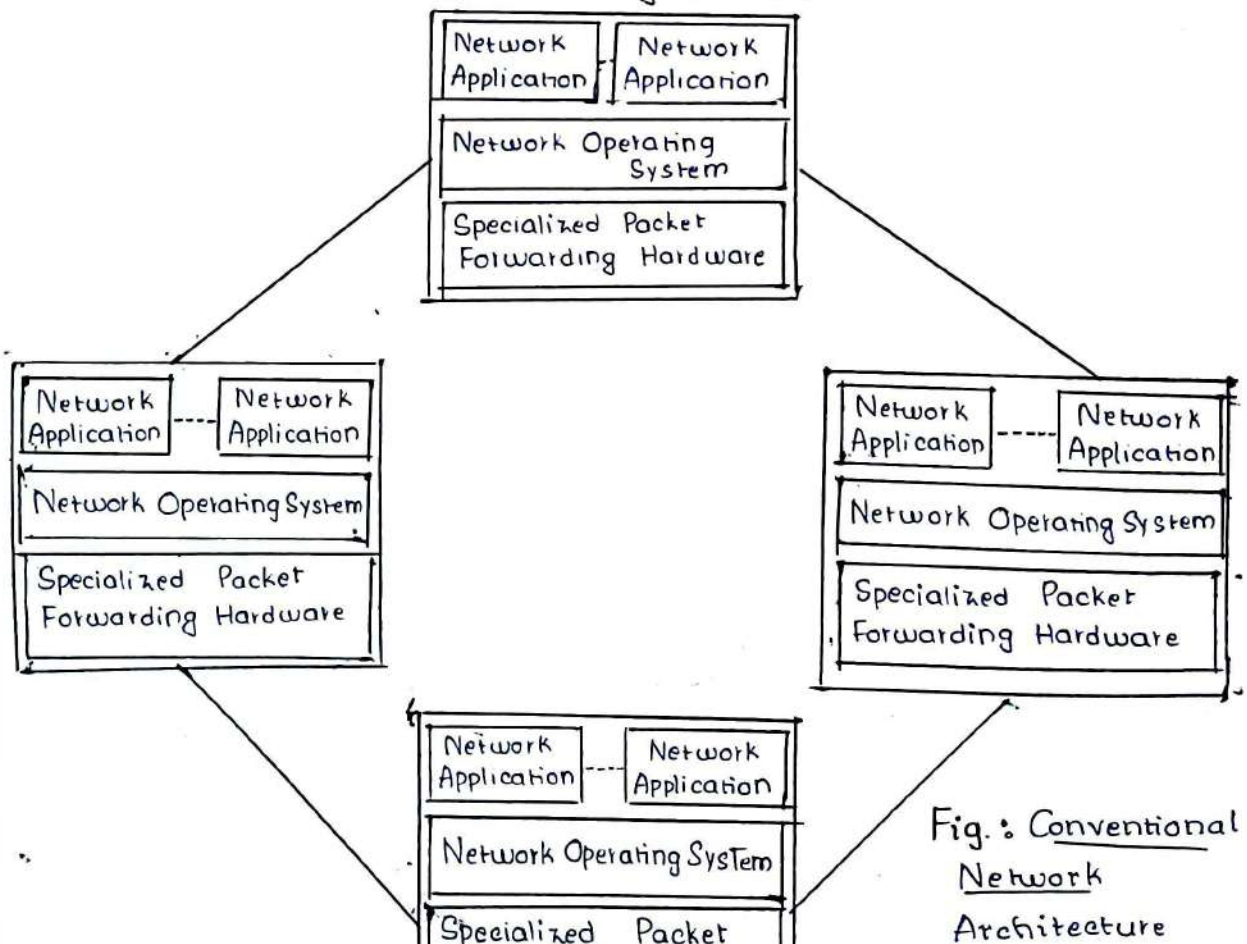


Fig.: Conventional Network Architecture

1. The above figure shows the conventional n/w architecture built with specialized hardware (switches, routers, etc)
2. Network devices in conventional n/w architecture are getting extremely complex with the increasing no. of distributed protocols being implemented & the use of proprietary hardware & interfaces
3. In this architecture, the control plane & data plane are coupled

Limitations of Conventional Network Architecture:

1. Complex Network Devices

- Conventional n/ws are getting more complex with increasing protocols being implemented to improve link speeds & reliability
- Interoperability is limited due to the lack of standard & open interfaces.
- Network devices use proprietary h/w & s/w & have slow product lifecycles with limiting innovation.
- The conventional n/ws were well suited for static traffic patterns & had a large no. of protocols designed for specific applications.
- Due to the complexity of conventional n/w devices, making changes in the n/ws to meet the dynamic traffic patterns has become increasingly difficult.

2. Management Overhead

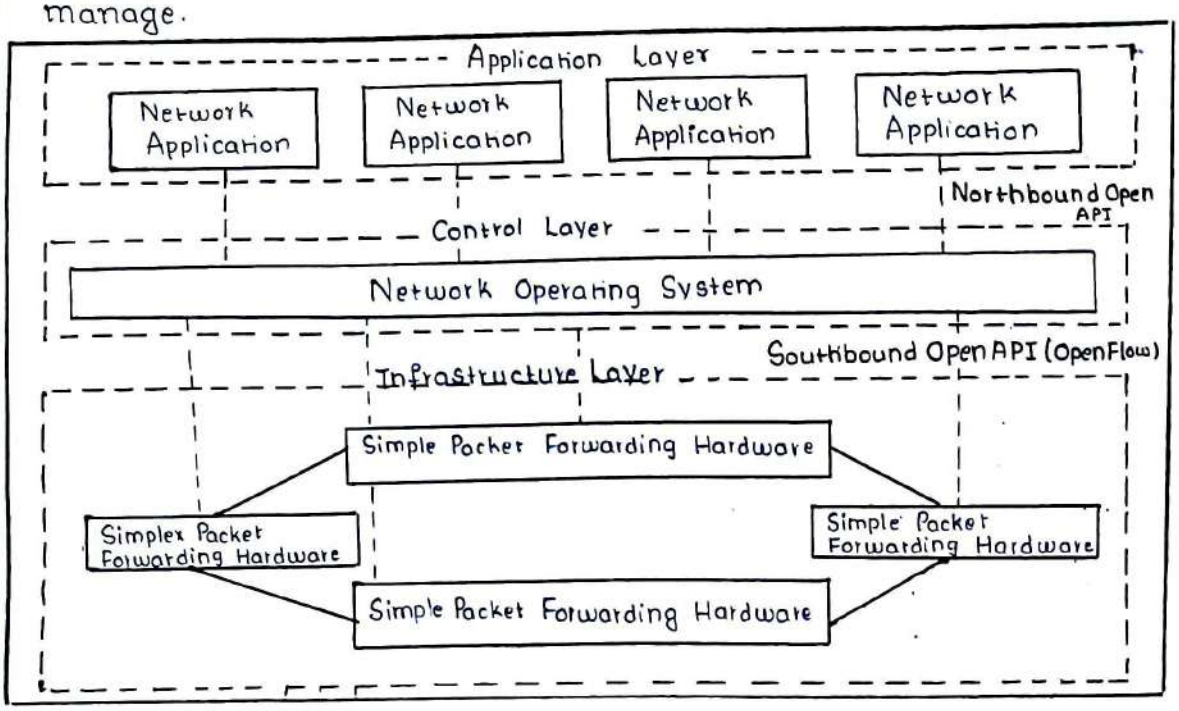
- Conventional n/ws involve significant management overhead.
- N/w managers find more & more difficult to manage multiple n/w devices & interfaces from multiple vendors.
- Upgradation of n/w requires configuration changes in multiple devices (switches, routers, firewalls, etc)

3. Limited Scalability

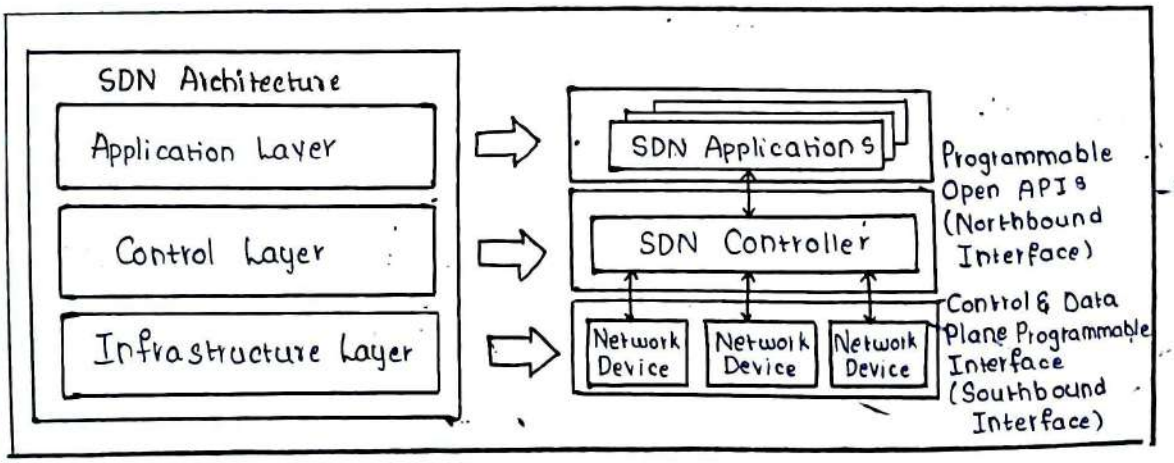
- The virtualization technologies used in cloud computing environments has increased the no. of virtual hosts requiring n/w access.
- Multi-tenanted applications hosted in the cloud are distributed across multiple virtual machines that require exchange of traffic.
- Big data applications run distributed algorithms on a large no. of virtual machines that require huge amounts of data exchange between the virtual machines.
- Such computing environments require highly scalable & easy to manage n/w architectures with minimal manual configurations, which is

→ SDN :

- Software - Defined Networking (SDN) is a networking architecture that separates the control plane from the data plane & centralizes n/w controller.
- Control plane is the part of n/w that carries the signaling & routing message traffic. (Computing routing table).
- Data plane is the part of n/w that carries the payload data traffic (forward traffic).
- SDN attempts to create n/w architectures that are simplex, inexpensive, scalable, agile & easy to manage.



(a) SDN Architecture



(b) SDN layers

- Fig (a) & (b) shows the SDN architecture & SDN layers in which the control & data planes are decoupled & n/w controller is centralized.
- Software-based SDN controllers maintain a certain

- Underlying infrastructure in SDN uses simple packet forwarding hardware as opposed to specialized hardware in conventional network.
- N/w devices become simple with SDN as they do not require implementations of a large no. of protocols.
- N/w devices receive instructions from the SDN controller on how to forward the packets
- These devices can be simplex & cost less as they can be built from standard hardware & s/w components.

Key benefits of SDN :

1. Centralized n/w controller
   - With separated control & data planes & centralized n/w controller, the n/w administrator can rapidly configure the n/w.
   - SDN applications can be deployed through the programmable open APIs.
   - This speeds up innovations on the n/w administrators no longer need to wait for the device vendors to embed new features in their proprietary hardware.

2. Programmable Open APIs
   - SDN architecture supports programmable open APIs for interface between the SDN application & control layers (northbound interface).
   - These open APIs allow implementing various n/w services such as routing, quality of service (QoS), access control, etc.

3. Standard communication interface (OpenFlow)
   - SDN architecture uses a standard communication interface between the control layer & infrastructure layers (Southbound interface).
   - OpenFlow is defined by the open networking foundation ONF is the broadly accepted SDN protocol for southbound interface.
   - With OpenFlow, the forwarding plane of the n/w devices can be directly accessed & manipulated.
   - OpenFlow uses the concept of flow to identify n/w traffic based on pre-defined match rules.

- The below figure shows the components of an OpenFlow & a openflow switch compressed of one (or) more flow tables & a group table, which perform packet lookups & forwarding & openflow channel to an external controller.

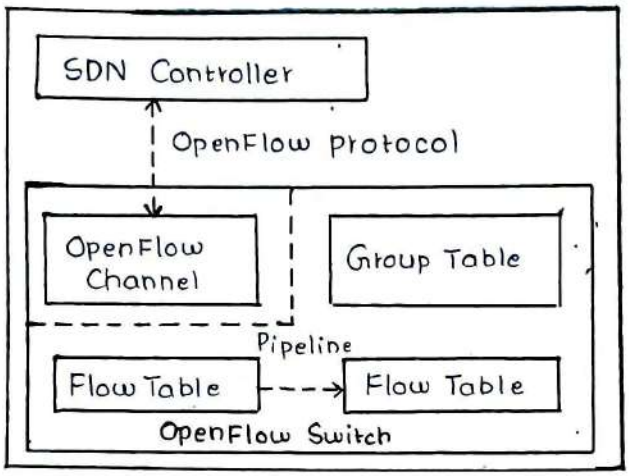


Fig.: OpenFlow Switch

- Open Flow : protocol is implemented on both sides of the interface b/w the controller & the n/w devices
- The controller manages the switch via OpenFlow protocol.
- The controller can add, update, & delete flow entries in flow tables.

Rule	Action	Stats
	1. Forward packet to port 2. Encapsulate & forward to controller 3. Drop packet 4 Send to normal processing pipeline	Packet + Byte Counters

Switch port	MAC Src	MAC dst	Eth type	VLAN ID	IP Src	IP dst	IP Prot	TCP Sport	TCP dport

Fig.: OpenFlow flow table

- The above figure shows an example of an openFlow flow table
- Each flow table contains a set of flow entries.
- Each flow entry consists of match fields, counters. & set of instructions to apply to matching packets.
- Matching starts at the first flow table & may continue to additional flow tables of the pipeline

6.8 Network Function Virtualization

1. Network function virtualization (NFV) is a technology that leverages virtualization to consolidate the heterogeneous n/w ~~level~~ devices onto Industry Standard high volume servers, switches & storage.
2. NFV is complementary to SDN as NFV can provide the infrastructure on which SDN can run.
3. NFV & SDN are mutually beneficial to each other but not dependent.
4. Network functions can be virtualized without SDN, similarly SDN can run without NFV.
5. NFV architecture is being standardized by the European Telecommunication Standards Institute (ETSI).

* Virtualized n/w function (VNF): VNF is a s/w implementation of a n/w function which is capable of running over the NFV Infrastructure (NFVI).

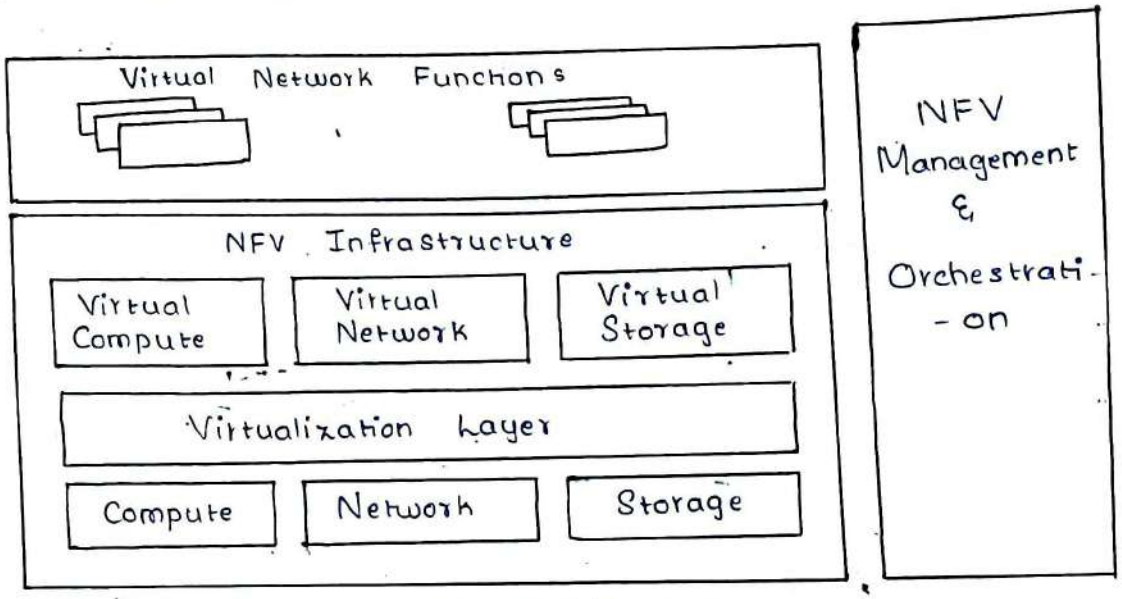


Fig.: NFV Architecture

* NFV Infrastructure (NFVI): It includes computer n/ws & storage resources that are virtualized.
* NFV Management & orchestration: This element focuses on all virtualization-specific management tasks & covers orchestration & lifecycle management of physical & s/w resources that support the infrastructure virtualization & the lifecycle management of VNFs.

- NFV is made up of n/w functions implemented in s/w that run on virtualized resources in cloud.
- N/w functions can be easily tested & upgraded by installing new s/w while the h/w remains same.
- Virtualizing n/w functions reduces the equipment costs & also reduces power consumption.
- The multi-tenated nature of the cloud allows

- NFV is applicable only to data plane & control plane functions in fixed & mobile nlws.
- Following figure shows use cases of NFV for home and enterprise nlws, content delivery nlws, mobile base stations, mobile core nlws & security functions.

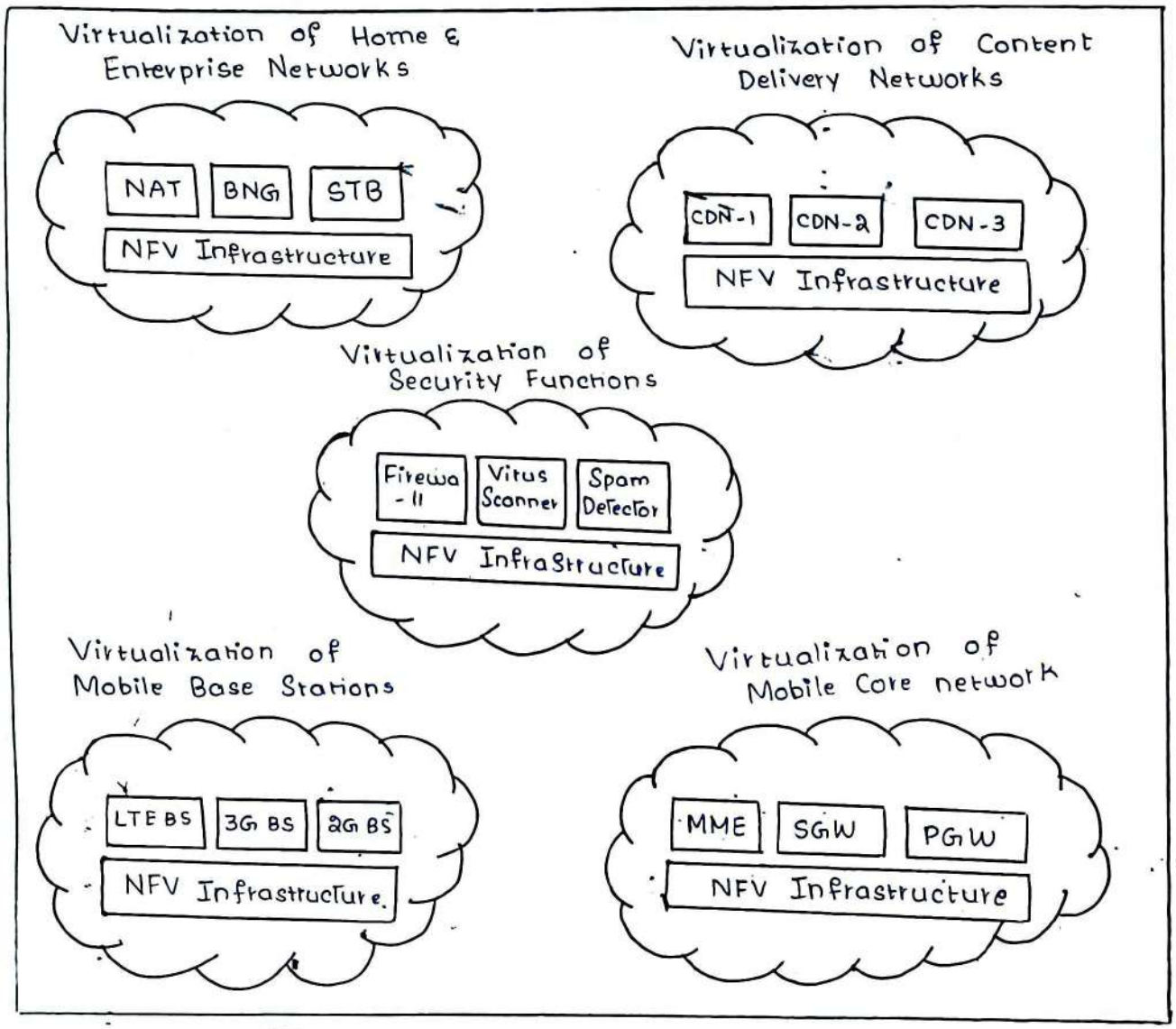


Fig.: NFV usecases

6.9 MapReduce

1. MapReduce is a parallel data processing model for processing & analysis of massive scale data.
2. MapReduce model has 2 parts / phases Map & Reduce.
3. MapReduce programs are written in a functional programming style to create map & reduce functions.
4. The input data to the map & reduce phases is in the form of key-value pairs.
5. Run-time systems (or) MapReduce are large clusters, which are built of commodity hardware.
6. MapReduce run-time systems take care of tasks such as partitioning the data, scheduling of jobs and communication blw nodes in the cluster.

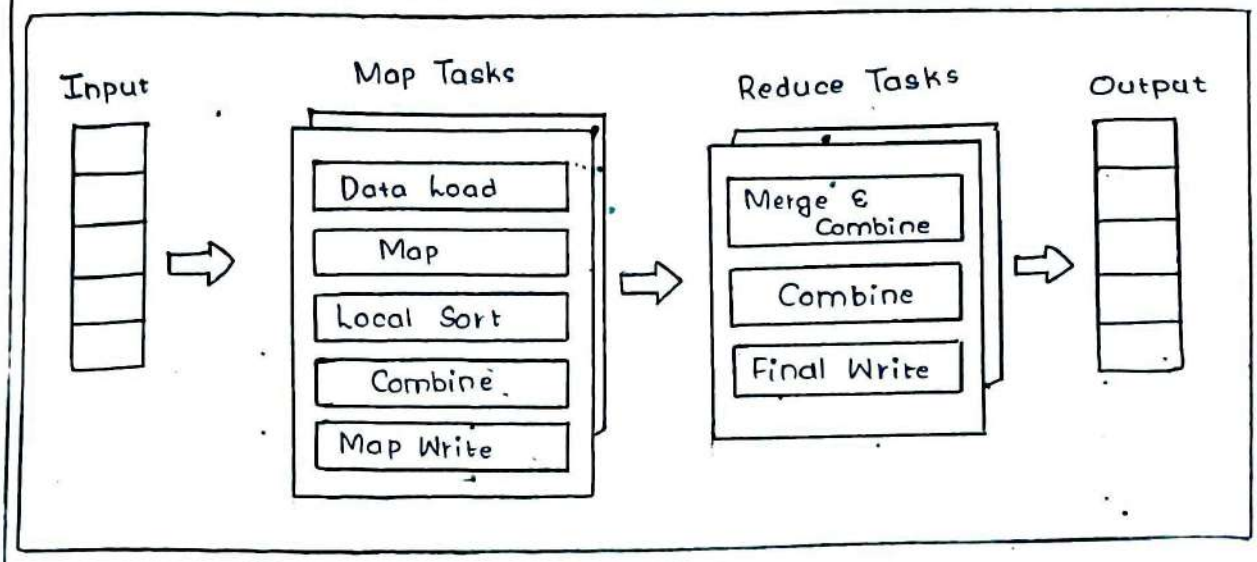


Fig.: MapReduce Work Flow

7. The above figure shows the workflow of MapReduce.
8. In Map phase, data is read from a distributed file system, partitioned among a set of computing nodes in the cluster & sent to the nodes as a set of key-value pairs.
9. The Map tasks process the i/p records independently & produce intermediate results of key-value pairs.
10. The intermediate results are stored on the local disk of the node running the Map task.
11. When all the Map tasks are completed, the Reduce phase begins in which the intermediate data with the same key is aggregated.
12. An optional combine task can be used to perform data aggregated on the intermediate data of the same key for the output of the mapper before transferring the o/p to the Reduce task.

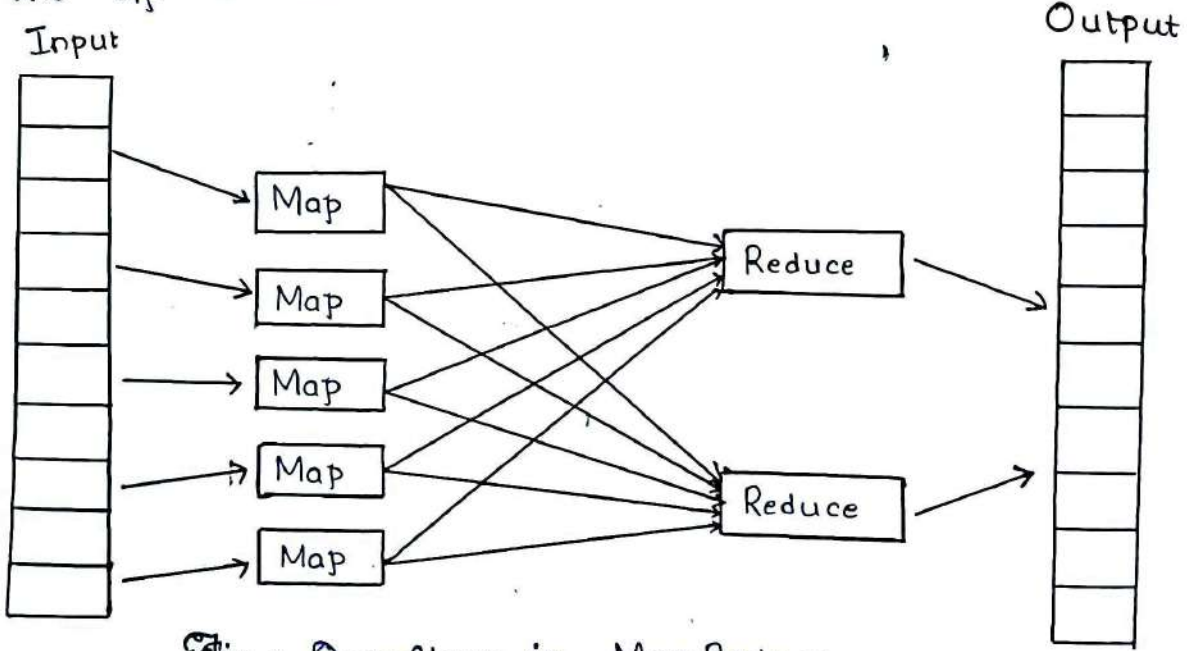


Fig.: Dataflow in MapReduce

13. MapReduce programs take a set of input key-value pairs & produce a set of o/p key-value pairs.

significant of data transmission b/w the nodes in cluster.

15. MapReduce programming model moves the computation to where the data resides thus decreasing transmission of data & improving the efficiency.

6.10 Identity and Access Management (IAM) / IDAM

1. IAM for cloud describes the authentication & authorization of users to provide secure access to cloud resources.
2. Organizations with multiple users can use IDAM services provided by the cloud service providers for management of user identifiers & user permissions
3. IDAM services allow organizations to centrally manage users, access permissions, security credentials & access keys.
4. Organizations can enable role-based access control to cloud resources & applications using IDAM services.
5. It allows creation of user groups where all the users in a group have same access permissions.
6. IDAM is enabled by a no of technologies such as OpenAuth, Role-based Access Control (RBAC), Digital Identities, Security tokens, identity providers etc.

Following figure shows the examples of OAuth & RBAC.

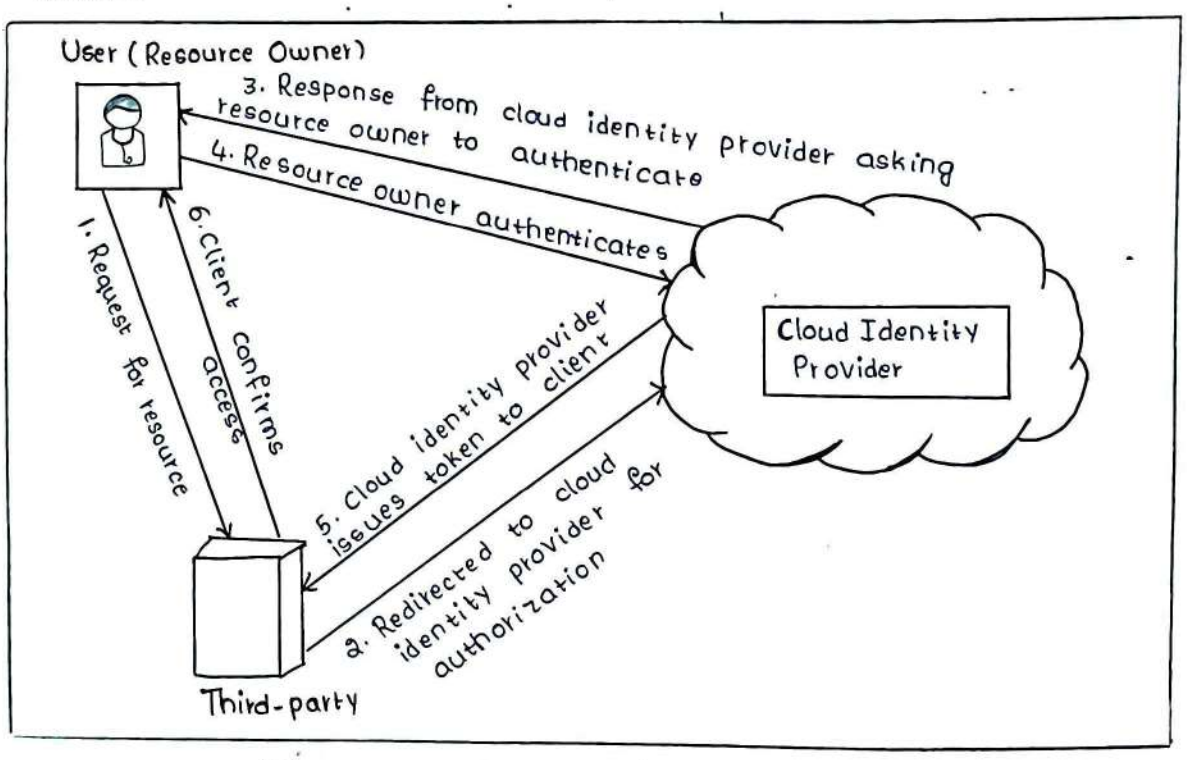


Fig.: OAuth example

- OAuth is an open standard for authorization that allows resource owners to share their private resources stored on one site with another site without handing out the credentials.

- The resource owner grants permission to access the resources in the form of a token.
- Tokens can be issued with a restricted page scope & limited lifetime & canceled independently.
- RBAC is an approach for restricting access to authorized users.

The following figure shows an example of a typical RBAC framework.

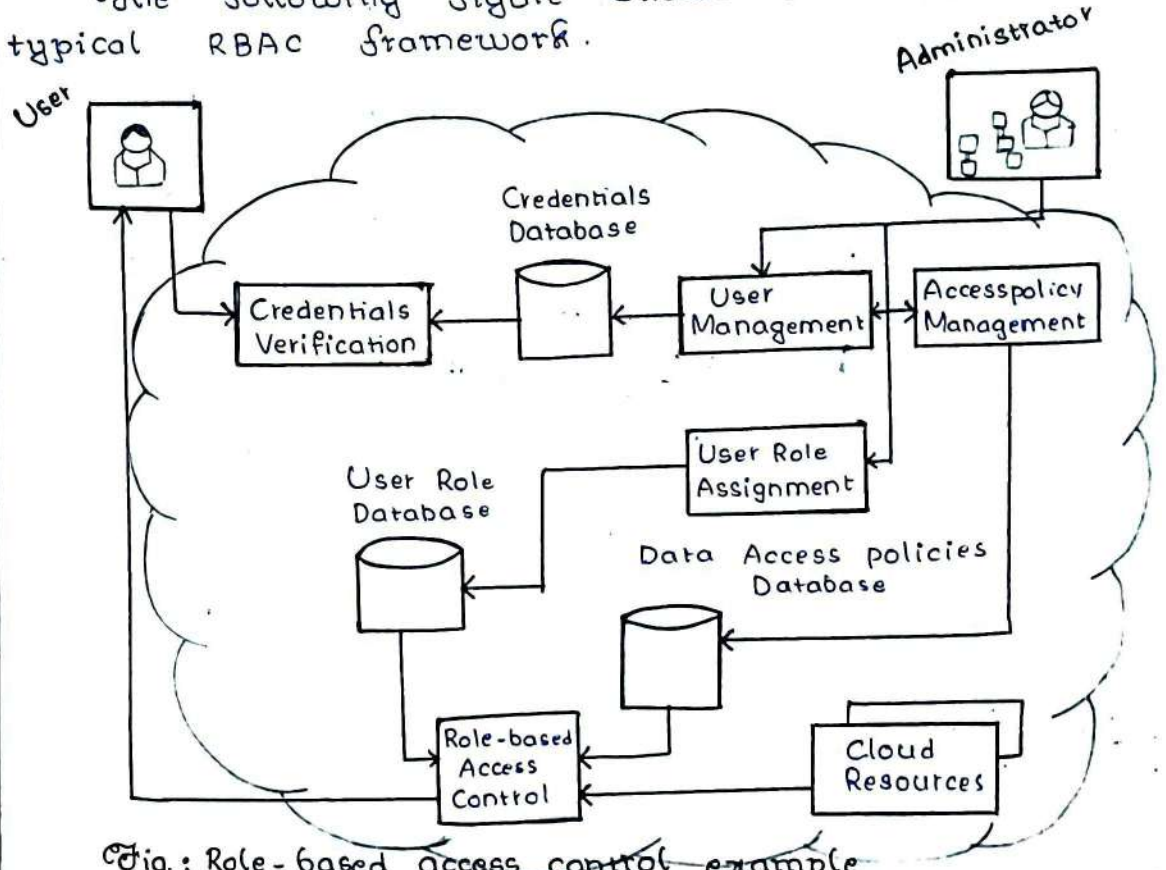


Fig.: Role-based access control example

- An user who wants to access cloud resources is required to send, his/her data to the system administrator who assigns permission & access control policies which are stored in the user roles & data access policies database respectively.

6.11 Service Level Agreements (SLA)

1. A Service Level Agreement SLA for cloud specifies the level of service that is formally defined as a part of service contract with the cloud service provider.
2. SLAs provide a level of service for each service which is specified in the form of minimum level of service guaranteed & a target level.
3. SLAs contain a no. of performance metrics & the corresponding service level objectives.

Following table lists the common criteria cloud

Criteria	Details
Availability	Percentage of time the service is guaranteed to be available
Performance	Response time, Throughput
Disaster Recovery	Mean time to recover
Problem resolution	Process to identify problems, support options resolution expectations.
Security & privacy of data	Mechanisms for security of data in storage & transmission

Table: List of criteria for cloud SLAs

6.12 Billing

Cloud service providers offers a no. of billing models with which are described as follows:

a) Elastic Pricing
b) Fixed Pricing
c) Spot Pricing

a. Elastic Pricing

- In Elastic pricing (or) pay-as-you-go pricing model, the customers are charged based on the usage of cloud resources.
- Cloud computing provides the benefit of on-demand provision of resources.
- On-demand provisioning & elastic pricing models bring cost savings for customers.
- Elastic pricing model is suited for customers who consume cloud resources for short durations & who cannot predict the usage beforehand.

b. Fixed pricing

- In fixed pricing models, customers are changed a fixed amount per month for the cloud resources.
- Fixed pricing model is suited for customers who want to use cloud resources for longer durations & want more control over the cloud expenses.
- For example, fixed amount can be charged per month for running a virtual machine instance, irrespective of the actual usage.

c) Spot pricing

- Spot pricing models offers variable pricing for cloud resources which is driven by market demand.
- When the demand for cloud resources is high, the prices increases & when the demand is lower, prices

Resource	Details
Virtual machines	CPU, memory, storage, disk I/o, network I/o
Network	N/w I/o, load balancers, DNS, firewall, VPN.
Storage	Cloud storage, storage volumes, storage gateways.
Data Services	Data import / export services, data encryption, data backup
Security services	Identity & access management, isolation
Support	SLA, Fault tolerance.
Application Services	Notification service, workflow service, payment service.
Deployment and management services	Monitoring service, deployment service.

Cloud Services and platforms :

1. 1. Compute Services :-

→ Compute Services provide dynamically scalable Compute Capacity in the cloud

→ Compute resources Can be provisioned on-demand in the term of virtual machines

→ Virtual machines Can be Created from standard images provided by cloud Service provider (or) Custom images Created by the users

→ A machine image is a template that Contains a software Configeration.

→ Compute Services Can be accessed from the web Consoles of these Services that provide graphical user interfaces for Provisioning, Managing and monitoring these Services.

→ cloud Service providers also provide API's for various programming languages that allow developers to access and manage these services programmatically.

features :-

1. Scalable :- Compute Services allow rapid provisioning of virtual machine instances as required the provisioned capacity Can be Scaled up (or) down based on the work load levels. Auto scaling Policies Can be defined for Compute Services that are triggered when the monitoring metrics go above predefined thresholds.

2. Flexible :- Compute Services give a wide range of options for Virtual machines with multiple instance types, operating systems etc

3. Secure :- Compute Services provide Various Security features that Control the access to the Virtual machines instances Such as Security apps, access Control lists, network firewalls etc, Users Can Securely Connect to the instances with SSH Using authentication machines Such as OAuth and keypairs.

Cost Effective :- cloud Service producers after Various billing options such as on-demand instances, reserved instances Spot instances etc.

Amazon Elastic Compute cloud (AEC$_2$)

1. AEC$_2$ is a Compute Service provided by Amazon.
2. The below figure shows the Screenshot of the Amazon EC$_2$ Console.
3. To launch a new instance click on the launch instance button this will open a wizard where you Can Select the Amazon Machine Image (AMI) with which you need to launch the instance.

5. when you launch an instance you need to Specify the instance type whether it is micro, Small, medium, large, extra-large etc
6. the number of instances launched will be based on the Selected AMI and availability of regions/zones for the instances.
7. when launching a new instances, the User selects a key pair from existing key pairs (or) Creates a new key pair for the instances
8. keypairs are used to Securily Connect to the instance after it launches
9. Security groups are used to open (or) block a Specific network Port for the launched instances
10. when the instance is launched its status can be viewed in the EC_2 Console
11. After launching a new instance, its State will be in pending, as it takes Couple of minutes for instance to Come into require state.
12. when the instance Comes into the running state, it is assigned a public DNS, private DNS, public IP and private IP. the public DNS can be used to Securely Connect to the instances using SSH.

Google Compute Engine (GCE)

1. Google Compute Engine is a Compute Service provided by google
2. the below figure shows a Screenshot of the Google Compute Engine (GCE) Console,
3. GCE Console allows users to Create and manage Compute instances
4. to Create a new instance, the User selects an instance Machine type, a zone in which the instance will be launched a machine image for the instance and provides an instance name, instance tags and meta data
5. Every instance is launched with a disk resource.
6. Depending on the instance type, the disk resource Can be a Scratch disk Space.
7. Network options allows you to Control the traffic to and from the instances

Windows Azure Virtual machines :-

1. Windows Azure Vms is the Compute Service from microsoft
2. the below figure Shows a Screenshot of the google Compute

3. GCE Console allows users to Create and manage Compute Engine (GCE) Instances
4. To Create a new Instance, the User Selects an Instance Machine types, a zone in Which the Instance will be launched, a Machine Image for the Instance and provides an Instance name, Instance tags and meta data
5. Every Instance is launched, a machine With a disk resource
6. Depending on the Instance type, the disk resource Can be a Scratch disk Space or persistent disk Space.
7. Network options allows you to Control the traffic to and from the instances.

Windows Azure Virtual Machines :-

1. Windows Azure VMs is the Compute Services From Microsoft
2. the below figure shows the Screenshot of windows Azure Virtual machines Console.
3. to Create a new Instance, Select the Instance type & the machine Image.
4. you Can provide a User name and password for Securely Connecting to the Instances.
5. Any changes made to the VM are persistently Stored and new VMs Can be Created from the previously Stored machine images.

2. Storage Services :-

→ Cloud Storage Services allows storage and retrival of any amount of data, at any time from anywhere on the web.
→ Most cloud storage Services organize data into buckets or Containers.
→ Buckets (or) Containers store objects which are individual pieces of data.

features :

1. Scalability : Cloud Storage Services provide high Capacity and Scalability. objects upto Several tera-bytes in size Can be Uploaded and multiple buckets | Containers Can be Created on cloud Storage.
2. Replication :- When an object is uploaded it is replicated at multiple

3. Access policies : cloud storage services provide several security features such as access control lists (Acls) bucket/Container level policies etc.

4. Encryption : cloud storage services provide server side Encryption (SSE) options to Encrypt all data stored in the cloud storage

5. Consistency : strong data Consistency is provided for all upload and delete operations.

Amazon Simple Storage Services

1. Amazon Simple Storage Service (S_3) is an online cloud based data storage infrastructure for storing and retrieving any amount of data.
2. S_3 provides highly reliable, scalable, fast, fully redundant and affordable storage infrastructure.
3. data stored on S_3 is organized in the form of buckets.
4. you must Create a bucket before you Can store data on S_3
5. S_3 Console provides simple wizards for Creating a new bucket and Uploading class-files.
6. you Can Specify the redundacy and Encryption options and access permissions.

Google cloud Storage

1. objects in GCS are organized into buckets.
2. Acls are used to Control access to objects and buckets
3. Acls Can be Configured to share objects and buckets with the entire world, a Google group (or) a Specific google account holders.

Windows Azure storage

1. Windows Azure storage is the cloud storage services From Microsoft.
2. it provides Various storage services such as blob storage Service, table Service and Queue Service
3. the blob storage service allows storing unstructured binary data large objects (blobs).
4. blobs are organized into Containers. two kinds of blobs Can be stored

   → block blobs

5. A Blob Can be Subdivided into Some number of blocks If a failure occurs while transferring a block blob, retransmission Can resume with the most recent block rather than Sending the entire blob again.

6. page blobs are divided into number of pages and are designed for random access.

3. Database Services :-

1. cloud database Services allows you to setup and operate relational (or) non relational databases in the cloud.
2. The benefit of Using cloud database Services is that it relieves the application developers from the time Consuming database administration tasks.
3. Relational databases provided by cloud Service providers include My SQL, Oracle, SQL Server.
4. Non-Relational (No-SQL) databases are Usally fully Managed and deliver Seamless throughput and Scalability.

features

1. Scalability :- cloud database Services allow provisioning of Compute and Storage resources as required to meet the application workload levels. provisioned Capacity Can be Scaled up (or) down for reading heavy workloads, readreplicas can be Created.
2. Reliability :- cloud database Services are reliable and provide automated backup and Snapshot options.
3. performance :- cloud database Services provide guaranteed performance with options Such as guaranteed input/output operations per Second (Iops) which Can be provisioned up front
4. Security :- cloud database Services provide Several Security feature to restrict the access to the database instances and stored data, Such as Network firewalls and authentication Mechanisms.

Amazon relational Data Store (Amazon RDS)

* The Amazon RDS Console provides an instance launched wizard that allows you to select the type of database to Create database instance size, allocated storage, DB instance identifier, DB Username and password.
* the status of the launched DB instances Can be viewed from the Console.
* it takes Several minutes for the instances Can to become available.
* Once the instance is available, you Can note the instance end Point from the instance property tab.

AMAzon Dynamo-DB :-

* Amazon DynamoDB is the non-relational (No-SQL) database Service from Amazon.
* the dynamODB data model includes include tables, items and attributes.
* A table is a Collection of items and each item is a Collection of attributes.
* To Store data in DynamoDB you have to Create a one (or) more tables and Specify how much throughput Capacity you want to Provision and reserve for reads and writes.
* DynamoDB is a fully managed Service that automatically Spreads the data and traffic for the Stored tables over a number of Servers to meet the throughput requirements Specified by Users.
* All Stored data is automatically replicated across multiple availability zones to provide data durability

Google cloud SQL

* Google SQL is the retational database Service from google
* it allows you to host My SQL database in Google Service cloud.
* it provides both Synchronous (or) Asynchronous geographic replication and the ability to import | export databases
* you Can Create new database instances from the google cloud SQL Console and manage existing instances

Google cloud Datastore :

* Google cloud Datastore is fully managed non-relational database from Google
* cloud Datastore offers ACID transactions and high availability of reads and writes.
* the cloud datastore data model Consists of entities. Each entity has one (or) more properties (key Value pairs) which Can be of several datatypes, such as strings and integers
* Each Entity has a kind and a key
* the entity kind is used for Categorizing the entity for the queries.
* the entity key uniquely identifies the entity.

Windows Azure SQL database :

* Windows Azure SQL database is the relational database Service from microsoft
* it is based on SQL Server, but it does not give each Customer a Separate instance of SQL Server.
* Instead the SQL database is a multi-tenant Service, with a logical SQL database Server for each Customer.

Windows Azure table Service

* it is a non relational (No-SQL) database Service from micro-Soft.
* this data model Consists of tables with multiple Entities.
* tables are divided into Some number of partitions, each of which Can be Stored on a Separate Machine.
* Each partion in a table holds a Specific number of Entities each Containing as many as 255 properties.
* Each property Can be one of the datatypes such as integers and string.
* tables do not have a fixed Schema and different entities in a table Can have different properties.

4. Application Services :

Application Runtime & frameworks :

1. Cloud based application runtimes and frameworks allow developers to develop and host applications in the cloud.
2. Application runtimes provide Support for programming languages
3. Application runtimes automatically allocate resources for applications and handle the application Scaling, without the need to run and maintain Servers.

Google App Engine : Google App Engine is the platform as a Service (Paas) from google, which includes both an applications runtime and webframeworks.

features :

* Runtimes : App Engine Supports applications developed in Java, Python PHP and Go programming languages. App Engine Provides runtime environment for java, python, PHP and Go Programming Language.
* Sandbox : the Sandbox environment provides a limited access to the underlying operating System. App Engine can only execute application Code Called from HTTP requests. it allows App Engine to distribute web requests for the application across multiple Servers.
* web frameworks : App Engine provides a Simple python web application framework called webapp2. App Engine Supports any framework written in pure python that Speaks WSGI, including Django cherrypy, pytons, web.py, and web2py.
* datastore : App Engine provides a No-SQL data storage Service.
* Email Service : Email Service allows applications to Send email Messages.
* Authentication : App Engine applications Can be integrated with google Accounts for user authentication.
* URL fetch Service : URL fetch Service allows Application to

Image Manipulation Service : it allows application to resize, crop, rotate, flip and enhance images.

Memcache : Memcache Service is a high performance in Memory key-Value cache Service that application Can Use for caching data items that do not need a persistent storage.

Task queues : task queues allow applications to do work in the background by breaking up work into Small, discrete units, Called tasks which are enqueued in task queues.

Scheduled Task Service : This Service allows applications to perform tasks at defined times (or) regular intervals.

Windows Azure web Sites

1. windows Azure web site is platform-as-a-Service (Paas) from Microsoft.
2. Azure websites allows you to host web application in the Azure cloud
3. Azure website provides shared and Standard options.
4. In the shared option, Azure web Sites run an a set of virtual Machine that may Contain multiple web sites Created by multiple Users.
5. In the Standard option, Azure web Sites run on virtual machines (VMs) that belong to an individual user.
6. Azure web Sites Supports applications Created in ASP.NET, PHP, Node.js and python programming languages.

Queuing Services :

1. cloud-based Queuing Service allow de-Coupling Components
2. the de-coupled Components Communicate via messaging queues.
3. Queues are Usefull for asynchronous processing.
4. A Queue is a place where you Can Store your messages untill they are extracted from the queue (or) expired.
5. Queues are Used to Store textual information. So it can be received and Used by a Customer later on.
6. Queuing Services from various cloud Service providers, allow short Messages of a few kB in size.
7. Message Can be enqued and read from the queue Simultaneously
8. the enqueued Messages are retained for a Couple of days to weeks

4.

Amazon Simple Queue Service :- (Amazon SQS)

1. Amazon Simple Queue Service (SQS) is a queuing Service from Amazon
2. SQS is a distributed queue that Supports messages of upto 256kB in size
3. SQS Supports multiple writers and reader and locks messages while they are being processed.
4. Application that require FIFO ordering of Messages can place additional Sequencing information in each Message so that they can be re-ordered after retrieving from a queue

Google Task Queue Services

1. it is queuing Service from google and is a part of google App Engine platform.
2. Task queues allow applications to execute tasks in background
3. Task is a unit of work to be performed by an application
4. the task objects consists of application - Specific URL with a request handler and an optional data payload for the tasks.
5. they are two different Configurations for task queues push Queue and pull Queue.
6. push Queue is the default queue that processess tasks based on the processing rate configered in the queue.
7. pull Queues allow task Consumers to lease a Specific number of tasks for a specific duration the tasks are processed and deleted before the lease ends.

Windows Azure Queue Services

1. Windows Azure Queue Service is a queuing Service from Microsoft
2. Azure Queue Services allows storing large number of messages that can be accessed from anywhere in the world via authenticated calls using HTTP or HTTPS.

EMail Services

Email Services allow applications hosted in

Amazon Simple Email Services

1. Amazon Simple Email Services is bulk and transactional Email-Sending Service from Amazon
2. SES is an outbound-only email-sending service that allows application hosted in the Amazon Cloud to send email such as marketing emails, transactional emails and other types of Correspondance.
3. to ensure high email deliverability, SES uses Content filtering technologies to scan the outgoing email Messages to ensure that they do not Contain material that is flagged as questionable
4. SES Service Can be accessed and Used from the SES Console.

Google email Service

1. Google Email Service is part of the Google App Engine platform that allows App Engine applications to send emails messages on behalf of app's administrators and Users with google accounts
2. Apps Send Messages Using the mail Service and receive messages in the form of HTTP requests initiated by App Engine and Posted to the app Engine and posted to the app

Notification Services

1. Cloud-based Notification Services (or) push Messaging Services allow applications to push Messages to internet Connected Smart devices Such as Smartphones, tablets etc.
2. push messaging services are based on publish-Subscribe model in which Consumers Subscribe to Various channels provided by a Publisher/producer.
3. Whenever new Content is available on one of these channels, the notification Service pushes that information out to the Consumer.
4. push notifications are Used for Smart devices as they help in displaying the latest information
5. Consumer application on Such devices Can increse their Consumer engagement with the help of push notifications.

Amazon Simple notification Service

1. Amazon Simple Notification Service (SNS) is a push messaging Service from Amazon
2. SNS has two types of clients - publishers and Subscribers

4. topic is a Communication channel
5. Subscribers are the Consumers who Subscribe to channels to receive notifications
6. SNS Can deliver notifications as SMS, email, etc.

Google cloud Messaging

1. Google cloud Messaging for Android provides push messaging for Android devices.
2. Gcm allows applications to Send data from the application Server to their User's Android devices, and also to receive messages from devices on the Same Connection.
3. Gcm provides a 'Send-to-Sync' Message Capability that Can be Used to inform an application to Sync data from the Server.
4. Gcm for chrome is another notification Service from Google that allows message to be delivered from the cloud to apps.

Windows Azure Notification Hubs :

1. Windows Azure notification Hubs is a push notification Service from microsoft that provides a Common interface to Send notifications to all major mobile platforms including windows Store, ios and Android.
2. platform notification Systems (PNS) are used to deliver notification Message.
3. Devices register their PNS handles with the notification Hub.
4. Each Notification hub Contains Credentials for each Supported PNS.
5. these Credentials are used to Connect to the PNS and Send push Notifications to the applications.

Media Services.

Cloud Service providers provide Various types of media Services that Can be Used by the applications for manipulating, transforming (or) transCoding media Such as images Vedios etc.

Amazon Elastic transCoder :

1. Amazon Elastic transcoder is a cloud-based Video transCoding Service from Amazon
2. Elastic transCoder Can be Used to Convert Video from their

3. Elastic transcoders provides a number of pre-defined transcoding Presets.
4. Transforming pipelines are used to perform multiple transforms in Parallel.
5. Elastic transcoder works with Amazon S3 Storage where the input and output Videofiles are stored.

Google images Manipulation Services

1. Google image manipulation Service is a part of google App Engine Platform.
2. Image manipulation Service provides the Capability to resize, Crop, rotate, flip and enhance images.
3. the image Service Can accept images data directly from the google cloud Storage
4. Image Services accepts images in Various formats including JPEG, PNG, WEPB, GIF & BMP formats and Can return transformed images in JPEG, WEBP & PNG formats.

Windows Azure media Services :-

1. Windows Azure media Service provides Various media Services Such as encoding & format Convertion, Content protection and on demand and live Streaming Capabilities.
2. It provides application the Capability to build media workflows for Uploading, Storing, encoding, format Conversion, Content protection and media delivery.
3. to Use Azure media Services, you Can Create jobs that process media Content in Several ways Such as encoding, encrypting doing formats Converlions etc.
4. Each media Services has one (or) more tasks Each task has Peset string, an input asset and an Dutput asset
5. Media assets in the Azure media Service can be delivered either by download (or) by streaming.

5. Content Delivery Services

1. Cloud based Content delivery Service include content Delivery Networks (CDNs).
2. A CDN is a highly distributed platform of services that helps minimize delays in loading web page Content by reducing the Physical distance between the Server and user.
3. this helps users around the world view the Same high quality Content without slowing down
4. CDNS are useful for Serving static Content Such as text images, Scripts etc ; and streaming media.
5. CDN's enables global Search, Saves a lot of money, 100% available, decreses Server load and provides 24/7 Customer Support

Amazon cloudfront

1. Amazon Cloudfront is a Content delivery Services from Amazon
2. Cloudfront Can be Used to deliver dynamic, Static and Streaming Content Using a global network of edge locations
3. the Content in cloudfront is organized into distributions each distribution Specifies the original location of the Content to be delivered which Can be An Amazon S3 bucket, an Amazon EC_2 instance, (or) an Elastic load balancer (or), your own origin Server.
4. distributions Can be accessed by their domain names
5. cloudfront helps in improving the performance of websites in Several ways.

⇒ By Caching the Static Content at the Edge location

⇒ By proxing requests for dynamic (or) interactive Content back to the origin running in the Aws Cloud

Windows Azure Content Delivery Network

1. windows Azure Content delivery Network (CDN) is the Content delivery Service From microsoft.
2. it Catches windows Azure bolbs and Static Content at the Edge locations to improve the performance of web
3. it Can be enabled on a windows Azure Storage account

6. Analytics Services

1. Cloud based analytics Services allows analyzing Massive data sets stored in the cloud either in cloud storages (or) in cloud databases using programming models such as map reduce
2. Using cloud analytics Services applications Can perform data Intensive tasks Such as data mining, log file analysis, machine learning, web indexing, etc.

Amazon elastic Mapreduce

Amazon Elastic MapReduce is the map reduce Services from Amazon based on the Hadoop-frame work running on Amazon EC_2 and Amazon S_3

EMR Supports Various Job types. Such as Customer JAR, & Hive program, Streaming job, pig Programs and HBase.

-> the EMR Console provides a Simple wizard for Creating new Mapreduce Job flows.

-> to Create a MapReduce Job enter the Job name, Select the Streaming option for the Job Flow, Specify the locations of input, output and the mapper and reducer programs and Specify the mapping number of nodes to use in the Hadoop cluster and the instance sizes.

=> the Job flow takes Several minutes to launch and Configure

-> A Hadoop cluster is Created as Specified in the Job flow the Map reduce program Specified in the input is executed

-> On Completion of the mapreduce Job. the result are Copied to the output location Specified and the Hadoop cluster is terminated.

Google Mapreduce Services :-

1. Google mapReduce Service is a part of the App Engine platform
2. App Engine Map reduce is optimized for App Engine environment and provides Capabilities Such as automatic Sharding, standard data input readers, Standard Output writers etc.
3. the mapReduce Service Can be accessed using the Google. MapReduce API
4. to execute MapReduce Job mapreduce pipeline objects is Represented within the App Engine application.
5. MapReduce pipeline Specifies the mapper, reducer, data input reader, output writer.

2. it allows quering datasets Using SQL-like Queries
3. the BigQuery queries are run against append-only tables and Use the processing power of google's infrastructure for Speeding up Queries.
4. to query data, it is first loaded into BigQuery Using the bigQuery Console.
5. Data Can be either in Csv (or) JSON format the uploaded data Can be queried Using BIGQuery's SQL dialect.

Windows Azure HDInSight :-

1. windows Azure HDISight is an analytics service from Microsoft
2. HDInSight deplays and provisions Hadoop cluster in the Azure cloud and makes Hadoop available as a Service
3. HDInsight Service Uses windows Azure blob storage as the default file System. It provides interactive consoles for both Javascript and Hive.

7. Deployment & Management Services

cloud based deployment & management services allow to easily deploy and Manage applications in the cloud. these Services automatically handle deployment tasks such as Capacity provisioning, load balancing, auto-Scaling and application health monitoring.

Amazon Elastic Beanstalk :-

1. Amazon provides a deployment Service called Elastic Beanstalk that allows you to quickly deploy and manage applications in the Aws cloud.
2. Elastic Beanstalk Supports Java, PHP, NET, Node.js, python and Ruby applications
3. With Elastic Beanstalk you just need to upload the application and Specify Configeration Settings in a Simple wizard and the Service automatically handles instances provisioning, Service Configeration load balancing & monitering.
4. the launch wizard allows you to Specify the Environment details Such as name, URL, application file, Container type, instance type etc.
5. When the environment is launched Elastic Beanstalk automatically Creates a new load balancer, launches and Configures application Package on the application Services.

automatically launches new application Server to handle the increased lead.

7 If the lead decreases, Auto Scalling stops additional instances and leaves at least are instance running.

Amazon Cloudformation :-

1. Amazon cloudformation is a deployment management Service from Amazon.
2. with cloudfront you Can Create deployments from a Collection OF Aws resources Such as Amazon Elastic Compute cloud, Amazon Elastic Block Store, Amazon Simple Notification Service, Elastic load Balancing and Auto Scaling.
3. A Collection of Aws resources that you want to manage together are Organized into a Stack.
4. Cloudformation Stacks are Created From cloud Formation templates.

Identity & Access Management Services (IDAM)

- IDAM Services allows managing the authentication and Authorization of Users to provide Secure access to cloud resources.
- IDAM Services are useful for organizations which have multiple Users who access the Cloud resources.
- Using IDAM Services you Can manage user identifiers, User permissions Security Credentials and access keys

Amazon Identity & Access Management :-

1. Aws Identify and Access management (IAM) allows you to manage Users and User permissions for an Aws account
2. With IAM you Can Manage Users, Security credentials Such as access keys, & permissions that Control which Aws resources Users Can Create.
3. Using IAm you Can Control what data Users can access and what resources users Can Create, & also allows you to Control Creation, rotation, & revocation Security Credentials of users.

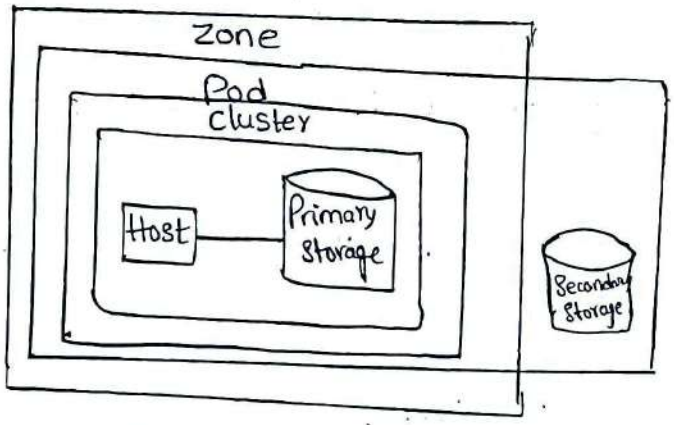
Windows Azure Active directory :-

1. windows Azure Active directory is an Identity & Access management Service from Microsoft
2. It provides a cloud based identity. provider that easily integrates with your on-premises active. directory. deployements and also provides Support for third party. Identity providers

9) Open Source private cloud Software

Cloud stack:

1) Apache cloudstack is an open Source cloud Software that can be Used for Creating private cloud.
2) it manages the Network, Storage and Compute nodes that make up a cloud Infrastructure
3) A cloud stack installation Consists of a management Server and the cloud infrastructure that it manages.
4. the cloud infrastructure can be as simple as one host running a large cluter of hundreds of hosts
5. the Management Server allows you to Configure and manage the cloud resources.



6. the above figure shows the architecture of cloud stack which is basically the management Server.
7. the management Server manages one (or) more zones where each Zone is a Single datacenter.
8. Each zone has one (or) more pods, A pod is a rack of hardware Consists of a Switch and one (or) more clusters.
9. A cluster Consists OF one (or) more host and a primary storage
10. A host is a Compute node that runs guest Virtual machines, the primary Storage of a cluster stores the disk Volumes for all virtual machines running on the host in the cluster.
11. Each zone has Secondary Storage that stores templates ISO images & disk Volume Snapshots

Eucalyptus :-

Eucalyptus is an open Source private cloud Software for building private and hybrid clouds that are Compatible with Amazon web Service API's.

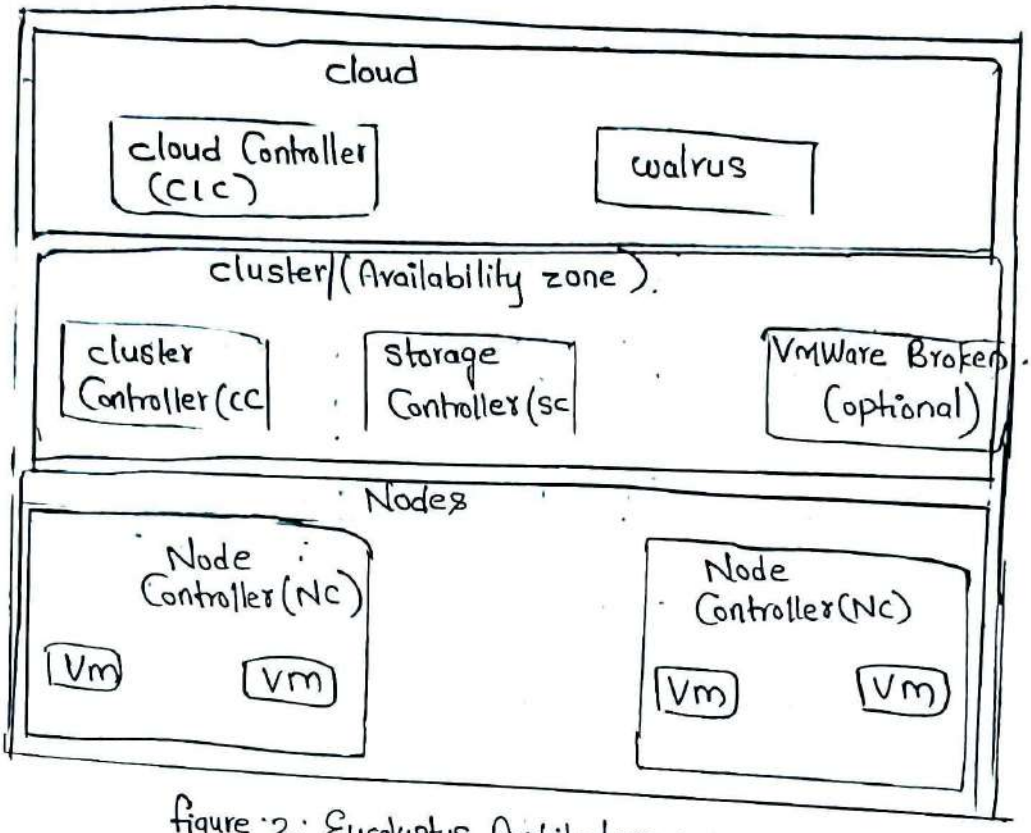


figure 2: Eucaluptus Architecture.

1. In the above figure Node Controller (NC) hosts the virtual machine instances and manages the virtual Network end points.
2. the cluster-level Consists of three Components - cluster Controller (CC) Storage Controller (SC) and Vmware Broker.
3. the CC manages the virtual machines and is the front end for a cluster.
4. the SC manages the Eucalyles block Volumes and Snapshots to the instances within its Specific cluster SC is equivalent to Aws Elastic Block store (EBS).
5. the Vmware Broker is an optional Component that provides an Aws-Compatible interface for Vmware Environments.
6. At the cloud-level there are two Components, that cloud Controller (CLC) and walrus.
7. CLC provides an administrative interface for cloud management and Performs high-level resources Scheduling, System accounting, authentication & Quota management.
8. walrus is equivalent to Amazon S3 and Serves as a persistant Storage to all of the virtual machines in the Eucalyptus cloud it can be Used as a Simple storage as a Service Solution

Open Stack

open Stack is a cloud operating System Consists of a Collection of Interaction Services that Control Computing Storage &

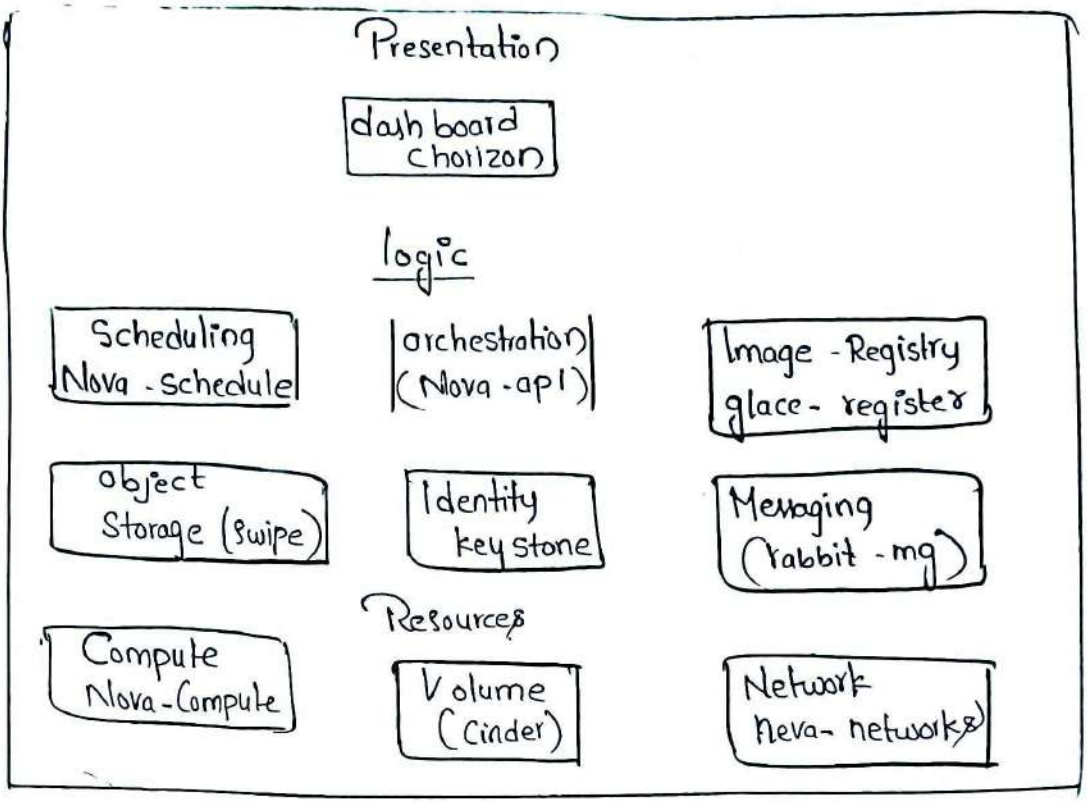


fig : Open Stack Architecture

1. The openstack Compute Services (Called Nova - Compute) manage Networks of Virtual machines running on nodes, providing Virtual services on demand
2. the network Service (Called Nova Networking) provides Connectivity between the interfaces of other open Stack Services
3. the Volume Service (cinder) manages Storage volumes for Virtual machines
4. the object storage Service (Swift) allow Users to store and retrieve files
5. the Identity Service (key store) provides authentication and authorization for other Services
6. the image registry (glauce) acts as a catalog and repository for virtual machine images
7. the open stack scheduler (nova - Scheduler) maps the nova - API Calls to the appropriate openstack Components
8. the Scheduler takes the Several Virtual machine request from the queue and determines where they should run
9. the Messaging Service (rabbit - mq) acts as a Central node for Message passing between daemons
10. Orchastration activities Such as running an instance are provided by a nova - api which accepts and responds end User Compute API calls

Unit - II : Hadoop and Python

* Apache Hadoop :

Apache Hadoop is an open source framework for distributed batch processing of big data MapReduce has been proposed as a parallel programming model Suitable for the cloud The Hadoop ecosystem consists of a number of projects as shown in below figure

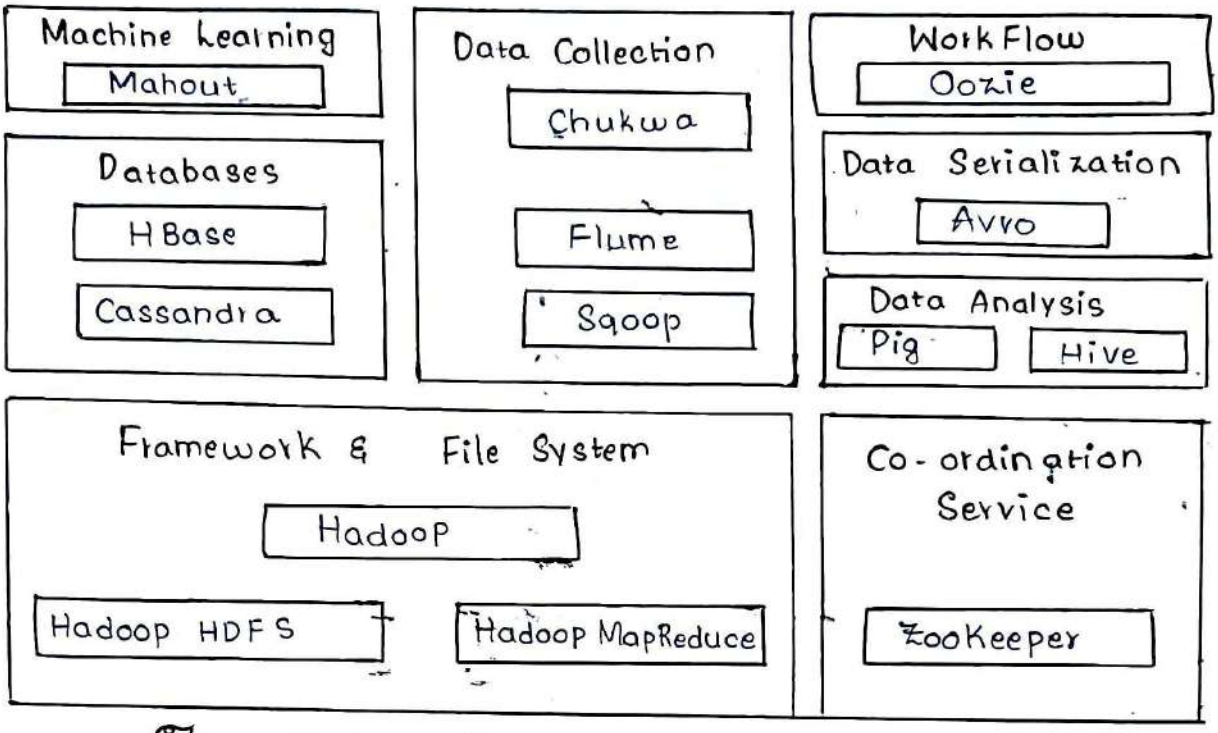


Fig. : Hadoop Ecosystem

These projects are described as follows :

1. Hadoop Common : Hadoop common consists of utilities & scripts for starting Hadoop, components & interfaces to access the file systems supported by Hadoop.
2. HDFS : HDFS runs on large clusters & provides high throughput access to data. It was built to store each file as a sequence of blocks all of which are of the same size expect the last block The blocks of each file are replicated on multiple machines in a cluster with a default replication factor of 3 to provide fault tolerance.
3. Hadoop MapReduce : MapReduce is a parallel programming model that allows distributing large scale data processing in a sequence of operations on datasets of key-value pairs
4. Hadoop YARN : A framework for job scheduling and cluster resource management.
5. HBase : HBase is a scalable, non-relational, distributed,

6. ZooKeeper : ZooKeeper is a high performance distributed co-ordination service for maintaining configuration information, naming, providing distributed synchronization & group services.

7. Pig : Pig is a data flow language & an execution environment for analyzing large datasets.

8. Hive : Hive is a distributed data warehouse infrastructure for Hadoop. Hive provides an SQL-like language called HiveQL that allows easy data summarization, adhoc queting, & analysis of large datasets stored in HDFS.

9. Chukwa : Chukwa is a data collection system for monitoring large distributed systems.

10. Mahout : Mahout is a scalable machine learning library for Hadoop. It provides a large collection of Machine learning algorithms, for clustering, classification & collaborative filteting which are implemented on top of Hadoop using MapReduce parallel processing programming model.

11. Cassandra : Cassandra is scalable multi-master database with no single points of failure. It is a NoSQL solution that provides a structured key-value store.

12. Avro : Avro is a data serialization system that provides rich data structures, a compact & fast binary data format, a container file to store the persistent data, cross-language RPC & simple integration with dynamic language.

13. Oozie : Oozie is a workflow scheduler system for managing Hadoop jobs such as MapReduce jobs, Pig, Hive, Sqoop, Distcp & System specific jobs.

14. Flume : Flume is a distributed, reliable & available service for collecting, analyzing & moving large amounts of data from applications to HDFS.

15. Sqoop : Sqoop is a tool that allows efficiently transferring bulk data between Hadoop & structured data stores such as relational databases.

* Hadoop MapReduce Job Execution :

- MapReduce is a parallel programming model.
- MapReduce job consists of two phases :
  - Map phase
  - Reduce phase
- In Map phase, data is read from a distributed filesystem & partitioned among a set of computing nodes in the cluster.

- The Map task process the input records independently & produce intermediate results as key-value pairs.
- The intermediate results are stored on the local disk of the node running the map task.
- When all the map tasks are completed, the Reduce phase begins in which the intermediate data with the same key is aggregated.
- An optional combine task can be used to perform data aggregation on the intermediate data of the same key for the output of the mapper before transferring the output to the Reduce task.

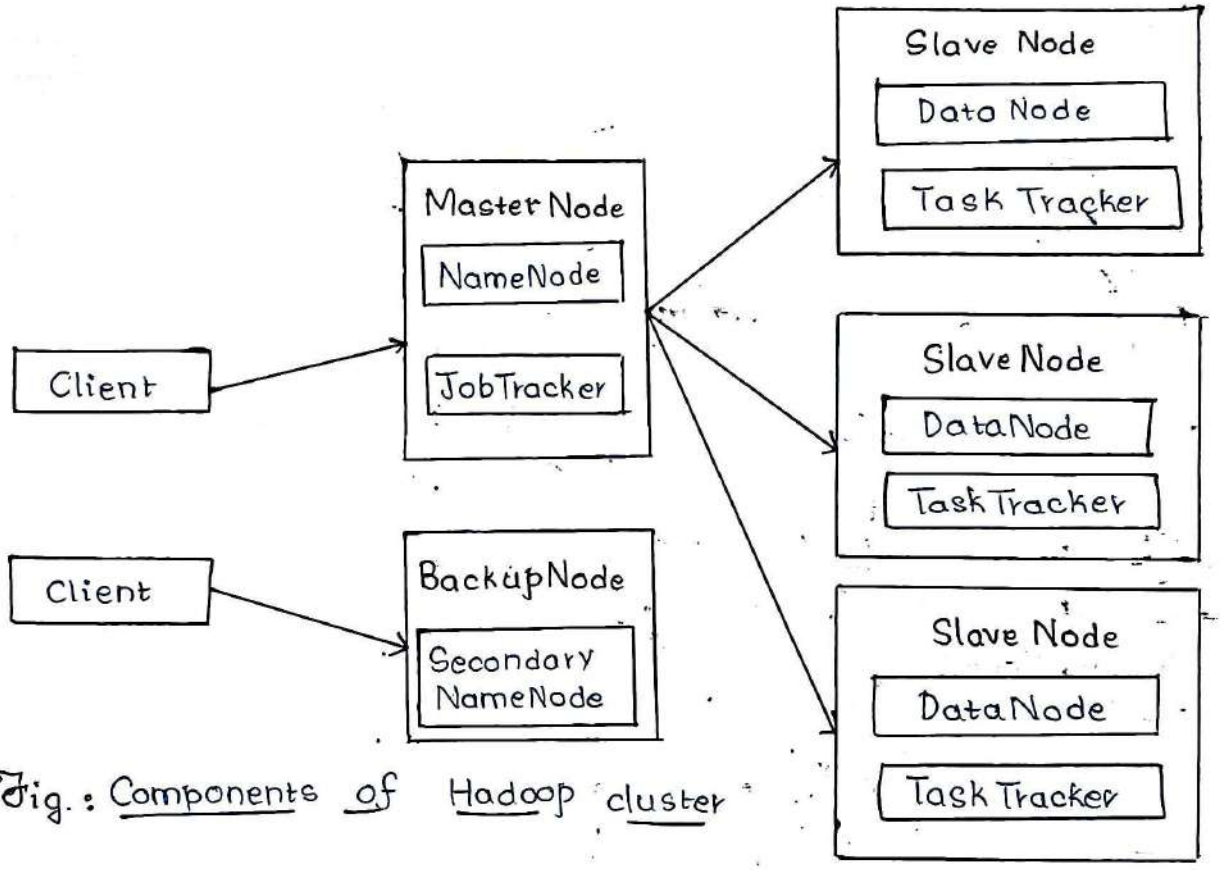


Fig.: Components of Hadoop cluster

- The above figure shows the components of a Hadoop cluster.
- A Hadoop cluster consists of a Master node, backup node & a no. of slave nodes.
- The Master node runs the name node & Job Tracker process.
- Slave nodes run the data node & Task Tracker components of Hadoop.
- The backup node runs the secondary namenodes process.
- The functions of the key processes of Hadoop are described as follows:

o Name Node:
- Name Node keeps the directory tree of all files in the

- Client applications talk to the NameNode whenever they wish to locate a file (or) when they want to add/copy/move/delete a file.
- The NameNode responds to the successful requests by returning a list of relevant datanode servers where the data lives.

◦ Secondary NameNode :

- The NameNode is a single point of failure for the HDFS cluster.
- When the NameNode goes down, the file system goes offline.
- An optional secondary NameNode which is hosted on a separate machine creates checkpoints of the name space.

◦ JobTracker :

- JobTracker is the service within Hadoop that distributes MapReduce tasks to a specific nodes in the cluster.

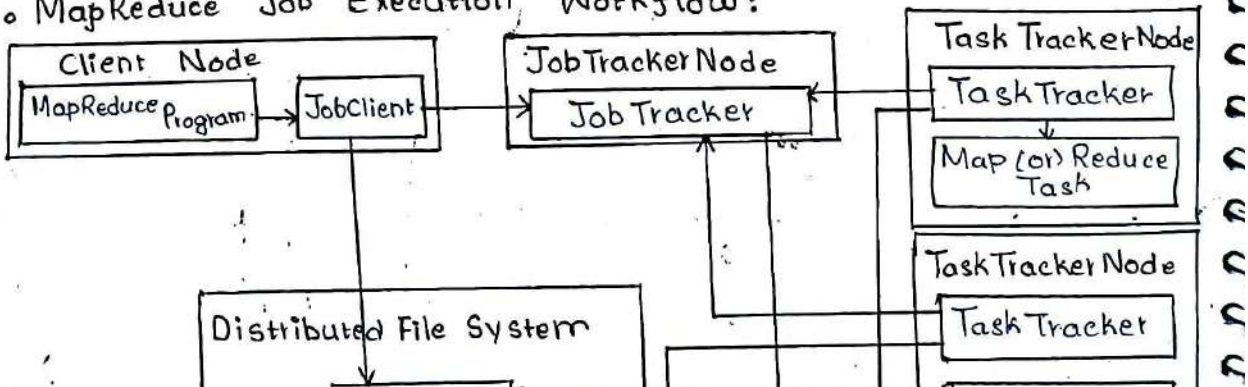
◦ TaskTracker :

- TaskTracker is a node in a Hadoop cluster that accepts Map, Reduce & Shuffle tasks from the JobTracker.
- Each TaskTracker has a defined no. of slots which indicate the no. of tasks that it can accept.
- If an empty slot is not found on the same node, then the JobTracker looks for an empty slot on a node in same rack.

◦ Data Node :

- DataNode stores data in an HDFS file system.
- A functional HDFS file system has more than one DataNode, with data replicated across them.
- Datanodes respond to requests from the NameNode for file system operations.
- Client applications can talk directly to a DataNode, once the NameNode has provided the location of data.
- Similarly, MapReduce operations assigned to TaskTracker instances near a DataNode, talk directly to the DataNode to access the files.

◦ MapReduce Job Execution Workflow :



- The above fig. shows MapReduce job execution workflow for first generation Hadoop MapReduce Framework MPI
- The job execution starts when the client applications submit jobs to the Job tracker.
- The JobTracker returns a JobIn to the client application.
- Job Tracker talks to the NameNode to determine the location of the data.
- JobTracker locates TaskTracker nodes with available slots (or) near the data.
- The TaskTracker sends out heartbeat msgs to JobTracker, every few minutes, to reassure the JobTracker that they are still alive.
- The JobTracker submits the work to the TaskTracker nodes when they poll for tasks.
- To choose a task for a Task Tracker, the jobtracker uses various scheduling algorithms.
- The default scheduling algorithm in Hadoop in FIFO.
- The TaskTracker nodes are monitored using heartbeat signals that are sent by TaskTrackers to JobTrackers.
- The TaskTracker creates a separate JVM process for each task, so that any task failure does not bring down the Task Tracker.
- When the process finishes, successfully (or) not, the TaskTracker notifies the Job Tracker.
- When a task fails the TaskTracker notifies the JobTracker & the JobTracker decides whether to resubmit job to some other TaskTracker (or) not.
- The JobTracker can blocklist a TaskTasker as unreliable if there are repeated task failures.
- When the job is completed, JobTracker updates its status.

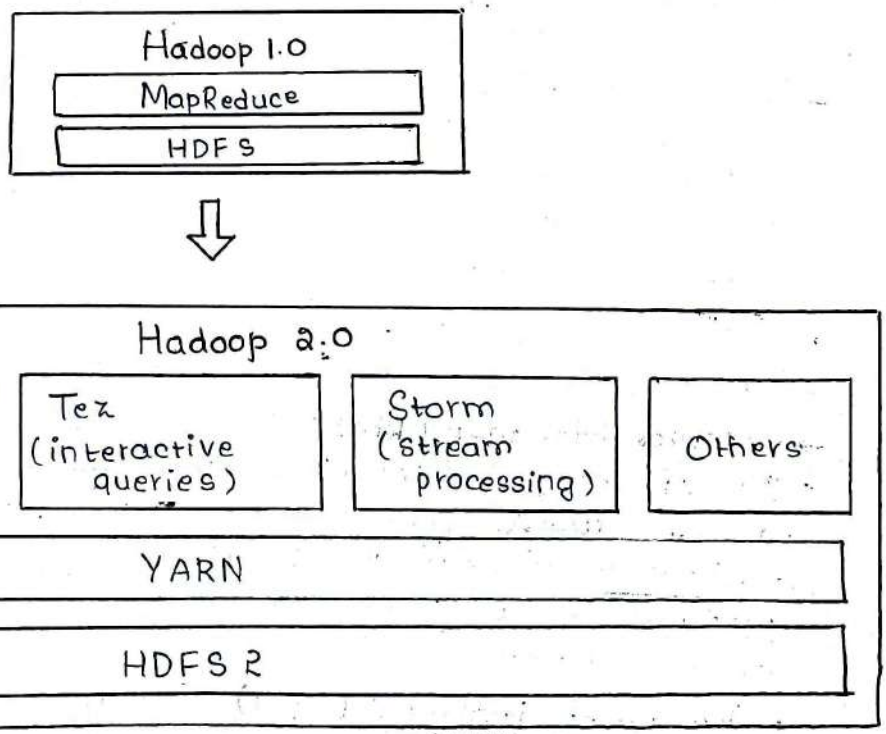


Fig.: Comparison of Hadoop 1.0 & Hadoop 2.0

- The new MapReduce architecture is called YARN (or) Map-Reduce 2.0 (MR2)
- In Hadoop 2.0 the original processing engine of Hadoop (MR) has been separated from the resource management as shown in figure.
- This makes YARN effectively an OS for Hadoop that supports different processing engines on a Hadoop cluster such as MapReduce for batch processing. Apache Tez for interactive queries, Apache Storm for streaming process.

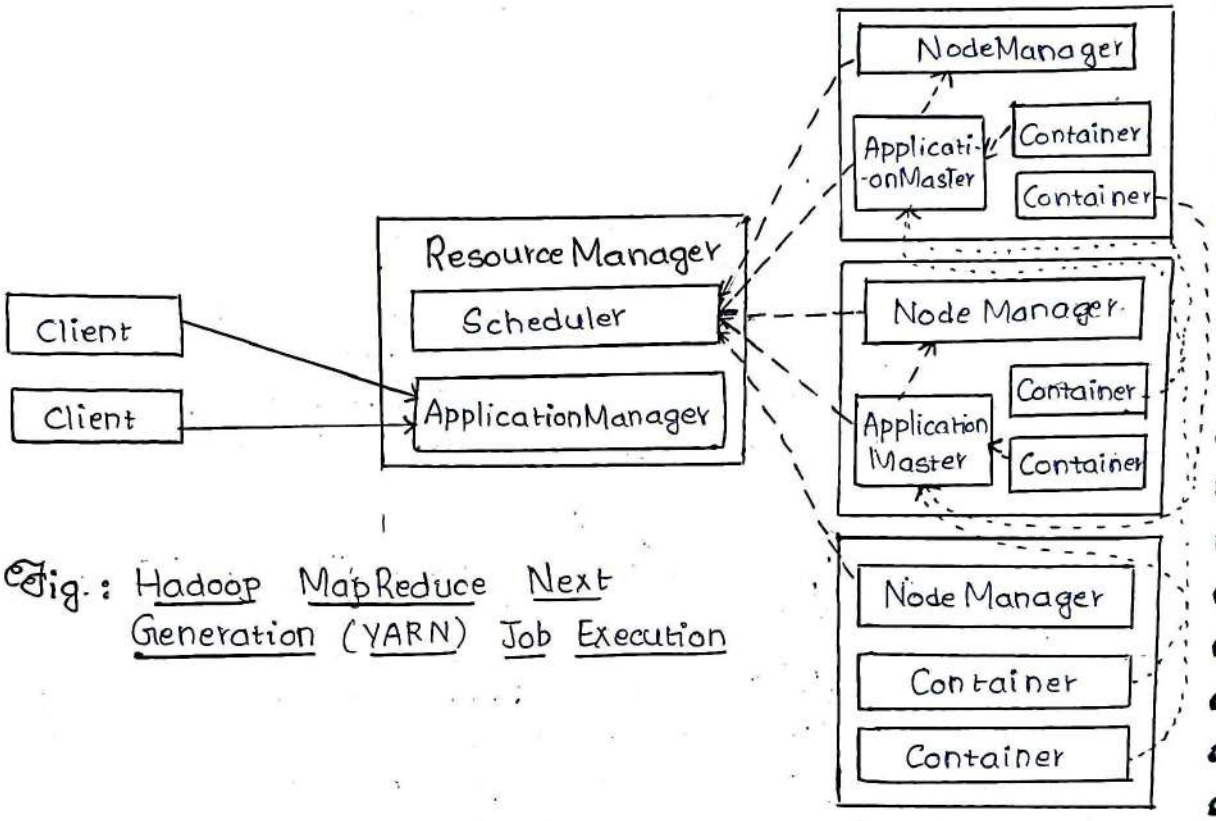


Fig.: Hadoop MapReduce Next Generation (YARN) Job Execution

- The above figure shows the MapReduce Job Execution Workflow for next generation Hadoop MapReduce framework (MR2).
- The next generation MapReduce architecture divides the 2 major functions of the Job Tracker –
- Resource Management & Job life-cycle management into separate components.
  - Resource Manager.
  - Application Master.
- The key components of YARN are as follows:

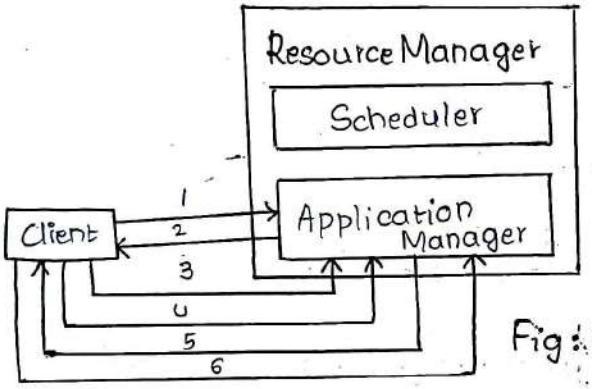
→ **Resource Manager (RM):** RM manages the global assignment of compute resources to applications. RM consists of two main services.
- Scheduler: Scheduler is a pluggable service that manages & enforces the resource scheduling policy in the cluster.
- Application Manager (AsM): It manages the running

restarting them on different nodes in case of failures.

* **Application Master (AM):** A per-application AM manages the application's life cycle. AM is responsible for negotiating resources from the RM & working with the NMs to execute & monitor the tasks.
* **Node Manager (NM):** A per-machine NM manages the user processes on that machine.
* **Containers:** Container is a bundle of resources allocated by RM. Each node has an NM that allows to create multiple containers based on the resource allocations made by the RM.

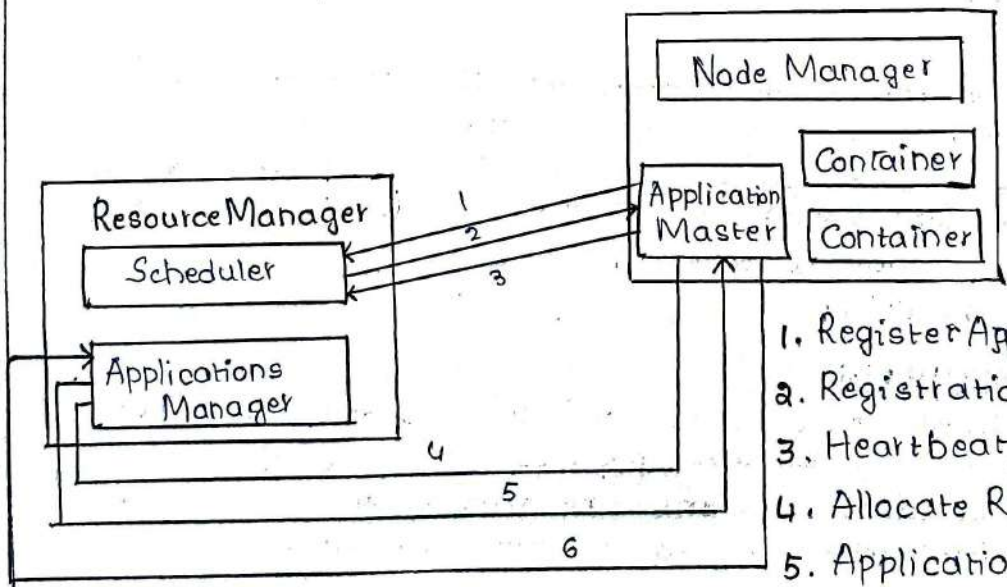
The above figure shows a YARN cluster with Resource Manager node and three Node Manager nodes. There are as many application masters running as there are applications (Jobs). Each application's AM manages the application tasks such as starting, monitoring & restarting tasks in case of failures.

To better understand the YARN job execution workflow analyze the interactions between the main components on YARN as shown in the following figure.



1. New Application Request
2. Response (Application ID)
3. Submit Application
4. Get Application Report
5. Application Report Response
6. Force kill application.

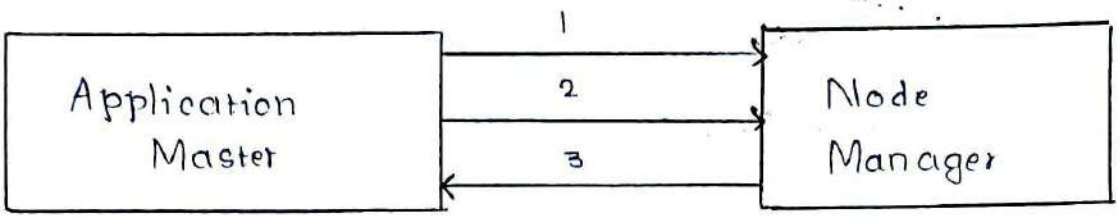
Fig: Client - Resource Manager Interaction



1. Register Application Master
2. Registration Response
3. Heartbeats
4. Allocate Request
5. Application Response
6. Finish Application

Fig.: Resource Manager - Application

Fig.: Application Master - Node Manager Instructions.



1. Start Container
2. Container Status Request
3. Container Status Response

* **Hadoop Schedulers:**

1. Hadoop Scheduler is a pluggable component that support different scheduling algorithms.
2. The default scheduler in Hadoop is FIFO.
3. In addition to this, two advanced schedulers are also available the fair scheduler, developed at Facebook, & the capacity scheduler, developed at Yahoo.
4. The pluggable scheduler - framework provides the flexibility to support a variety of workloads with varying priority & performance constraints.
5. Efficient job schedulers & job scheduling make Hadoop a multi-tasking system that can process multiple datasets for multiple jobs for multiple users simultaneously.

Hadoop scheduling algorithms are discussed as follows:

- **FIFO:**

→ FIFO is the default scheduler in Hadoop that maintains a work queue in which jobs are queued.
→ The scheduler pulls jobs in First In First Out manner (oldest Job first) for scheduling.
→ There is no concept of priority (or) size of job in FIFO scheduler.

- **Fair Scheduler:**

→ Fair Scheduler allocates resources evenly between multiple jobs & also provides capacity guarantees.
→ Fair Scheduler assigns resources to jobs such that each job gets an equal share of available resources on average overtime.
→ Unlike the FIFO Scheduler, which forms a queue of

→ Task slots that are free are assigned to the new jobs, so that each job gets roughly the same CPU time.
→ The fair scheduler maintains a set of pools into which jobs are placed. Each pool has a guaranteed capacity.
→ A configuration file is used for specifying the pools & the guaranteed capacities.
→ When there is a single job running, all the resources are assigned to that job. When there are multiple jobs in the pools, each pool gets atleast as many task slots guaranteed.
→ The fair scheduler keeps track of the compute time received by each job.
→ When multiple users are submitting users jobs, to ensure fairness, each user is assigned to a pool.
→ It is possible to limit the no. of running jobs per user (or) per pool by specifying the limits in the configuration file.

The following table lists the configurable properties of the fair scheduler. **Table**: List of configurable properties of Fair Scheduler.

Property Name	Details
Mapred.fairscheduler.allocation.file	Absolute path to an XML file which contains the allocations for each pool, as well as per pool & per-user limits on no. of running jobs. If this property is not provided, allocations are not used.
Mapred.fairscheduler.assignmultiple.	Allows the scheduler to assign a map task & a reduce task, on each heartbeat, which improves cluster throughput when there are many small tasks to run.
Mapred.fairscheduler.sizebasedweight	Take into account job sizes in calculating their weights for fair sharing.
Mapred.Fairscheduler.PoolnameProperty	Specify which jobconf property is used to determine the pool that a job belongs in.
Mapred.Fairscheduler.Weightadjuster	Specify a class to adjust the weight of running jobs.
Mapred.Fairscheduler.loadmanager	Specify a class that determines how many maps & reduces can run on a given TaskTracker.

∘ Capacity Scheduler :

1. The capacity scheduler has similar functionalities as Fair scheduler but adopts a different scheduling viewpoint.
2. In capacity scheduler we have multiple job queues for scheduling our tasks.
3. In capacity scheduler, you define a number of named queues each with a configurable number of map & reduce slots for performing job operation.
4. Each job queue has its own slots to perform its tasks.
5. In case we have task to perform in only one queue then the tasks to perform can access the slot of other queues also as they are free to use.
6. When the new task entets to some other queue then jobs in running in its own slots of the cluster are replaced with its own job.
7. It also provides a level of abstraction to know which occupant is utilizing the more cluster resource (or) slots, so that the single user (or) the application doesn't take unnecessary slots in the cluster.
8. The capacity scheduler mainly contains 3 types of the queue that are root, parent & leaf which are used to represent cluster, organization, (or) any subgroup, application submission respectively.
   Following table lists the configurable properties of the capacity scheduler.

Property Name	Details
Mapred. Capacity-scheduler. queue. <queue-name>. guaranteed_capacity	Percentage of number of slots in the cluster that are guaranteed to be available for job in this queue. The sum of guaranteed capacities for all queues should be less than (or) equal to 100.
Mapred. capacity-scheduler. queue <queue-name>. reclaim-time-limit	The amount of time, in seconds, before which resources distributed to other queues will be reclaimed.
Mapred. Capacity-Scheduler. queue. <queue-name>. minimum-user-	Each queue enforces a limit on the percentage of resources allocated to a user at any given time, if there is competition for them. This

Mapred. Capacity-Scheduler. queue. <queue-name>. supports Priority	If true, priorities of jobs will be taken into account in scheduling decisions.

* **Hadoop Cluster Setup:**

1. Hadoop framework is written in Java & has been designed to work with commodity hardware.
2. The Hadoop's filesystem HDFS is highly fault-tolerant.
3. The suitable operating system to host Hadoop is Linux, however, it can be set up on Windows operating systems with Cygwin environment.
4. The below figure shows the multi-node Hadoop cluster configuration.
5. The Hadoop cluster comprises of one master node that runs the NameNode & JobTracker & two slave nodes that run the TaskTracker & DataNode.
6. The hardware used for the Hadoop cluster example described consists of three Amazon EC2 instances running Ubuntu Linux.
7. The steps involved in setting up a Hadoop cluster are:

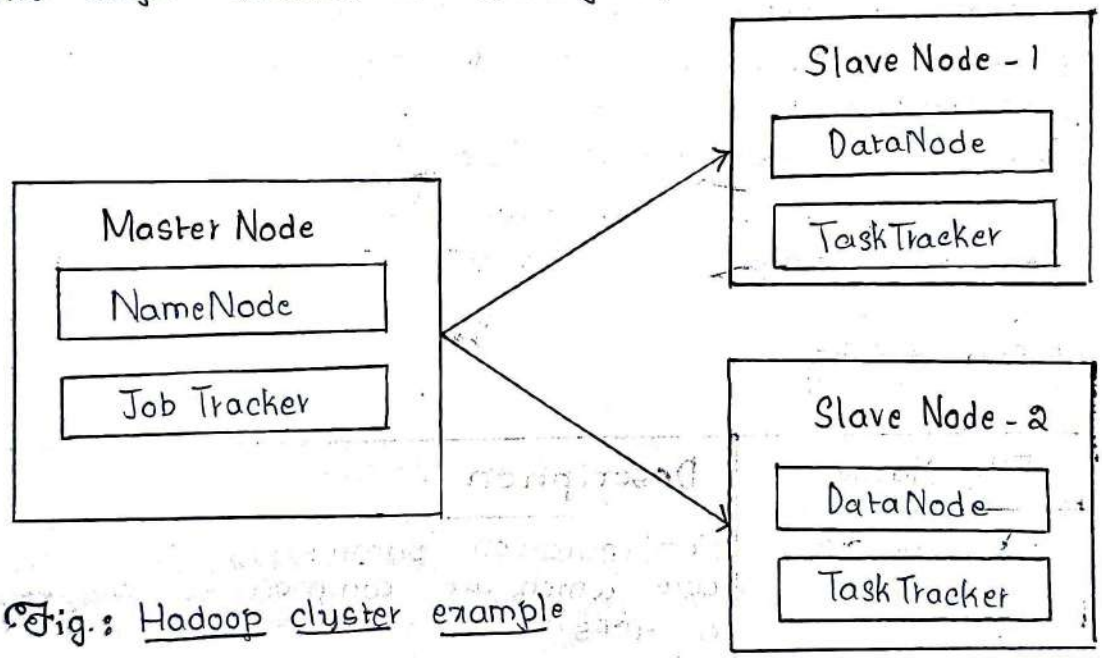


Fig.: Hadoop cluster example

Install Java

Hadoop requires Java 6 (or) latest version.

Install Hadoop

To setup a Hadoop cluster, the Hadoop setup tarball is downloaded & unpacked on all the nodes.

Commands to install Java:

```
# Verify if Java is installed
$ java -version
```

```
$ sudo add-apt-repository ppa:ferramroberto/java
$ sudo apt-get update
$ sudo apt-get install sun-java6-jdk
$ sudo update-java-alternatives -s java-6-sun
```

Install & configure Hadoop commands.

```
$ wget http://apache.techartifact.com/mirror/hadoop/common/hadoop-1.0.4/hadoop-1.0.4.tar.gz
Star xzf hadoop-1.0.4 for .gz
# Change hostname of node
# Change hostname master
# sudo hostname slave1
# sudo hostname slave2
# Modify /etc/hosts file & add private IPs of Master and Slave nodes:
$ sudo vim/etc/hosts
# <private_IP_master> master
# <private_IP_slave1> slave1
# <private_IP_slave2> slave2
$ ssh-keygen -t rsa -f /.ssh/id_rsa
$ sudo cat/.ssh/id_rsa.pub >> /.ssh/authorized_keys
# Open authorized keys file & copy authorized keys of each node
$ sudo vim/.ssh/authorized_keys
# Save host key configuration fingerprints by connecting to every node using SSH
# ssh master
# ssh slave1
# ssh slave2
```

File Name	Description
core-site.xml	Configuration parameters for Hadoop core which are common to MapReduce & HDFS.
mapred-site.xml	Configuration parameters for MapReduce daemons - JobTracker & TaskTracker.
hdfs-site.xml	Configuration parameters for HDFS daemons - NameNode & Secondary NameNode & DataNode.
hadoop-env.sh	Environment variables for Hadoop daemons.
masters	list of nodes run on Secondary NameNode
slaves	list of nodes run TaskTracker & DataNode
log4j.properties	Logging properties for Hadoop daemons

Networking

- Configure the network such that all the nodes can connect to each other over the network.
- To make addressing of nodes simple, assign simple host names to nodes.
- Hadoop control scripts use SSH for cluster-wide operations such as starting & stopping NameNode, DataNode, JobTracker, TaskTracker & other daemons on the nodes in the cluster.
- For the control scripts to work, all the nodes in the cluster must be able to connect to each other via a password-less SSH login.
- To enable this, public/private RSA key pair is generated on each node.
- The private key is stored in the file /.ssh/id_rsa & public key is stored in the file /.ssh/id_rsa.pub.
- The public SSH key of each node is copied to the /.ssh/authorized_keys file of every other node.
- The final step to setup the networking is to save host key fingerprints of each node to the known_hosts file of every other node.

Configure Hadoop

With the Hadoop setup package unpacked on all nodes & networking of nodes setup, the next step is to configure the Hadoop cluster.

→ Sample configuration – core-site.xml

```
<? xml version = "1.0" ?>
<configuration>
<property>
<name> fs.default.name </name>
<value> hdfs://master:54310 </value>
</property>
</configuration>
```

→ Sample configuration hdfs-site.xml

```
<? xml version = "1.0" ?>
<configuration>
<property>
<name> dfs.replication </name>
<value> 2 </value>
</property>
</configuration>
```

→ Sample configuration mapred-site.xml

```
<? xml version = "1.0" ?>
<configuration>
```

→ Sample masters and slave files

```
$cd hadoop/conf
# Open the masters file and add hostname of masternode
 $vim masters
# master
# Open the masters file and add hostname of slavenodes
$vim slaves
# slave1
# slave2
```

→ Sample: Starting & stopping Hadoop cluster

```
$cd hadoop-1.0.4
# Format NameNode
$bin/hadoop namenode -format
# Start HDFS daemons
$bin/start-dfs.sh
# Start MapReduce daemons
$bin/start-mapred.sh
# check status of daemons
$jps
# Stopping Hadoop cluster
# bin/stop-mapred.sh
$bin/stop-dfs.sh
```

Starting and Stopping Hadoop Cluster

- If Hadoop cluster is correctly installed, configured & started, status of Hadoop daemons can be viewed using the administration web-pages for daemons.
- Hadoop publishes the status of HDFS & MapReduce jobs to an internally running web server on master node of Hadoop cluster.

← → C

NameNode: 'master: 54310'
Started:
Version:
Browse the filesystem
Name Node Logs

Name Node Storage:

Storage Directory	Type	State
/tmp/hadoop-ubuntu/name	IMAGE_AND_EDITS	Active

master Hadoop Map/Reduce Administration

State :

Started:

Version:

Cluster Summary (Heap Size is 25.12 MB/966.69 MB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Reserved Reduce Slots	Map Task Capacity
0	0	0	2	0	0	4

Running Jobs

none

Retired Jobs

none

This is Apache Hadoop release 1.0.4

Fig. 4.11: Hadoop MapReduce administration

NameNode - http://<NameNode Host Name>:50070/

JobTracker - http://<JobTracker Host Name>:50030/

NameNode 'master:54310'

Started :

Version :

Browse the file system

Namenode Logs

Go back to DFS home

Live Datanodes : 2

Node	Last contact	Admin State	Configured Capacity(GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Remaining (%)	Blocks
slave1	2	In Service	7.87	0	1.65	6.23	0	79.09	1
slave2	2	In Service	7.87	0	1.65	6.23	0	79.09	1

fig: Hadoop HDFS status page showing live data nodes

master Hadoop Machine List

Active Task Trackers

Name	Host	Running Tasks	Max Map Tasks	Max Reduce Tasks	Failures	Node Health Status
tracker_slave1_localhost/	slave1	0	2	2	0	N/A
tracker_slave2_localhost/	slave2	0	2	2	0	N/A

* Reference Architecture for Cloud Applications:

- Multi-tier cloud applications can have various deployment architecture alternatives.
- Choosing the right deployment architecture is important to ensure that the application meets the specified performance requirements.

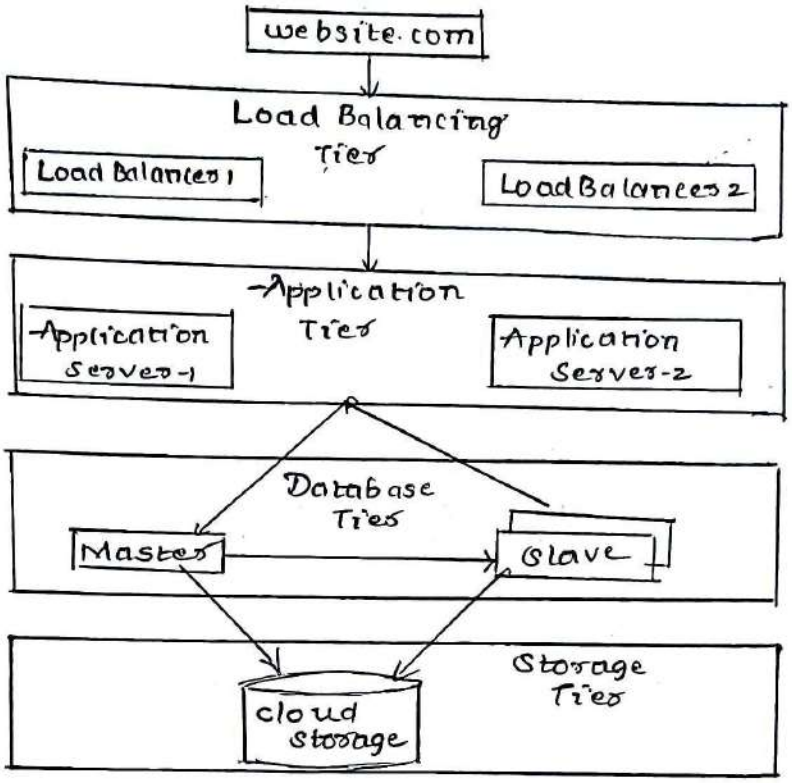


Fig: Typical deployment architecture for e-commerce, Business-to-Business, Banking & financial applications.

Load Balancing Tier: The first tier is the load balancing tier.

- Load Balancing Tier consists of one (or) more load balancers.
- It is recommended to have atleast two load balancer instances to avoid the single point of failure.
- It is also recommended to provision the load balancer instances in separate availability zones of the cloud service provider to improve reliability & availability.

Application Tier:

- The second tier is the application layer that consists of one (or) more application servers to configure auto-scaling.
- Auto scaling can be triggered when the recorded values for any of the specified metrics such as CPU usage, Memory usage etc goes above defined threshold.
- The minimum & maximum size of the application

servers running at all times to avoid a single point of failures.

- When an auto-scaling event occurs, a new instance is launched.
- In autoscaling options, the threshold for scaling down are also specified.

Database Tier :

- The third tier is the database tier which includes a master database instance & multiple slave instance.
- The master node servers all the write requests & read requests are served from slave nodes.
- This improves the throughput for the database tier since most applications have a higher no. of read requests than write requests.
- Multiple slave nodes also serve as a backup for the master node.
- For both master & slave nodes, it is highly recommended to use a disk subsystem for storage & not the instance-attached store to ensure the reliability & availability because in the event of failures if the instance-attached storage is used for the database, all data will be lost.
- Whereas, in the case of separate disk volumes, it is possible to restore the database.

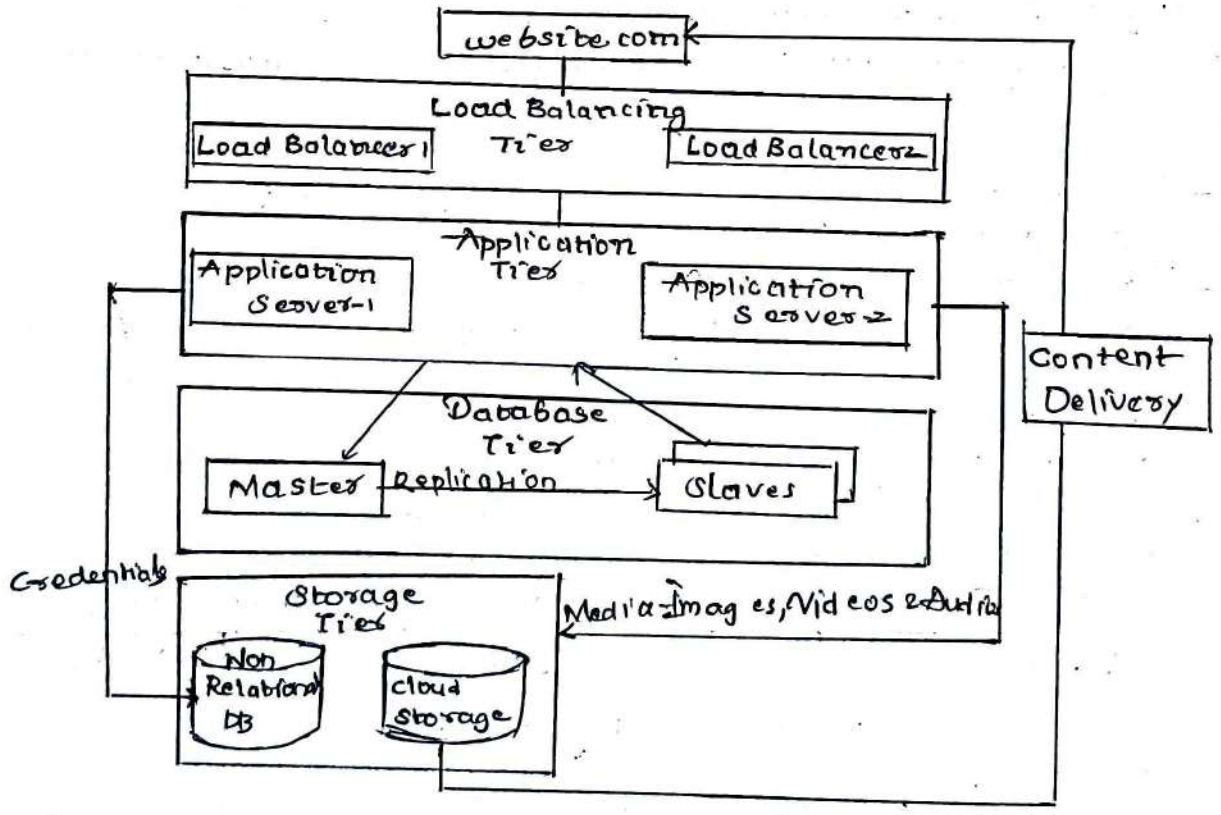


fig: Typical deployment architecture for content delivery applications such as online photo albums, video webcasting etc.

- Both relational & non-relational data stores are shown in the figure, in the deployment.
- A content delivery network CDN which consists of a global network of edge locations used for media delivery.
- CDN is used to speed up the delivery of static content such as images & videos.

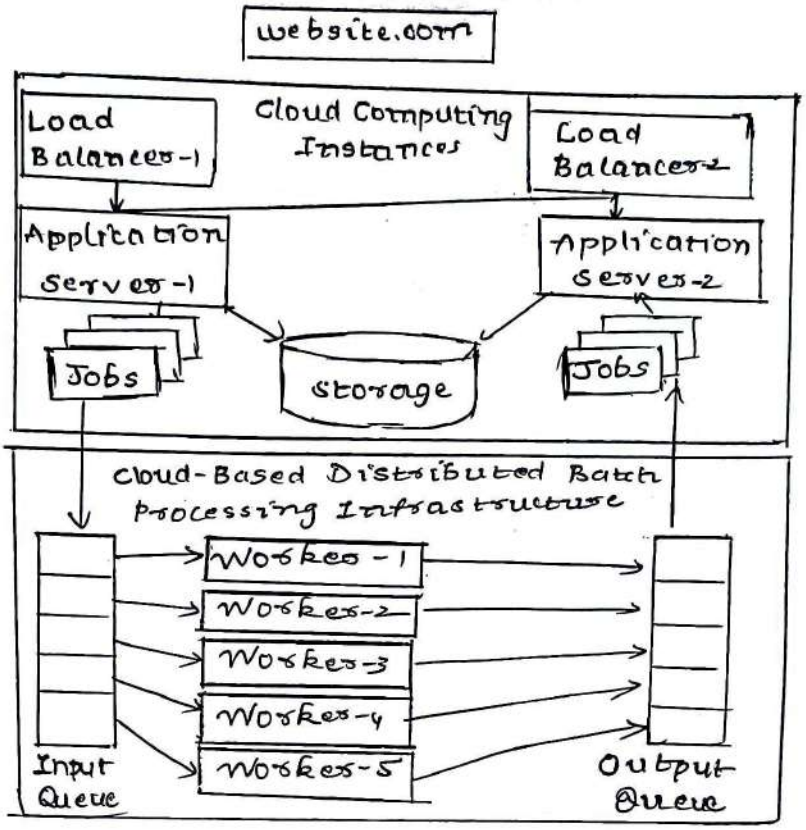


fig: Typical deployment architecture for compute intensive applications for Data Analytics, Media Transloding etc.

- The above figure shows web, application, storage, compu-ting / analytics & database tiers.
- The analytics tier consists of cloud-based distributed batch processing framework such as Hadoop suitable for analyzing big data.
- Data analysis jobs such as MapReduce jobs are submitted to the analytics tier from the application servers.
- The jobs are queued for execution & after completion the analyzed data is presented from application servers.

Table: Examples of popular cloud development tools.

Java	PHP	.NET	Python
Apache Tomcat Oracle Weblogic GlassFish IBM WebSphere JBoss	ZendServer Quercus	Internet Information Services (IIS) Web Server Windows Server AppFabric	Django Gunicorn Mod_Python Mod_wsgi

Cloud Application Design Methodologies :

Service Oriented Architecture (SOA) :

- Service Oriented Architecture (SOA) is a well established architectural approach for designing & developing the applications in the form of services that can be stored & reused.
- SOA is a collection of discrete software services that form a part of an application & collectively provide the functionality of an application.
- SOA services are developed as loosely coupled modules. The services communicate with each other by passing messages.
- Services are described using web service description language (WSDL).
- WSDL is an XML-based WSDL that is used to create service descriptions containing information on the functions performed by a service & the inputs & outputs of the service.

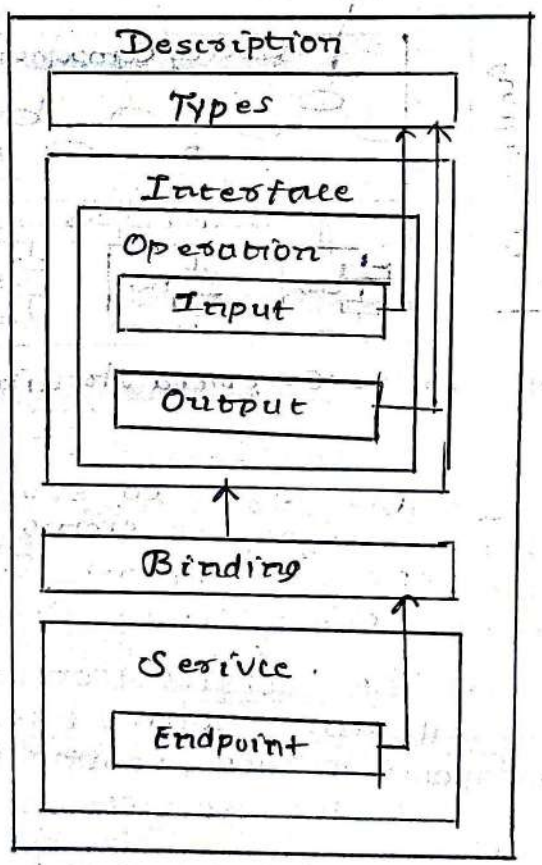


fig: WSDL concepts for representation of web services

- Service: It describes a discrete system function that is exposed as a web service.
- Endpoint: It is the address of the web service.
- Binding: It specifies the interface & transport protocol.
- Interface: It defines a web service & operations that can be performed by the service & inputs & outputs.
- Operation: It defines how the message is decoded &

* Soft services communicate using the simple object oriented protocol (SOAP).
- It is a protocol that allows exchange of structured information between web services.
- When WSDL is combination with SOAP is used to provide web services over the internet.
- SOA allows reuse of services for multiple applications as it is designed to perform a small function.

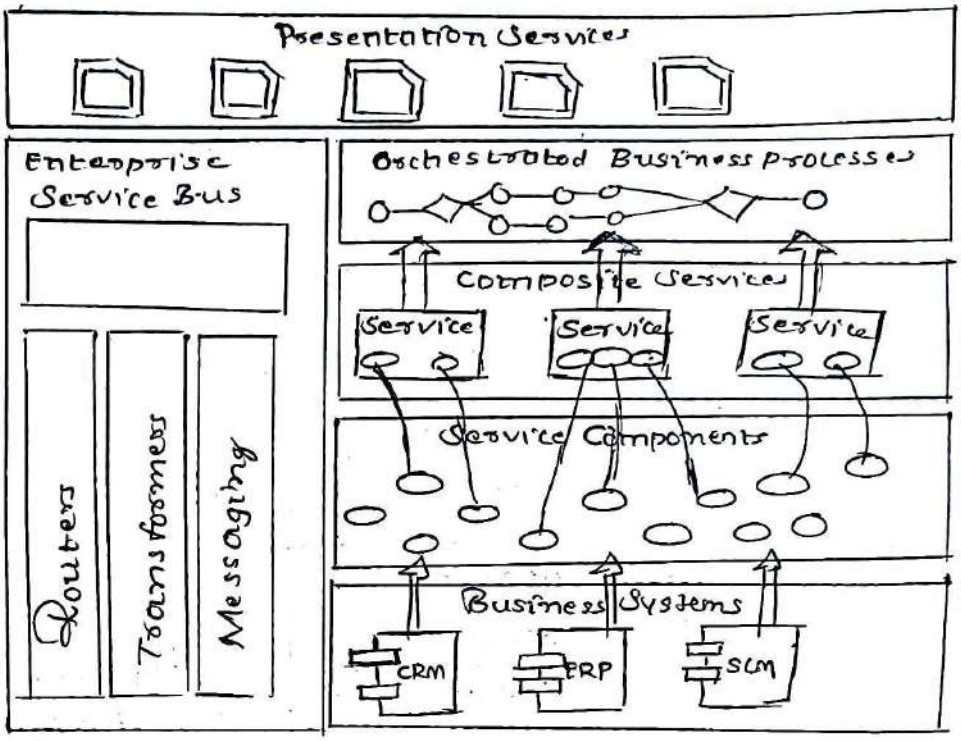


fig. Layers of Service Oriented Architecture.

The above figure shows the layers of SOA including:
→ Business Systems - This layer consists of custom built applications & legacy systems such as Enterprise Resource Manager (ERP), Customer Relationship Management (CRM), Supply Chain Management (SCM) etc.
→ Service Components - The service components allow the above layers to interact with business systems. The service components are responsible for realizing the functionality of the services exposed.
→ Composite services - It can be used to create enterpr. -ise scale components (or) Business - unit specific components.
→ Orchestrated Business Processes : Composite services can be orchestrated to create higher level business processes. In this layer the compositions and orchestrations of the compute services are

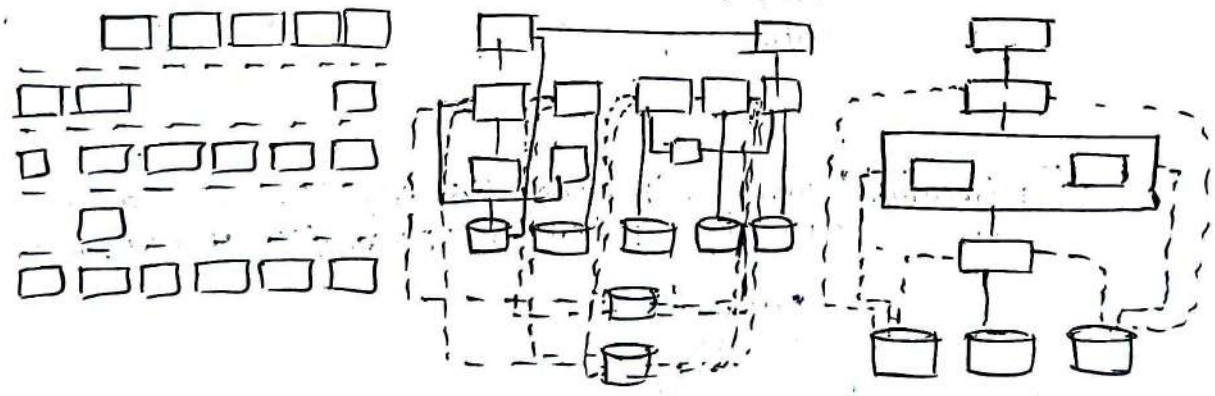
→ Presentation Services - It includes user interfaces that exposes the services & the orchestrated business processes to the users.

→ Enterprise Service Bus - It integrates the services through adapters, routing, transformation & messaging mechanisms.

∘ Cloud Component Model (CCM):

- Cloud Component Model is an application design methodologie that provides a flexible way of creating cloud applications in rapid, convenient & platform independent manner.
- CCM is an architectural approach for cloud applications that is not tied to any specific programming language (or) cloud platform.
- Cloud applications designed with CCM approach have better portability & interoperability.
- CCM based applications have better scalability by separating application components & providing asynchron--ous communication mechanisms.
- CCM makes maintainability of cloud applications easier the functionality of individual components can be improved (or) upgraded.
- CCM approach provides cost benefits for cloud applications.
- Cost benefits come by scaling cloud resources up only for those components which require additional computing capacity.

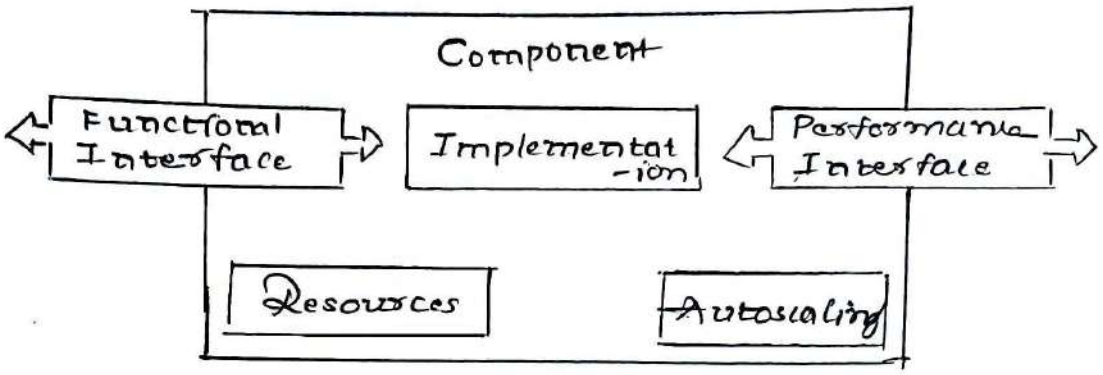
Component Design	Architecture Design	Deployment Design
Define cloud component model for application	Define Interactions between application components	Assign application components to cloud resources



- The above figure shows the steps involved in

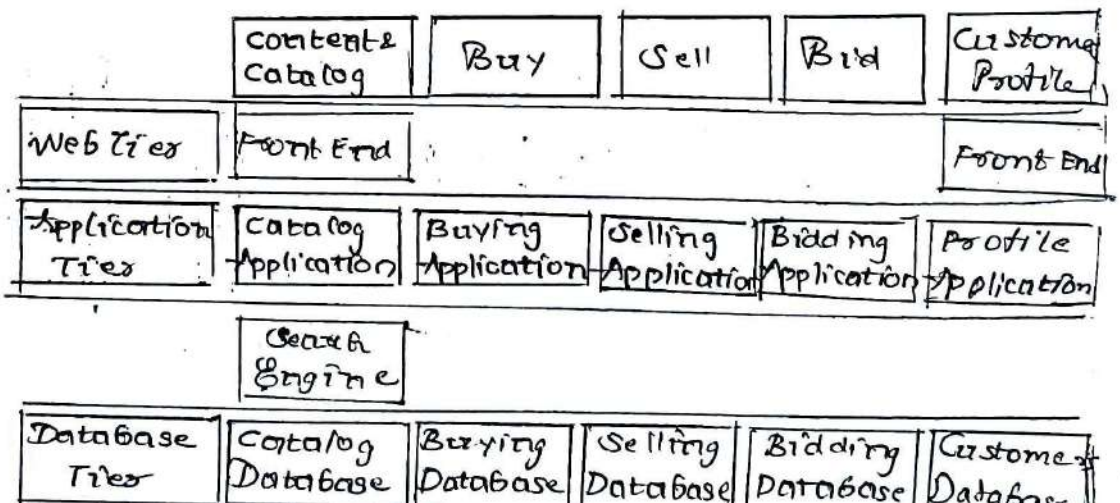
→ Component Design:

- In the first step, a cloud component model is created for the application based on comprehensive analysis of the applications functions & building blocks.
- CCM allows identifying the building blocks of a cloud application which are classified based on the functions performed & type of cloud resources required.
- Each building block performs a set of actions to produce the desired outputs for other components.



fig(b): Architecture of a CCM Component

- Each component takes specific inputs, performs a pre-defined set of actions & produces desired outputs.
- Components report their performance to a performance database through a performance interface.
- Components have no. of resources such as webpages, images, documents, database tables etc.
- Auto-scaling performance constraints & conditions can be specified for each component.
- Component-based approach is applicable to both web ~~pages~~ based applications & mobile applications.
- Below figure shows a CCM component map for an e-commerce applications.



→ Architecture Design :

- The second step in the CCM design methodology is architecture design.
- In this step, interactions between the application components are defined as shown in the below figure.

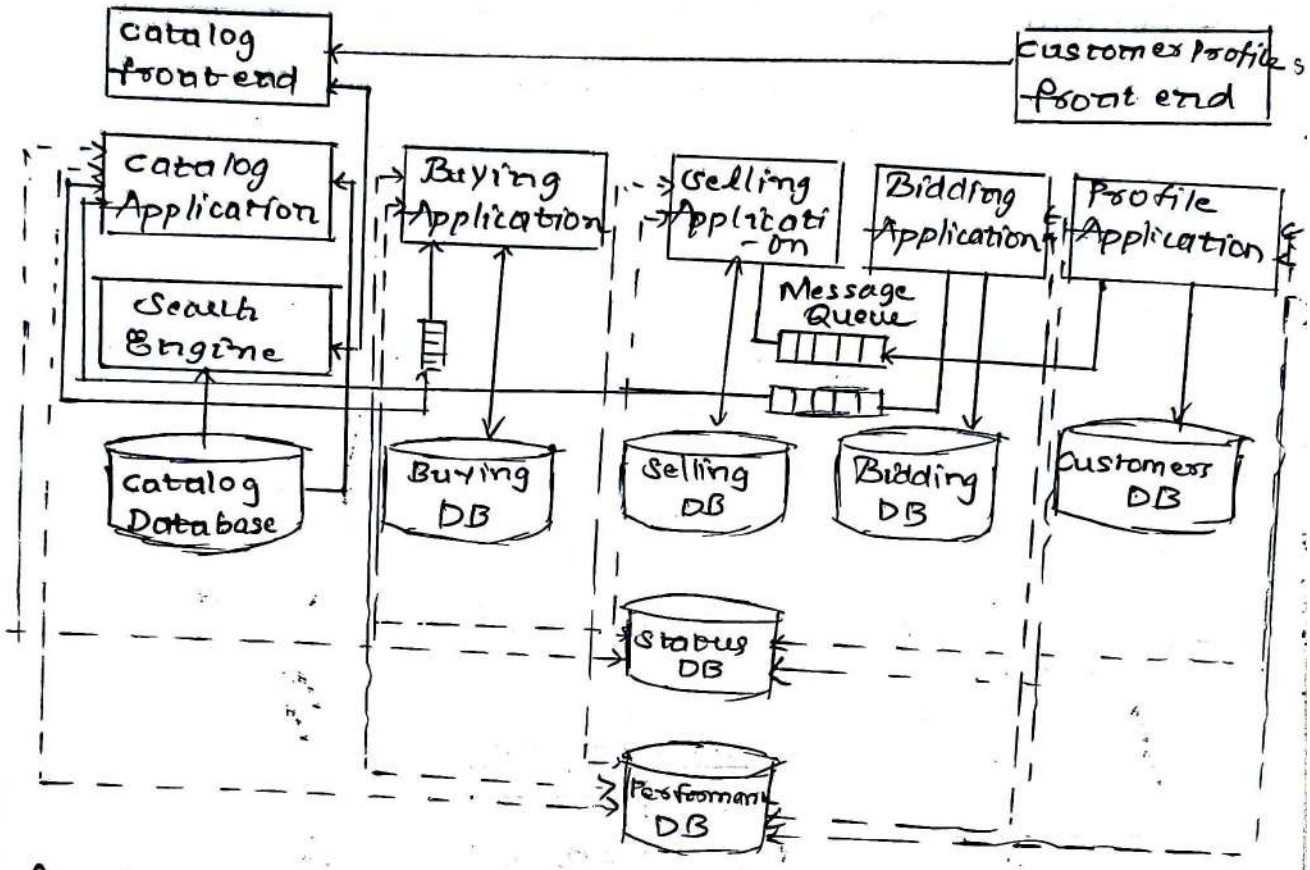


fig: Web interaction diagram for an e-commerce application

CCM component have the following characteristics :

Loose Coupling: Components in CCM are loosely coupled. Loose coupling of components relies on the use of REST communication protocol that allows component developed in different programming languages to communicate with each other.

Asynchronous Communication :

- Loosely coupled components communicate asynchronously through message based communication.
- Allowing asynchronous communication between components, it adds capacity by adding additional servers when the application load increases.
- Asynchronous communication is made possible by using messaging queues.
- The benefit of messaging queues is that the

Stateless Design:

- Components in the cloud component model are stateless.
- By storing session state outside of the component in a database, stateless component design enables the distribution & horizontal scaling

→ Deployment Design:

- The third step in CCM design methodology is deployment design
- In this step, application components are mapped to a specific cloud resources such as web servers, application servers, database servers, etc.
- The application components are designed to be loosely coupled & stateless with asynchronous communication, components can be deployed independently of each other.
- This approach makes it easy to migrate application components from one cloud to the other.
- With this flexibility in application design & deployment, the application developers ensure that the applications meet the performance & cost requirements with changing contexts.
- Following figure shows the deployment design for an e-commerce application.

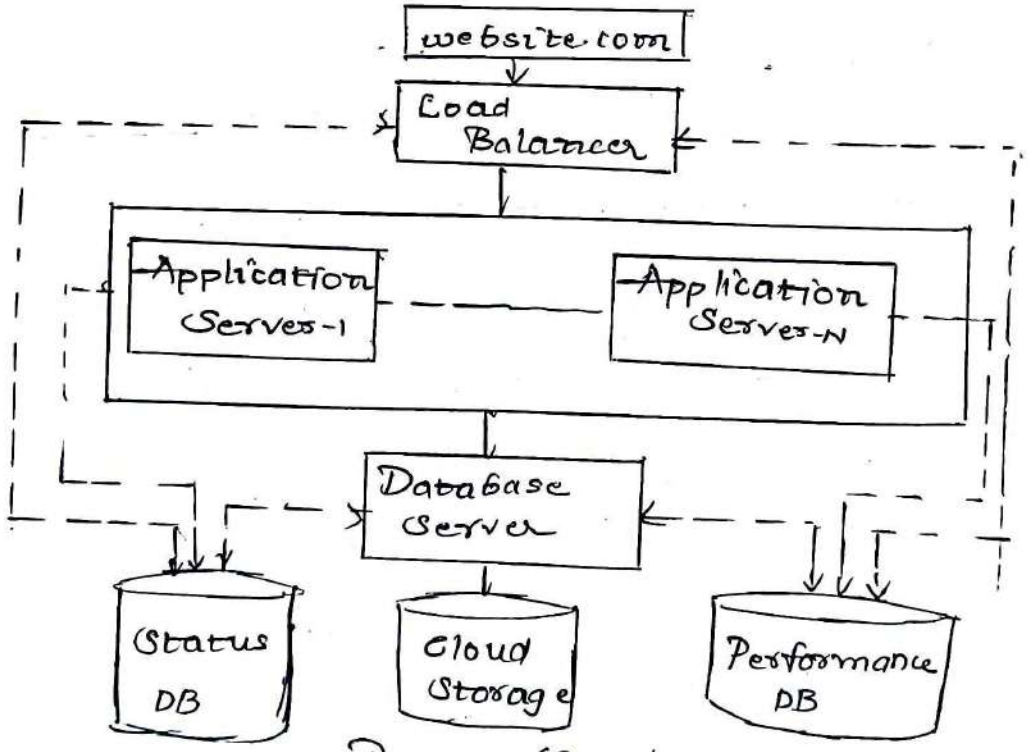


Fig: Multi-tier cloud deployment for an E-commerce Application

- Following table lists similarities & differences between SOA and CCM.

Similarities:

	SOA	CCM
Standardization & Re-use	SOA advocates principles of reuse & well defined relationship between service provider and service consumer.	CCM is based on reusable components which can be used by multiple cloud applications.
Loose Coupling	SOA is based on loosely coupled services that minimize dependencies.	CCM is based on loosely coupled components that communicate asynchronously.
Statelessness	SOA services minimize resource consumption by deferring the management of a state information.	CCM components are stateless. State is stored outside of the components.

Differences:

	SOA	CCM
End points	SOA services have small & well-defined set of endpoints through which many types of data can pass.	CCM components have very large no. of endpoints. There is an endpoint for each resource in a component, identified by a URI.
Messaging	SOA uses a messaging layer above HTTP by using SOAP which provided constrai-nts to developers.	CCM components use HTTP and REST for messaging.
Security	Uses WS-Security, SAML and other standards for security.	CCM Components use HTTPS for security.
Interfacing	SOA uses XML for interfacing.	CCM allows resources in components represent different formats for interfacing.
Consumption	Consuming traditional SOA services in a browser is cumbersome.	CCM components & the underlying component resources are exposed as XML, JSON over

Iaas, Paas & Saas Services for Cloud Applications :

- Cloud Service provides such as Amazon, Google, Microsoft, etc provide diverse infrastructure (Iaas), platform (Paas) & software (Saas) services that the developers can use for developing & deploying applications in cloud computing environments.
- Cloud service providers make cloud resources available to the users with cloud APIs via a web-based interfaces.
- There are numerous cloud service providers with their own cloud APIs.
- For example, Salesforce Paas (Force.com) allows developers to create applications using Apex & VisualForce.
- Similarly, applications designed for Google App Engine (GAE) platform must implement GAE Specific interfaces and have a GAE Specific deployment descriptor.
- CloudFoundry is another open Paas offering that supports a variety of clouds, frameworks & application services.
- Open Paas offerings provide greater flexibility than the vendor-specific Paas offerings as they support more programming languages & frameworks & provide a greater choice of clouds for deployment.

Model View Controller :

Model View Controller (MVC) is a popular software design pattern for web applications.

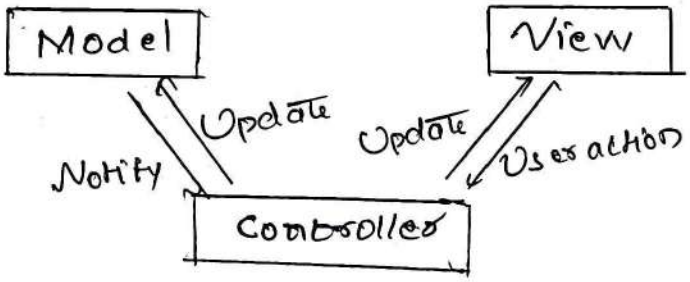


fig. Model View Controller

The MVC pattern consists of three parts :

→ Model :

- Model manages the data & behavior of applications.
- Model processes events sent by the controller.
- Model has no information about the views & controller.
- Model responds to the requests for information about its state from the view.
- It responds to the instructions to change stat

→ View:

- View prepares an interface which is shown to user.
- Users interact with the application through views.
- Views present the information that the model (or) controller tell the view to present to the user & also handles user requests & sends them to controller.

→ Controller:

- Controller glues the model to the view.
- Controller processes user requests & updates the model when the user manipulates the view.
- Controller also updates the view when the model changes.

* MVC separates the application logic, the data, and the user interface.
* The benefit of using MVC is that it improves the maintainability of the application & allows reuse of code.
* Applications built with MVC architecture can be updated easily due to the separation of the model from view
* In MVC, both the view & controller depends on the model, but the model doesnot depends on either.
* This allows the model to be developed & tested independently.
* In traditional applications the view is generally tightly coupled with the model. Since the views are likely to change more frequently than model.
* In MVC, the views can be changed without affecting the model.

RESTful Web Services

- Representational State Transfer (REST) contains a set of architectural principles by which you can design web services & web APIs that focus on a system's resources & how resource states are addressed and transferred.
- The REST architectural consists apply to the components, connectors, and data elements, within a distributed hypermedia system.
- The REST architectural system constraints are as follows:

- Client - Server:

- The principle behind the client - server constraint is the separation of concerns.
- For example, clients should not be concerned with the

- Separation of client & server allows to develop and update independently.

- Stateless :
- Each request from client to server must contain all the necessary information to understand the request, & cannot take advantage of any stored context of the server.
- The session state is kept entirely on the client.

- Cacheable :
Cacheable constraint requires that the data within a response to a request be implicitly (or) explicitly labelled as cacheable (or) non-cacheable.

- Layered System :
- Layered System constraint the behavior of components such that each component cannot see beyond the immediate layer with which they are interacting.
- System scalability can be improved by allowing intermediaries to respond to requests instead of the end server.

- Uniform Interface :
- Uniform interface constraint requires that the method of communication between a client & server must be uniform.

- Code on Demand :
- Servers can provide executable code (or) scripts for clients to execute in their context. This is the optional constraint.
- A RESTful web service is a web API implemented using HTTP & REST principles.
- RESTful web service is a collection of resources which are represented by URIs.

HTTP method	Resource Type	Action	Example
GET	Collection URI	List all the resources in a collection.	http://example.com/api/tasks/ (list all the task)
GET	Element URI	Get information about resource	http://example.com/api/tasks/1/ (get information on task-1)
POST	Collection URI	Create a new resource	http://example.com/

POST	Element URI	Generally not used.	
PUT	Collection URI	Replace the entire collection with another collection.	http://example.com/api/tasks
PUT	Element URI	Update a resource	http://example.com/api/tasks/1/
Delete	Collection URI	Delete the entire collection	http://example.com/api/tasks/
Delete	Element URI	Delete a resource	http://example.com/api/tasks/1/

Table: HTTP request methods and actions.

Data Storage Approaches

There are two broad categories of approaches for the databases & their pros & cons.

→ Relational (SQL) Approach:

- A Relational database is database that conforms to the relational model popularized by Edgar Codd in 1970.
- The below table summarizes 12 rules that codd introduced for relational databases.

Rule	Description
1. Information Rule.	All information in a relational database is represented explicitly at logical level & in exactly one way - by values in tables.
2. Guaranteed Access Rule	All data must be accessible. Every individual scalar value in relational DB is guaranteed to be logically accessible by specifying table name, primary key value & column name.
3. Systematic treatment of null values	Null values are supported in fully relational DBMS for representing missing information & inapplicable information in a systematic way.
4. Dynamic online catalog based on relational model	The system must support an online, relational catalog to authorized users using same relational language.
5. Comprehensive sub-language	A relational system must support atleast one language whose statements are expressible, by a well-defined syntax, as character,

6. View updating rule	All views that are theoritically updatable are also updatable by system
7. High level insert, update, delete	The system must support set at a time insert, update & delete operators.
8. Physical data independence	Any changes made in either storage representations (or) access methods, should not require the applications to be changed.
9. Logical data independence	Changes to the logical level must not require a change to an application based on the structure.
10. Integrity independence	Integrity constraints are specific to a particular relational database must be definable in relational data sub-language & storable in catalog, not in the application programs.
11. Distribution independence	A relational DBMS has distribution dependence. That is, existing applications should continue to operate successfully when existing distributed data are redistributed around the system.
12. Non-subversion rule.	If a relational system has a low-level language, that low level cannot be used to subonext the integrity rules & constraints expressed in higher-level relational language.

Table: Codd's rules for relational database.

A relational database has various constraints described as follows:

- Domain Constraint: It restricts, the domain of each attribute (or) set of possible values for attribute.
- Entity Integrity Constraint: It states that no primary key value can be null.
- Referential Integrity Constraint: It is required to maintain consistency among tuples in two relations, referential integrity requires every value of one attribute to exist as a value of another attribute in another relation.
- Foreign Key: For cross-referencing between multiple

Relational databases support atleast one comprehensive sub-language i.e., structured query language (SQL).
Relational databases provide ACID guarantees that database transactions are processed reliably as described below:

Atomicity: Atomicity property ensures that each transaction is either all (or) nothing.

Consistency: It ensures that each transaction brings the database from one valid state to another.

Isolation: It ensures concurrency control, i.e., results of incomplete transactions are not visible to other transactions. The transactions are isolated from each other until they finish.

Durability: It ensures that once a transaction committed, the data remains as it is, i.e., it is not affected by system outages such as power loss.

Ex: Oracle DB, Microsoft SQL Server, MySQL, SAP Sybase IQ, etc.

Pros and Cons of Relational DB:

Pros	Cons
1. Well defined consistent model.	1. Performance is the major constraint for relational DB.
2. Provide ACID guarantees.	2. Limited support for complex data structures.
3. Relational integrity maintained through entity and referential integrity constraints.	3. A complete knowledge of the DB structure is required to create & adhoc queries.
4. Well suited for online transaction processing (OLTP) application.	4. Most relational DB systems are expensive.
5. Stable & standardized databases available.	5. Some relational DB have limits on size of fields.
6. The DB design & normalization steps are well-defined & underlying structure is well understood.	6. Integrating data from the multiple relational DB systems can be cumbersome.

Non-Relational (No-SQL) Approach:

- Non-Relational database are becoming popular with the growth of cloud computing.
- Non-relational databases have better horizontal scaling

- Non-relational databases donot provide ACID guarantees.
- Most non-relational databases offer eventual consistency.
- Some authors have refered to the term BASE (Basically Available, Soft State, Eventual Consistency) guarantees for non-relational databases as opposed to ACID guarantees provided by relational databases
- Non-relational databases can achieve high scalability, fault tolerance & availability These databases can be distributed on a large cluster of machines.
- Unlike relational databases, the non-relational DB donot have a strict schema.
- The records can be in the form of key-value pairs (or) documents
- The commonly used categories include:

- Key-value store: Key-value store databases are suited for applications that require storing unstructured data without a fixed schema.
- Document Store: Doc store databases store semi-structured data in the form of documents which are encoded in different standards such as JSON, XML, BSON, YAML.
- Graph Store: These are designed for storing data that has graph structure (nodes & edges).
- Object Store: Object Store solutions are designed for storing data in the form of objects defined in an object-oriented programming language.

Pros and Cons of Non-relational Databases:

Pros	Cons
1. Easy to scale-out. Higher performance for massive scale data as compared to relational DB.	1. Donot provide ACID guarantees, therefore less suitable for applications such as transaction processing that require consistency.
2. Most solutions are either open-source (or) cheaper as compared to relational DB.	2. No fixed schema. There is no common data storage model Different solutions have different data storage models.
3. High availability & fault tolerance provided by data replication.	3. Limited support for aggregation as compared to relational databases.
4. No fixed schema. Support unstructured data.	4. No well-defined approach for database design.
5. Very fast retrieval of the data Suitable for real-time applications.	5. Lack of consistent model lead to solution lock_in

Introduction to Python:

Python is a general-purpose high level programming language suitable for providing solid foundation to the reader in the area of cloud computing.

The main characteristics of Python are:

Multi-paradigm programming language: Python supports more than one programming paradigm including object-oriented programming & structured programming.

Interpreted language: Python is an interpreted language & doesnot require an explicit compilation step. The python interpreter executes the program source code directly, statement by statement as a processor (or) scripting engine.

Interactive language: Python provides an interactive mode in which user can submit commands at the python prompt & interact with the interpreter directly.

The key benefits of python are: Easy-to-learn, read & maintain. Python is easy to learn yet an extremely powerful language for a wide range of application. Due to its simplicity, programs written in python are easy to maintain.

Object and Procedure Oriented: It supports both object oriented & procedure oriented programming. Procedure oriented allow programs to be written around procedures (or) function that allow reuse of code. Object oriented program to be written around objects that include both data & functionality.

Extendable: Python is an extendable language & allow integration of low-level modules written in such as cloud computing. It is useful when you want to speed up a critical portion of a program.

Scalable: Due to the minimalistic nature of python, it provides a manageable structure for large programs.

Portable: Python is an interpreted language, programmer donot have to worry about compilation, linking and loading of program. Python program can directly executed from source code & copied from one machine to other. The python interpretor converts the source code to an intermediate form called byte code & then translate into native language of the specific system and runs it.

Broad library Support: Python has a broad library support & works on various platform such as Windows,

image processing, network programming, cryptography.

* Python data types & data structures:

Numbers: Number is a datatype to store numeric values. Numbers are immutable datatype. Therefore, changing the value of number datatypes result in a newly allocated object.

Ex: Example working with Number

```
#integer
>>> a = 5
>>> type(a)
<type 'int'>

# floating point
  b = a.5
  type(a)
<type 'float'>

# Addition
>>> c = a+b
>>> c
   7.5
<type 'float'>

# Subtraction
>>> d = a-d
>>> d
   a.5
<type 'float'>
```

Strings: A string is simply a list of character in order. There are no limits to the no. of characters you have in string. A string has 0 character is called Empty String.

Ex: Working with strings

```
# Create String
>>> s = "Hello World!"
>>> type(s)
  <type 'str'>

# String concatenation
>>> t = "This is a sample program"
>>> r = s+t
>>> r
  "Hello World! This is a sample program"

# Print String
>>> print(s)
  Hello World!

# Convert to upper / lower case
  >>> s.upper()
   'HELLO WORLD!'
  >>> s.lower()
     'hello world!'
```

Lists: List is a compound datatype used to group together other values. A list item separated by commas & enclosed within separate brackets.

Ex: Working with lists.

```
>>> fruits = ['apple', 'orange', 'banana', 'mango']
>>> type(fruits)
<type 'list'>
>>> len(fruits)
4
```

Appending an item to a list

```
>>> fruits.append('pear')
>>> fruits
['apple', 'orange', 'banana', 'mango', 'pear']
```

Combining lists

```
>>> vegetables = ['carrot', 'potato', 'onion', 'beans', 'radish']
>>> vegetables
['carrot', 'potato', 'onion', 'beans', 'radish']
>>> eatables = fruits + vegetables
>>> eatables
['apple', 'orange', 'banana', 'mango', 'pear', 'carrot',
 'potato', 'onion', 'beans', 'radish']
```

Lists can be nested

```
>>> nested = [fruits, vegetables]
>>> nested
[['apple', 'mango', 'orange', 'banana', 'pear'], ['potato',
  'onion', 'carrot', 'beans', 'radish']]
```

Tuples: A tuple is a sequence data type that is similar to list. A tuple consists of no. of values separated by commas & within paranthesis. The elements of tuples cannot be changed, so tuple thought of read-only list.

Ex: Working with Tuple

```
>>> fruits = ("apple", "mango", "banana", "pineapple")
>>> fruits
("apple", "mango", "banana", "pineapple")
>>> type(fruits)
<type 'tuple'>
```

Get length of tuple

```
>>> len(fruits)
```

Dictionaries: Dictionary is a mapping data type, a kind of hash table that maps keys to values. Keys in dictionary can be of any datatype, though number & string are commonly used for keys.

Ex: Working with dictionaries

```
>>> student = {'name' : 'Mary', 'id' : '8776' }
>>> student
  {'name' : 'Marv' , 'id' : '8776' }

# Get length of a dictionary
>>> len (student)
  2
```

Type Conversions:

Ex: Working with Type Conversions

```
# Convert to String
>>> a = 10000
>>> str(a)
    '10000'

# Convert to int
>>> b = "2013"
>>> int(b)
  2013
```

* **Control Flow:** A control flow is the order in which the program code executes. A control flow of python regulated by conditional, loop & functional calls.

If: The If statement in python is similar to 'if' statement in other languages.

Ex:
```
>>> a = 25 * * 5
>>> if  a> 10000 :
        print ("More")
    else :
        print ("Less")
```
O/p: More.

For: The For statement in python iterates over items of any sequence (list, string) in the order in which they appear in the sequence. The behavior is different from the 'for' statement in other language such as 'c', which initialization, incrementing.

Ex:
```
hellostring = "Hello World"
fruits = ['apple', 'orange', 'banana', 'mango' ]
student = 'name' : 'Mary' , 'id' : '8776'
# Looping over characters in a string
```

While : The while statement in Python executes the statement within the while loop as long as the while condition is true.

Ex : # prints even numbers upto 100

```
>>> i = 0
>>> while i <= 100 :
        if i % 2 == 0 :
            print (i)
            i = i+1
```

Range : The Range statement in Python generates a list of numbers in arithmetic progression.

Ex : Generate a list of numbers from 0-9

```
>>> range (10)
    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Break / Continue : The break & continue statements in Python are similar to the statements in C. The break statement breaks out of the for / while loop whereas the continue statement continue with next iteration.

Ex : # Break statement example

```
>>> y = 1
>>> for x in range (4, 256, 4) :
        y = y * x
        if y > 518 :
            break
        print y
```

o/p : 4
32
384

Pass : The pass statement in Python is a null operation. The pass statement is used when a statement is required syntactically but you donot want any command (or) code to execute.

Ex :

```
fruits = ['apple', 'orange', 'banana', 'mango']
for items in fruits :
    if item == 'banana' :
        poss
    else :
        print (item)
```

o/p : apple
orange
mango

Functions: A function is a block of code that takes information in the form of parameters. A function in Python is a block of code begins with the keyword 'def' followed by the function name & paranthesis.

Ex:
```
>>> def displayFruits ( fruits = ['apple', 'orange' ] ) :
        print ( " There are %d fruits in the list "
            %. ( len ( fruits ) )
        for items in fruits :
            print (item)
```

```
#Using default arguments
>>> displayFruits ( )
```

o/p: apple
orange

Modules: Python allows organizing program code into different modules which improves the code readability & management. It defines functionality in the form of functions (or) classes module can be imported using the import keyword.

Ex:
```
def averageGrade ( student ) :
    sum = 0.0
    for key in student :
        sum = sum + student [ key ] ['grade' ]
    average = sum/ (en ( student )
    return average

def printRecords ( student ) :
    print ( " There are %d students" %(len (student)))
    i = 1
    for key in student :
        print ( " Student %d :" %. (i))
        print ( " Name :" + student [key ]['name' ] )
        print ( " Grade :" + str ( student [key ] ['grade' ]))
    i = i + 1
```

```
>>> import student
>>> student = '1' : 'name' : 'Bob' , 'grade' : 2.5 ,
              '2' : 'name' : 'Mary' , 'grade' : 3.5 ,
              '3' : 'name' : 'David' , 'grade' : 4.2
>>> student. PrintRecords ( student )
```

o/p: There are 3 students
Student : 1 :
Name : Bob

```
Grade : 3.5
Student - 3 :
Name : David
Grade : 4.2
>>> avg = student. averagegrade (Student)
>>> print ("The average grade is : %. 0.2 f" %. (avg))
```

o/p: 3.4

Packages: Package is a hierarchial file structure that consists of module & subpackages. The package is organized into a root directory with sub directories. Each directory contains a special file named __init__.py to treat directory as package.

```
Ex: slc image/                    // Toplevel package
    __init__.py                   // Treat directory as a package
    color/                        // color subpackage
        __init__.py
        colorconv.py
        colorlabel.py
        rgb_colors.py
        draw/                     // draw subpackage
            __init__.py
            draw.py
            setup.py
```

File Handling: Python allows reading & writing to files using the file object. The open(filename, mode) function is used to get a file object. The mode can be read(r), write(w), append(a), read and write (r+ or w+), read-binary (rb), write-binary (wb).

Ex:

```
>>> fp = open ('file.txt', 'r')
>>> content = fp. read()
>>> print (content)
>>> fp. close()
```

Date/ Time Operations: Python provides several functions for date & time access & conversions. The date & time module allows manipulating date & time in several ways.

```
Ex: >>> from datetime import date
    >>> now = date. today()
    >>> print("Date =" + now. strftime("%. m - %. d - %. y")
    Date : 07 - 24 - 13
    >>> print ("Day of week :" + now. strftime ("%. A")
```

```
>>> print ("Month : " + now.strftime ("%B"))
Month : July
>>>
>>> then = date (2013, 6, 7)
>>> timediff = now - then
>>> timediff.days
47
```

* **Classes**: Python is an object-oriented programming if provides all the standard features of OOPS such as class, class variables, class methods, inheritance, function overloading, operator overloading.

Class: A class is a simple type of object & user defined prototype for an object composed of three things name, attributes & operations / methods.

Instance / Object: An object is an instance of data structure defined by a class.

Inheritance: It is the process of forming new class from an existing class (or) base class.

Function Overloading: It is a form of polymorphism that allow a function to have different meanings, depending on its context.

Operator Overloading: Operator overloading is the form of polymorphism that allow assignment of more than one function to a particular operator.

Function Overriding: Function overriding allows a child class to provide a specific implementation of function already provided by the base class. Child class implementation override same name, parameters & return type as function in base class.

Ex:
```
>>> class student :
        studentcount = 0
        def __init__ (self, name, id) :
            print ("Constructor called")
            self.name = name
            self.id = id
        student.studentcount = student.studentcount + 1
        self.grades
        def __del__ (self) :
            print ("Destructor called")
        def getstudentCount (self) :
            return student.StudentCount
```

```
        def getGrade(self, key):
            return self.grades[key]
        def printGrades(self):
            for key in self.grades:
                print(key + ":" + self.grades[key])
```

```
>>> s = student('Steve', '98988')
o/p: constructor called
>>> s.addGrades('Math', '90')
>>> s.addGrades('Physics', '85')
>>> s.printGrades()
o/p: Physics: 85
     Math: 90
>>> Mathgrade = s.getGrade('Math')
>>> print(Mathgrade)
o/p: 90
>>> count = s.getStudentCount()
>>> print(count)
o/p: 1
>>> del s
o/p: Destructor called
```

7.1 Python for Amazon Web Services

In this section you will learn how to use Python for Amazon Web Services. Boto is a Python package that provides interfaces to Amazon Web Services (AWS) . Currently Boto supports the following AWS services:

• Compute

- Amazon Elastic Compute Cloud (EC2)
- Amazon Elastic MapReduce (EMR)
- AutoScaling

• Content Delivery

- Amazon CloudFront

• Database

- Amazon Relational Data Service (RDS)
- Amazon DynamoDB
- Amazon SimpleDB
- Amazon ElastiCache
- Amazon Redshift

• Deployment and Management

- AWS Elastic Beanstalk
- AWS CloudFormation
- AWS Data Pipeline

• Identity & Access

- AWS Identity and Access Management (IAM)

• Application Services

- Amazon CloudSearch
- Amazon Simple Workflow Service (SWF)
- Amazon Simple Queue Service (SQS)
- Amazon Simple Notification Server (SNS)
- Amazon Simple Email Service (SES)

• Monitoring

- Amazon CloudWatch

• Networking

- Amazon Route53
- Amazon Virtual Private Cloud (VPC)
- Elastic Load Balancing (ELB)
- Payments and Billing
- Amazon flexible payments service(FPS)

• Storage

- Amazon Simple Storage Service (S3)

- Amazon Glacier
- Amazon Elastic Block Store (EBS)

7.1.1 Amazon EC2

Amazon EC2 (Elastic Compute Cloud) is an Infrastructure as a Service (IaaS) provided by Amazon. It allows users to create and use virtual machines in the cloud.

- EC2 provides scalable and pay-as-you-go computing power. This means you can increase or decrease resources as needed and pay only for what you use.
- EC2 runs virtual machines, called instances, inside Amazon's cloud computing environment.
- The Python program shown in Box 7.1 is used to launch an EC2 instance using the Boto library.
- The program first connects to the EC2 service by calling the boto.ec2.connect_to_region function.
- While connecting, the AWS region, access key, and secret key are provided to authenticate the user.
- After the connection is established, the program launches a new EC2 instance using the conn.run_instances function.
- The AMI ID, instance type, EC2 key handle, and security group are passed as parameters to this function.
- The run_instances function returns a reservation object that contains information about the launched instance.
- The instance is obtained from the reservation using reservation.instances.
- The program checks the status of the instance using the instance.update function.
- **Waiting for Instance to Run**
  The program waits until the instance reaches the "running" state.
- **Displaying Instance Information**
  Once the instance is running, the program displays details such as the public DNS name, instance IP address, and launch time.

Python program for launching an EC2 instance

```
import boto.ec2
from time import sleep

ACCESS_KEY = "<enter access key>"
SECRET_KEY = "<enter secret key>"
REGION = "us-east-1"
AMI_ID = "ami-d0f89fb0"
EC2_KEY_HANDLE = "<enter key handle>"
INSTANCE_TYPE = "t1.micro"
SECGROUP_HANDLE = "default"
print "Connecting to EC2"
conn = boto.ec2.connect_to_region(
   REGION,
aws_access_key_id=ACCESS_KEY,
aws_secret_access_key=SECRET_KEY)
print "Launching instance with AMI-ID %s, with keypair %s, instance type %s, security
group %s" % (
   AMI_ID, EC2_KEY_HANDLE, INSTANCE_TYPE, SECGROUP_HANDLE)
```

```
reservation = conn.run_instances(
image_id=AMI_ID,
key_name=EC2_KEY_HANDLE,
instance_type=INSTANCE_TYPE,
security_groups=[SECGROUP_HANDLE])
instance = reservation.instances[0]
print "Waiting for instance to be up and running"
status = instance.update()
while status == 'pending':
sleep(10)
   status = instance.update()
if status == 'running':
   print "Instance is now running. Instance details are:"
   print "Instance Size: " + str(instance.instance_type)
   print "Instance State: " + str(instance.state)
   print "Instance Launch Time: " + str(instance.launch_time)
   print "Instance Public DNS: " + str(instance.public_dns_name)
   print "Instance Private DNS: " + str(instance.private_dns_name)
   print "Instance Public IP: " + str(instance.ip_address)
   print "Instance Private IP: " + str(instance.private_ip_address)
```

7.1.2 Amazon AutoScaling

- Amazon AutoScaling is a service that automatically increases or decreases the number of EC2 instances based on user-defined conditions.
- Purpose of AutoScaling
  It helps applications handle sudden increases in workload and reduces resources when demand is low, which saves cost.
- Automatic Scaling
  During high traffic or workload spikes, AutoScaling adds more EC2 instances. When the workload decreases, it removes extra instances.
- Creating an AutoScaling Group
  The Python code shown in Box 7.4 demonstrates how to create an AutoScaling group using the Boto library.
- Connecting to AutoScaling Service
  The program first connects to the AutoScaling service using the boto.ec2.autoscale.connect_to_region function.
- Providing AWS Credentials
  The EC2 region, AWS access key, and AWS secret key are provided to authenticate the connection.
- Creating a Launch Configuration
  After connecting, a launch configuration is created using conn.create_launch_configuration.
- What is a Launch Configuration?
  A launch configuration contains instructions for launching new EC2 instances, such as the AMI ID, instance type, and security groups.

- Creating an AutoScaling Group
  The launch configuration is then linked to a new AutoScaling group using conn.create_auto_scaling_group.
- Configuring AutoScaling Group Settings
  The AutoScaling group is configured with minimum and maximum number of instances, availability zones, and optional load balancers.
- Defining Scaling Policies
  Scaling policies are created to decide when to scale up or scale down the number of instances.
- Scale-Up Policy
  A scale-up policy is defined with ChangeInCapacity and scaling_adjustment = 1, which means one instance is added when triggered.
- Scale-Down Policy
  A scale-down policy is defined with ChangeInCapacity and scaling_adjustment = -1, which means one instance is removed when triggered.
- Creating CloudWatch Alarms
  Amazon CloudWatch alarms are created to monitor system metrics and trigger scaling policies.
- Scale-Up Alarm Condition
  The scale-up alarm is triggered when the average CPU utilization exceeds 70% for more than 60 seconds.
- Scale-Down Alarm Condition
  The scale-down alarm is triggered when the average CPU utilization drops below 50%.

Python code for deleting an AutoScaling group.

Before deleting a group, all the instances in the group must be terminated by calling `group.shutdown_instances`.

```
import boto.ec2.autoscale
from boto.ec2.autoscale import LaunchConfiguration
from boto.ec2.autoscale import AutoScalingGroup
from boto.ec2.cloudwatch import MetricAlarm
from boto.ec2.autoscale import ScalingPolicy
import boto.ec2.cloudwatch

ACCESS_KEY = "<enter access key>"
SECRET_KEY = "<enter secret key>"
REGION = "us-east-1"
print "Connecting to Autoscaling Service"
conn = boto.ec2.autoscale.connect_to_region(
   REGION,
aws_access_key_id=ACCESS_KEY,
aws_secret_access_key=SECRET_KEY)
print "Get all groups"
groups = conn.get_all_groups()
print groups
group_to_delete = groups[0]
print "Shutting down all instances from group: " + str(group_to_delete)
group_to_delete.shutdown_instances()
print "Deleting group"
```

```
group_to_delete.delete()
print "Done!"
```

7.1.3 Amazon S3

Amazon S3 is an online cloud-based data storage infrastructure for storing and retrieving any amount of data. S3 provides highly reliable, scalable, fast, fully redundant, and affordable storage infrastructure.

- **Connecting to Amazon S3**
  First, a connection to the Amazon S3 service is established using the boto.connect_s3 function.
- **Providing AWS Credentials**
  The AWS access key and AWS secret key are passed to the function to authenticate the user.
- **Defining Upload Functions**
  The program defines two separate functions to upload files to Amazon S3.
- **Uploading to a Specific Folder (Path)**
  The upload_to_s3_bucket_path function uploads a file to a specific path (folder) inside the S3 bucket.
- **Uploading to the Bucket Root**
  The upload_to_s3_bucket_root function uploads a file directly to the root of the S3 bucket without placing it inside any folder.

Python program for uploading a file to an S3 bucket

```
import boto.s3
 ACCESS_KEY = "<enter access key>"
SECRET_KEY = "<enter secret key>"
 conn = boto.connect_s3(
aws_access_key_id=ACCESS_KEY,
aws_secret_access_key=SECRET_KEY)
 def percent_cb(complete, total):
    print ('.')
 def upload_to_s3_bucket_path(bucketname, path, filename):
mybucket = conn.get_bucket(bucketname)
fullkeyname = os.path.join(path, filename)
    key = mybucket.new_key(fullkeyname)
key.set_contents_from_filename(filename, cb=percent_cb, num_cb=10)
 def upload_to_s3_bucket_root(bucketname, filename):
mybucket = conn.get_bucket(bucketname)
    key = mybucket.new_key(filename)
key.set_contents_from_filename(filename, cb=percent_cb, num_cb=10)
 upload_to_s3_bucket_path("mybucket2013", "data", "file.txt")
```

7.1.4 Amazon RDS

Amazon RDS (Relational Database Service) is a cloud service that allows users to create and manage relational databases such as MySQL, Oracle, and Microsoft SQL Server.

- **Purpose of Amazon RDS**
  RDS makes it easy to set up, operate, and scale relational databases in the cloud without managing the underlying infrastructure.
- **Launching an RDS Instance Using Python**
  The Python program shown in Box 7.7 is used to launch an Amazon RDS instance.

- **Connecting to RDS Service**
  The program first connects to the RDS service by calling the boto.rds.connect_to_region function.
- **Providing AWS Credentials**
  The AWS region, access key, and secret key are passed to authenticate the connection.
- **Creating a Database Instance**
  After connecting, the conn.create_dbinstance function is used to create a new RDS instance.
- **Providing Database Details**
  The function requires several parameters such as:
- Instance ID
- Database size
- Instance type
- Database username and password
- Database engine (e.g., MySQL)
- Database name
- Security groups
- **Waiting for Database Availability**
  The program waits until the RDS instance status becomes **available**.
- **Displaying RDS Instance Details**
  Once the instance is ready, the program displays details such as instance ID, creation time, and database endpoint.

7.1.5 Amazon DynamoDB

- Amazon DynamoDB is a fully managed, scalable, and high-performance NoSQL database service.
- **Establishing a Connection**
  A connection to the DynamoDB service is established by calling boto.dynamodb.connect_to_region.
- **Connection Parameters**
  The following details are provided to the connection function:
- DynamoDB region
- AWS access key
- AWS secret key
- **Creating the Table Schema**
  After connecting to DynamoDB, a schema for the new table is created using conn.create_schema.
- **Schema Components**
  The schema specifies:
- Hash key name and type
- Range key name and type
- **Creating the DynamoDB Table**
  The table is created by calling conn.create_table

7.1.6 Amazon SQS

Amazon Simple Queue Service (SQS) is a highly scalable and reliable hosted queue service used to store messages as they move between different components of an application.

Establishing a Connection

A connection to the SQS service is established by calling `boto.sqs.connect_to_region`.

Connection Parameters

The following information is passed to the connection function: AWS region,AWS access key,AWS secret key

Creating a Queue

After successfully connecting to the SQS service, the function `conn.create_queue` is used to create a new queue by providing the queue name as an input parameter.

Retrieving All Queues

The function `conn.get_all_queues` is used to retrieve a list of all available SQS queues.

7.2 Python for Google Cloud Platform

In this section you will learn how to use Python for Google Cloud Platform. The examples in this section use the Google APIs Python Client library for accessing Google APIs.

7.2.1 Google Compute Engine

Google Compute Engine provides scalable and flexible virtual machine computing capabilities in the cloud

Box 7.18: Credentials file for Google Cloud examples – client_secrets.json

```
{
 "installed": {
   "client_id": "<enter client id>",
   "client_secret": "<enter client secret>",
   "redirect_uris": ["http://localhost:8080/", "urn:ietf:wg:oauth:2.0:oob"],
   "auth_uri": "https://accounts.google.com/o/oauth2/auth",
   "token_uri": "https://accounts.google.com/o/oauth2/token"
 }
}
```

Step 1: The program requests access to Google Compute Engine using this OAuth 2.0 scope: `https://www.googleapis.com/auth/compute`

This scope allows the script to create and manage VM instances.

Step 2: The script reads your OAuth credentials from: `client_secrets.json`

This file contains your **client ID** and **client secret**, which identify your application to Google.

Step 3: The user is prompted (usually via a browser) to log in and approve access.

After approval, Google returns an **access token**.

Step 4: The access token is saved locally in:`oauth2.dat`

This prevents the need to repeat the login and authorization process every time the script runs.

Step 5: If `oauth2.dat` already exists and the token is still valid, the script uses it directly without asking for authorization again.

Step 6: After authentication, the script builds a **Google Compute Engine service object** using the API client library.

This object is used to send requests to GCE.

Step 7: The script defines a JSON request body containing the VM details, such as:

- Instance name
- Machine type (e.g., `n1-standard-1`)
- Zone (e.g., `us-central1-a`)
- Boot disk image
- Network interfaces

Step 8: The script calls the Compute Engine API method: `instances().insert()`.This sends the VM configuration to Google Cloud.

Step 9: Google processes the request and starts creating the virtual machine with the specified configuration.
Step 10: Once provisioning is complete, your new Compute Engine VM is up and running.
Authorize → Save token → Connect to GCE → Send VM configuration → Instance gets created.

7.2.2 Google Cloud Storage

- Google Cloud Storage is a cloud service for storing data in the Google's cloud [106].
- Data stored on Google Cloud Storage is organized into buckets.
- Box 7.21 shows the Python program for uploading a file to a Google Cloud Storage bucket. This example uses the OAuth 2.0 scope (`https://www.googleapis.com/auth/devstorage.full_control`) and credentials in the credentials file to request a fresh access token, which is then stored in the `oauth2.dat` file.
- After completing the OAuth authorization, an instance of the Google Cloud Storage service is obtained. To upload a file the `objects().insert` method of the Google Cloud Storage API is used.
- The request to this method contains the bucket name, file name and media body containing the `MediaIoBaseUpload` object created from the file contents.

7.21: Python program for uploading a file to Google Cloud Storage

```
import httplib2
from oauth2client.client import flow_from_clientsecrets
from oauth2client.file import Storage
from apiclient.errors import HttpError
from oauth2client.client import AccessTokenRefreshError
from oauth2client.tools import run
from apiclient.discovery import build
from apiclient.http import MediaIoBaseUpload
      import io
      API_VERSION = 'v1beta2'
GS_SCOPE = 'https://www.googleapis.com/auth/devstorage.full_control'
CLIENT_SECRETS = 'client_secrets.json'
OAUTH_STORAGE = 'oauth2.dat'
BUCKET = 'bucket'
FILENAME = 'file.txt'
FILE_TYPE = 'text/plain'
      def main():
   # OAuth 2.0 authorization
   flow = flow_from_clientsecrets(CLIENT_SECRETS, scope=GS_SCOPE)
   storage = Storage(OAUTH_STORAGE)
   credentials = storage.get()

   if credentials is None or credentials.invalid:
      credentials = run(flow, storage)
      http = httplib2.Http()
   auth_http = credentials.authorize(http)
     gs_service = build('storage', API_VERSION, http=auth_http)
     # Uploading file
```

```
fp = open(FILENAME, 'r')
io = io.BytesIO(fp.read().encode())
media = MediaIoBaseUpload(io, FILE_TYPE)
request = gs_service.objects().insert(
    bucket=BUCKET,
    name=FILENAME,
    media_body=media
)
response = request.execute()
  if __name__ == '__main__':
main()
```

7.2.3 Google Cloud SQL

- Google Cloud SQL is a MySQL database in the Google's cloud .
- Box 7.24 shows the Python program for launching a Google Cloud SQL instance. This example uses the OAuth 2.0 scope (`https://www.googleapis.com/auth/sqlservice.admin`) and credentials in the credentials file to request a fresh access token, which is then stored in the `oauth2.dat` file.
- After completing the OAuth authorization, an instance of the Google Cloud SQL service is obtained. To launch a new instance the `instances().insert` method of the Google Cloud SQL API is used.
- The request body to this method contains properties such as instance name, project, tier, pricingPlan and replicationType.

7.2.4 Google BigQuery

- Google BigQuery allows querying massive scale datasets with SQL-like queries [107].
- The BigQuery queries are run against append-only tables and use the processing power of Google's infrastructure for speeding up queries.
- The Python program for creating a BigQuery dataset uses the OAuth 2.0 scope (`https://www.googleapis.com/auth/bigquery`) and credentials in the credentials file to request a fresh access token, which is then stored in the `oauth2.dat` file.
- After completing the OAuth authorization, an instance of the Google BigQuery service is obtained. The `jobs().insert` method of the Google BigQuery API is used for inserting a dataset. The request body of this method contains properties such as configuration, load, and schema.
- In any example, the data is loaded from a CSV file. The schema property specifies the schema of the CSV file. The `jobs().insert` method returns immediately, therefore the `jobs.get` is called to get the job status.

7.2.5 Google Cloud Datastore

- Google Cloud Datastore is a NoSQL (i.e., schema-less) object datastore that provides robust and scalable storage . Data objects in the Datastore are known as entities.
- An entity has one or more named properties, each of which can have one or more values. Entities of the same kind need not have the same properties, and an entity's values for a given property need not all be of the same data type.

- Box 7.29 shows the Python program for creating a Google Cloud Datastore entity. This example uses the OAuth 2.0 scopes (`https://www.googleapis.com/auth/datastore` and `https://www.googleapis.com/auth/userinfo.email`) and credentials in the credentials file to request a fresh access token, which is then stored in the `oauth2.dat` file.
- After completing the OAuth authorization, an instance of the Google Cloud Datastore service is obtained. The `datasets().blendWrite` method of the Google Cloud Datastore API is used for creating a new entity.

7.2.6 Google App Engine

- Google App Engine is a web application hosting service [105]. Applications hosted in Google App Engine are easy to build, maintain, and scale. Applications run in a secure sandbox environment that provides limited access to the underlying operating system.
- The sandbox isolates an application in its own secure, reliable environment that is independent of the hardware, operating system and physical location of the server. The benefit of the sandbox is that it allows Google App Engine to distribute web requests for the application across multiple servers, and start up servers to meet application workloads.
- Google App Engine includes a simple web application framework called `webapp2`. App Engine also supports any framework written in pure Python that speaks WSGI, including Django, CherryPy, Pylons, web.py, and web2py.
- Box 7.31 shows the Python code for a simple Google App Engine application for student records. This application uses the App Engine Users service for signing into the application using Google accounts.
-
- For storing data, the App Engine Datastore is used. The application contains two request handlers, one for the main page (`MainPage`) and the other for saving records (`SaveRecord`).

7.3 Python for Windows Azure

Windows Azure provides three compute models that you can use to host web applications: Web Sites, Cloud Services, and Virtual Machines .

7.3.1 Azure Cloud Service

An application that is run in Windows Azure is called a Windows Azure cloud service that includes the application code and configuration. To deploy an application in Windows Azure

Figure 7.1: Student Records App - screenshot of login page

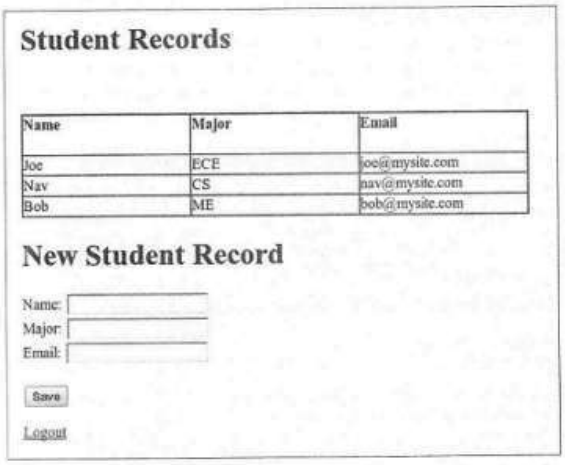


Figure 7.2: Student Records App - screenshot of home page

as a cloud service, three components are needed – service definition file, service configuration file, and service package.

Below as shows an example of creating a cloud service using the Azure service management API. Cloud service is created using the `create_hosted_service` method.

Python example for creating an Azure cloud services

```
from azure import *
from azure.servicemanagement import *
subscription_id = '<enter subscription ID>'
certificate_path = '<enter PEM file path>'
sms = ServiceManagementService(subscription_id, certificate_path)
name = 'mycloudservice'
label = 'mycloudservice'
des = 'my hosted service'
location = 'West US'
# You can enter the location or an affinity group
sms.create_hosted_service(name, label, des, location)
```

7.3.2 Azure Virtual Machines

Windows Azure Virtual Machines allows you to provision on-demand, scalable compute infrastructure. Box 7.3.4 shows an example of creating a new Azure virtual machine. To create a virtual machine, a cloud service is first created. Virtual machine is created using the `create_virtual_machine_deployment` method of the Azure service management API.

Python example of creating an Azure virtual machine

```
from azure import *
from azure.servicemanagement import *
subscription_id = '<enter subscription ID>'
certificate_path = '<enter PEM file path>'
sms = ServiceManagementService(subscription_id, certificate_path)
name = 'mycloudservice'
location = 'West US'
# You can either use the location or an affinity group
sms.create_hosted_service(service_name=name,
                          label=name,
                          location=location)
# Name of an OS image as returned by list_os_images
image_name =
'b97a78b8c4d52b05eac6a2ebad85__Ubuntu-12_04_2-LTS-amd64-server-20130528-en-us-30
GB'
# Destination storage account container/blob where the VM disk will be created
media_link = 'http://mystorage.blob.core.windows.net/mycontainer/ubuntu.vhd'
# Linux VM configuration
linux_config = LinuxConfigurationSet('mystorage', 'myusername', 'mypassword', True)
os_hd = OSVirtualHardDisk(image_name, media_link)
sms.create_virtual_machine_deployment(service_name=name,
                                      deployment_name=name,
                                      role_name=name,
                                      role_type='PersistentVMRole',
                                      system_config=linux_config
                                      os_virtual_hard_disk=os_hd
                                      roll_size=’small’)
```

7.3.3 Azure Storage

Windows Azure Storage services allow you to store and access various forms of data (Blobs, Tables and Queues). Blobs are used to store unstructured binary and text data. Tables are used to store non-relational structured data. Queues are used to store messages that a client can access. A storage service must be created before storing data in blobs, tables or queues. Box 7.3.5 shows an example of creating a new Azure storage service using the create_storage_account method of the service management API. Box 7.3.6 shows an example listing Azure storage services.

Box 7.3.5: Python example of creating an Azure storage service

```
from azure import *
from azure.servicemanagement import *
subscription_id = '<enter subscription ID>'
certificate_path = '<enter PEM file path>'
sms = ServiceManagementService(subscription_id, certificate_path)
name = 'mystorage'
label = 'mystorage'
location = 'West US'
desc = 'My storage account'
result = sms.create_storage_account(name, desc, label, location=location)
operation_result = sms.get_operation_status(result.request_id)
print('Operation status: ' + operation_result.status)
```

Azure Blob Service

Azure Blobs service allows you to store large amounts of unstructured text or binary data such as video, audio and images. Box 7.3.7 shows an example of using the Blob service for storing a file. Blobs are organized in containers. The `create_container` method is used to create a new container. After creating a container the blob is uploaded using the `put_blob` method. Blobs can be listed using the `list_blobs` method. To download a blob, the `get_blob` method is used.

Box 7.3.7: Python example of using Azure Blob Service

```
from azure.storage import *
key = '<enter key>'
blob_service = BlobService(account_name='myaccountname', account_key=key)
# Create Container
blob_service.create_container('mycontainer')
# Upload Blob
filename = 'images.txt'
myblob = open(filename, 'r').read()
blob_service.put_blob('mycontainer', filename, myblob, x_ms_blob_type='BlockBlob')
# List Blobs
blobs = blob_service.list_blobs('mycontainer')
for blob in blobs:
   print(blob.name)
   print(blob.url)
# Download Blob
output_filename = 'output.txt'
blob = blob_service.get_blob('mycontainer', 'myblob')
with open(output_filename, 'w') as f:
   f.write(blob)
```

Azure Table Service

Azure Table service provides No-SQL capabilities for applications that require storage of large amounts of unstructured data. Box 7.3.8 shows an example of using the Table service. To create a table method is used to create a new table. A table is a collection of entities. Tables don't have a fixed schema. Therefore entities in a table can have different sets of properties. Entity is a set of properties. To insert an entity into a table, the insert_entity method is used. Entities stored in a table can be queried using the query_entities method.

Python example of using Azure Table Service

```
from azure.storage import *
key = '<enter key>'
table_service = TableService(account_name='myaccountname', account_key=key)
# Create Table
table_service.create_table('gstudents')
# Insert Entity
item = {'PartitionKey': 'C-Students', 'RowKey': '1', 'Name': 'Fred', 'Major': 'CS', 'Grade': 3.8}
table_service.insert_entity('gstudents', item)
# Query Entities items = table_service.query_entities('gstudents', "PartitionKey eq
'C-Students'")for item in items:
```

```
    print(item.Name)
    print(item.Major)
    print(item.Grade)
```

Azure Queue Service

Windows Azure Queues store large numbers of messages. Box 7.3.9 shows an example of using the Queue service. To create a new queue the create_queue method is used. Messages are inserted into the queue using the put_message method and retrieved using the get_messages method.

Python example of using Azure Queue Service

```
from azure.storage import *
import time
key = '<enter key>'
queue_service = QueueService(account_name='myaccountname', account_key=key)
# Create Queue
queue_service.create_queue('myqueue')
# Insert Message into Queue
msg_datetime = time.asctime(time.localtime(time.time()))
msg = "Test message generated on: " + msg_datetime
queue_service.put_message('myqueue', msg)
# Get Queue Length
queue_metadata = queue_service.get_queue_metadata('myqueue')
count = queue_metadata.x_ms_approximate_messages_count
print("Queue Length: " + count)
# Retrieve Messages from Queue
messages = queue_service.get_messages('myqueue')
for message in messages:
    print(message.message_text)
    queue_service.delete_message('myqueue',
                                 message.message_id,
                                 message.pop_receipt)
```

7.4 Python for MapReduce

In this section you will learn how to create a MapReduce job in Python and run it on a Hadoop cluster. Let us create a MapReduce job for computing an inverted index from a set of text files. An inverted index consists of a number of rows where each row holds a unique term and list document identifiers in which the term occurs.

Box 7.40 shows the inverted index mapper program. The map function reads the data from the standard input (stdin) and splits the tab-limited data into document-ID and contents of the document. The map function emits key-value pairs where key is each word in the document and value is the document-ID.

Box 7.41 shows the inverted index reducer program. The key-value pairs emitted by the map phase are shuffled to the reducers and grouped by the key. The reducer reads the key-value pairs grouped by the same key from the standard input (stdin) and creates a list of document-IDs in which the word occurs. The output of reducer contains key-value pairs where key is a unique word and value is the list of document-IDs in which the word occurs.

Box 7.42 shows the commands to run the inverted index MapReduce program. First, we copy the directory containing the input to Hadoop filesystem. The input contains an aggregated file in which each line contains a document-ID and the contents of the document separated by a tab. A Hadoop streaming job is then created by specifying the input mapper and reducer

programs and the locations of the input and output. When the streaming job completes, the output directory will have a file containing the inverted index.

Box 7.40: Inverted Index Mapper in Python

```
#!/usr/bin/env python
import sys
for line in sys.stdin:
    doc_id, content = line.split('\t')
    words = content.split()
    for word in words:
        print("%s\t%s" % (word, doc_id))
```

Box 7.41: Inverted Index Reducer in Python

```
#!/usr/bin/env python
import sys
current_word = None
current_docids = []
word = None
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
 # parse the input we got from mapper.py
    word, doc_id = line.split('\t')
 if current_word == word:
        current_docids.append(doc_id)
    else:
        if current_word:
            print("%s\t%s" % (current_word, current_docids))
        current_docids = []
        current_docids.append(doc_id)
        current_word = word
```

Box 7.42: Running Inverted Index MapReduce on Hadoop Cluster

```
$HADOOP_HOME/bin/hadoop fs -copyFromLocal ./documents input
$HADOOP_HOME/bin/hadoop jar contrib/streaming/hadoop-streaming.jar \
-mapper mapper.py -reducer reducer.py \
-input input/ -output output
```

7.5 Python Packages of Interest

7.5.1 JSON

- JavaScript Object Notation (JSON) is an easy to read and write data-interchange format. JSON is used as an alternative to XML and is easy for machines to parse and generate.
- JSON is based on two structures – a collection of name-value pairs (e.g., a Python dictionary) and ordered lists of values (e.g., a Python list).
- JSON format is often used for serializing and transmitting structured data over a network connection, for example, transmitting data between a server and web application.

7.5.2 XML

XML (Extensible Markup Language) is a data format for structured document interchange. Box 7.45 shows an example of an XML file. In this section you will learn how to parse, read and write XML with Python. The Python *minidom* library provides a minimal implementation of the Document Object Model interface and an API similar to that in other languages. Box 7.46 shows a Python program for parsing an XML file. Box 7.47 shows a Python program for creating an XML file.

Box 7.45: XML example

```
<?xml version="1.0"?>
<catalog>
  <plant id="1">
    <common>Bloodroot</common>
    <botanical>Sanguinaria canadensis</botanical>
    <zone>4</zone>
    <light>Mostly shady</light>
    <price>2.44</price>
    <availability>031599</availability>
  </plant>
  <plant id="2">
    <common>Columbine</common>
    <botanical>Aquilegia canadensis</botanical>
    <zone>3</zone>
    <light>Mostly shady</light>
    <price>9.37</price>
    <availability>030699</availability>
  </plant>
  <plant id="3">
    <common>Marsh Marigold</common>
    <botanical>Caltha palustris</botanical>
    <zone>4</zone>
    <light>Mostly sunny</light>
    <price>6.81</price>
    <availability>051799</availability>
  </plant>
</catalog>
```

Box 7.46: Parsing an XML file in Python

```
from xml.dom.minidom import parse
dom = parse("test.xml")
for node in dom.getElementsByTagName("plant"):
    id = node.getAttribute("id")
    print("Plant ID:", id)
 common = node.getElementsByTagName("common")[0].childNodes[0].nodeValue
    print("Common:", common)
botanical = node.getElementsByTagName("botanical")[0].childNodes[0].nodeValue
    print("Botanical:", botanical)
zone = node.getElementsByTagName("zone")[0].childNodes[0].nodeValue
    print("Zone:", zone)
```

Box 7.47: Creating an XML file with Python

```
# Python example to create the following XML:
# <?xml version="1.0"?>
# <Class>
#   <Student>
#     <Name>Alex</Name>
#     <Major>ECE</Major>
#   </Student>
# </Class>
from xml.dom.minidom import Document
doc = Document()
# create base element
base = doc.createElement("Class")
doc.appendChild(base)
# create an entry element
entry = doc.createElement("Student")
base.appendChild(entry)
# create an element and append to entry element
name = doc.createElement("Name")
nameContent = doc.createTextNode("Alex")
name.appendChild(nameContent)
entry.appendChild(name)
# create an element and append to entry element
major = doc.createElement("Major")
majorContent = doc.createTextNode("ECE")
major.appendChild(majorContent)
entry.appendChild(major)
fp = open("foo.xml", "w")
doc.writexml(fp)
fp.close()
```

7.5.3 HTTPLib & URLLib

HTTPLib2 and URLLib2 are Python libraries used in network/internet programming [91, 92]. HTTPLib2 is an HTTP client library and URLLib2 is a library for fetching URLs.Box 7.48 shows an example of an HTTP GET request using the HTTPLib. The variable resp contains the response headers and content contains the content retrieved from the URL.

Box 7.48: HTTP GET request example using HTTPLib

```
>>> import httplib2
>>> h = httplib2.Http()
>>> resp, content = h.request("http://example.com", "GET")
>>> resp
{'status': '200', 'content-length': '1270', 'content-location':
 'http://example.com', 'x-cache': 'HIT', 'accept-ranges': 'bytes',
 'server': 'ECS (cpm/F858)', 'last-modified': 'Thu, 25 Apr 2013 16:13:23 GMT',
 'etag': '"7806024f-4d3b12978ec0"', 'date': 'Wed, 31 Jul 2013 12:36:05 GMT',
 'content-type': 'text/html; charset=UTF-8'}
```

```
>>> content
'<!doctype html>\n<html>\n<head>\n<title>Example Domain</title>\n
<meta charset="utf-8">\n'
```

7.5.4 SMTPLib

Simple Mail Transfer Protocol (SMTP) is a protocol which handles sending email and routing e-mail between mail servers. The Python **smtplib** module provides an SMTP client session object that can be used to send email.

Box 7.52 shows a Python example of sending email from a Gmail account. The string message contains the email message to be sent. To send email from a Gmail account, Gmail SMTP server is specified in the server string.

To send an email, first a connection is established with the SMTP server by calling smtplib.SMTP with the SMTP server name and port. The user name and password provided are then used to login to the server. The email is then sent by calling server.sendmail function with the from address, to address list and message as input parameters.

Box 7.52: Python example of sending email

```
import smtplib
from_email = "<enter-gmail-address>"
recipients_list = ["<enter-sender-email>"]
cc_list = []
subject = "Hello"
message = "This is a test message."
username = "<enter-gmail-username>"
password = "<enter-gmail-password>"
server = "smtp.gmail.com:587"
def sendemail(from_addr, to_addr_list, cc_addr_list,
              subject, message,
              login, password,
              smtpserver):
  header = "From: %s\n" % from_addr
    header += "To: %s\n" % ",".join(to_addr_list)
    header += "Cc: %s\n" % ",".join(cc_addr_list)
    header += "Subject: %s\n\n" % subject
  message = header + message
 server = smtplib.SMTP(smtpserver)
    server.starttls()
    server.login(login, password)
    problems = server.sendmail(from_addr, to_addr_list, message)
    server.quit()
# Send email
sendemail(from_email, recipients_list, cc_list,
        subject, message, username, password, server)
```

7.5.5 NumPy

NumPy is a package for scientific computing in Python . NumPy provides support for large multi-dimensional arrays and matrices.

NumPy capabilities include:

- Creation and manipulation of arrays
- Matrix support

• Binary and string operations
• Linear algebra
• Discrete Fourier Transform
• Statistics
• Random sampling
• Sorting, searching and counting
• Financial functions
• Polynomials
• Logic functions

NumPy matrix examples

```
# Creating a matrix
>>> A = matrix('5,6,8,2;3,6;1,6,9')
>>> A
matrix([[5, 6, 8],
        [2, 3, 6],
        [1, 6, 9]])

>>> B = matrix('2,4,6;7,3,1;7,8,2')
>>> B
matrix([[2, 4, 6],
        [7, 3, 1],
        [7, 8, 2]])

>>> B = matrix('2,4;7,1;7,2')
matrix([[2, 4],
        [7, 1],
        [7, 2]])

# Matrix transpose
>>> C = B.transpose()
>>> C
matrix([[2, 7, 7],
        [4, 1, 2]])
[4, 1, 2]]

# Matrix multiplication
>>> D = A * B
>>> D
matrix([[108, 42],
        [ 67, 23],
        [107, 28]])

# Matrix inverse
>>> A.I
matrix([[ 0.2,         0.13333333, -0.26666667],
        [ 0.26666667, -0.82222222,  0.31111111],
        [-0.2,         0.53333333, -0.06666667]])
```

7.5.6 Scikit-learn

Scikit-learn is an open source machine learning library for Python that provides implementations of various machine learning algorithms for classification, clustering, regression and dimension reduction problems.

Scikit-learn is a very useful package for big data analytics applications. In scikit-learn for clustering and classification of big data. Scikit-learn capabilities include :

- **Supervised learning**
    - Generalized Linear Models
    - Support Vector Machines
    - Stochastic Gradient Descent
    - Nearest Neighbors
    - Gaussian Processes
    - Partial Least Squares
    - Naive Bayes
    - Decision Trees
    - Ensemble methods
    - Multiclass and multilabel algorithms
- Feature selection
- Semi-Supervised learning
- Linear and Quadratic Discriminant Analysis
- Isotonic regression
- **Unsupervised learning**
    - Gaussian mixture models
    - Manifold learning
    - Clustering
    - Decomposing signals in components
    - Covariance estimation
    - Novelty and Outlier Detection
    - Hidden Markov Models
- **Model selection and evaluation**
    - Cross-Validation: evaluating estimator performance
    - Grid search: setting estimator parameters
    - Pipeline: chaining estimators
    - FeatureUnion: Combining feature extractors
    - Model evaluation

- **Dataset transformations**
  - Preprocessing data
  - Feature extraction
  - Kernel Approximation
  - Random Projection
  - Pairwise metrics, Affinities and Kernels
- Dataset loading utilities
  - General dataset API
  - Sample images
  - Sample generators

7.6 Python Web Application Framework – Django

Django is an open source web application framework for developing web applications in Python .

1. A web application framework in general is a collection of solutions, packages and best practices that allows development of web applications and dynamic websites.
2. Django is based on the Model-Template-View architecture and provides a separation of the data model from the business logic and the user interface.
3. Django provides a unified API to a database backend.
4. Django can work with different databases without requiring any code changes. With this flexibility in web application design combined with the powerful capabilities of the Python language and the Python ecosystem.
5. Django is best suited for cloud applications.
6. Django consists of an object-relational mapper, a web templating system and a regular-expression based URL dispatcher.

7.6.1 Django Architecture

Django is Model-Template-View (MTV) framework. The roles of model, template and view are:

Model

The model acts as a definition of some stored data and handles the interactions with the database. In a web application, the data can be stored in a relational database, non-relational database, an XML file, etc. A Django model is a Python class that outlines the variables and methods for a particular type of data.

Template

In a typical Django web application, the template is simply an HTML page with a few extra placeholders. Django's template language can be used to create various forms of text files (XML, email, CSS, Javascript, CSV, etc.).

View

The view ties the model to the template. The view is where you write the code that actually generates the web pages. View determines what data is to be displayed, retrieves the data from the database and passes the data to template.

7.6.2 Starting Development with Django

Project and App Creation

- Project files:

- __init__.py: Marks folder as Python package
- manage.py: Site management functions
- settings.py: Configuration settings
- urls.py: URL routing

- App files:
  - models.py: Data models
  - views.py: Application views

Commands (Box 7.57)

```
bash
django-admin.py startproject blogproject
python manage.py startapp myapp
python manage.py runserver
```

- Default server: http://localhost:8000

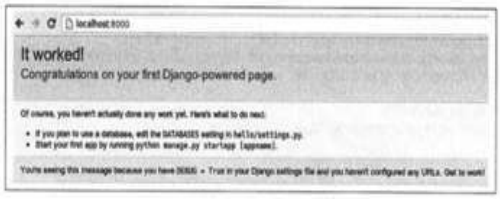


Figure 7.3: Django default project

Figure 7.3: Django default project

Database Configuration

Relational Database Support

- Supports MySQL, PostgreSQL, Oracle, SQLite.

MySQL Setup (Box 7.58)

```
bash
sudo apt-get install mysql-server mysql-client
sudo mysqladmin -u root -h localhost password 'mypassword'
```

MySQL Configuration (Box 7.59)

```
python
DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.mysql',
      'NAME': 'django',
      'USER': 'root',
      'PASSWORD': 'mypassword',
      'HOST': 'localhost',
      'PORT': '',
   }
}
```

Non-Relational Database Support

MongoDB Setup (Box 7.60)

```
bash
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10
echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' | sudo tee
/etc/apt/sources.list.d/10gen.list
sudo apt-get update
sudo apt-get install mongodb-10gen
```

Django-MongoDB Engine Setup

bash

```
sudo pip install https://github.com/wikiremedia/django-nonrel/get/tip.tar.gz
sudo pip install https://github.com/django-nonrel/mongodb-engine/get/tip.tar.gz
sudo pip install https://github.com/django-nonrel/django/get/tip.tar.gz
```

MongoDB Configuration (Box 7.61)

python

```
DATABASES = {
   'default': {
      'ENGINE': 'django_mongodb_engine',
      'NAME': '<database-name>',
      'HOST': '<mongodb-hostname>',
      'PORT': <mongodb-port>,
   }
}
```

Defining a Model (Box 7.62)

python

```
class Instructor(models.Model):
   first_name = models.CharField(max_length=30)
   last_name = models.CharField(max_length=40)
   email = models.EmailField()
   def __unicode__(self):
      return self.last_name

class Course(models.Model):
   name = models.CharField(max_length=30)
   semester = models.CharField(max_length=10)
   instructor = models.ForeignKey(Instructor)
   def __unicode__(self):
      return self.name

class Student(models.Model):
   first_name = models.CharField(max_length=30)
   last_name = models.CharField(max_length=40)
   email = models.EmailField()
   courses = models.ManyToManyField(Course)
   def __unicode__(self):
      return self.first_name
```

- Sync models with database:

bash

```
python manage.py syncdb
```

Django Admin Site

- Enables content management without extra code.
- Add django.contrib.admin and django.contrib.admindocs to INSTALLED_APPS.
- Register models in admin.py (Box 7.63)

python

```
from django.contrib import admin
from myapp.models import Instructor, Student, Course

admin.site.register(Instructor)
```

admin.site.register(Student)
admin.site.register(Course)

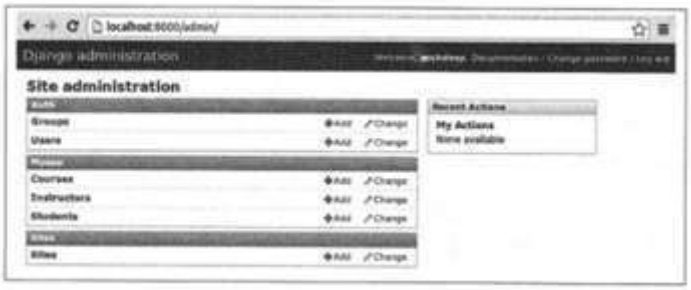


Figure 7.4: Screenshot of Django admin site

Figure 7.5: Django admin site - adding new items to Instructor table

Figure 7.6: Django admin site - adding new items to Course table

Figure 7.7: Django admin site - adding new items to Student table

Figure 7.4: Admin interface – Site administration , *Figure 7.5: Add Instructor form* , *Figure 7.6: Add Course form* , *Figure 7.7: Add Student form*

Defining a View

- Views retrieve data and pass it to templates.
- Also handle form submissions and database updates.

Example View (Box 7.64)

python

```
from django.shortcuts import render_to_response
from myapp.models import *
def student(request):
```

```
    student_entries = Student.objects.all()
    instructor_entries = Instructor.objects.all()
    course_entries = Course.objects.all()
    return render_to_response('student.html', {
        'students': student_entries,
        'instructors': instructor_entries,
        'courses': course_entries
    })
```

Defining a Template

- Templates separate presentation from data.
- Use placeholders like {{ entry.name }} and tags like {% for entry in list %}.

Template Example (Box 7.65) 📍

Figure 7.8: Screenshot of home page of Student Records app rendered from template in Box 7.65

Figure 7.8: Student Records home page

```
html
<h2>Students</h2>
<table>
<tr>
  <td><h3>Last Name</h3></td>
  <td><h3>Email</h3></td>
  <td><h3>Courses</h3></td>
</tr>
{% for entry in student_entries %}
<tr>
  <td>{{ entry.last_name }}</td>
  <td>{{ entry.email }}</td>
  <td>{{ entry.courses.all|join:", " }}</td>
</tr>
{% endfor %}
</table>
```

Defining URL Patterns

- URLs map to views using regular expressions.

Example (Box 7.66)

```
python
url(r'^records/(?P<studentname>\w+)', 'myapp.views.studentrecord')
```

Blog App URL Configuration

```
python
urlpatterns = patterns('',
    url(r'^$', 'myapp.views.home', name='home'),
    url(r'^admin/doc/', include('django.contrib.admindocs.urls')),
    url(r'^admin/', include(admin.site.urls)),
)
```

7.6.3 Django Case Study – Blogging App

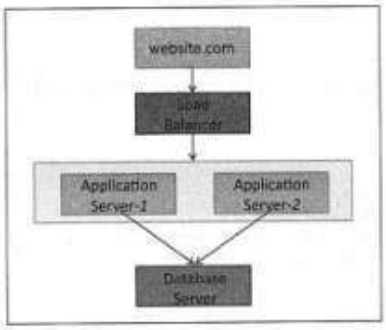


Figure 7.9: Multi-tier deployment architecture for blogging app

Figure 7.9: Multi-tier deployment architecture

- Components:
  - Load Balancer (HAProxy)
  - Application Servers (Django)
  - Database Server (MongoDB)

Blog Model (Box 7.67)

```
python
class Post(models.Model):
    created_at = models.DateTimeField(auto_now_add=True, db_index=True)
    title = models.CharField(max_length=255)
    body = models.TextField()
    def __unicode__(self):
        return self.title
```

URL Patterns (Box 7.68)

```
python
url(r'^$', ListView.as_view(
    queryset=Post.objects.all(),
    context_object_name="posts"),
    name="home"
)
```

Base Template (Box 7.69)

```
html
<title>My Blog Application</title>
{% block page_header %} {% endblock %}
{% block content %} {% endblock %}
```

Posts Template (Box 7.70)

```
html
{% for post in post_list %}
  <h2>{{ post.title }}</h2>
  <p>{{ post.body|truncatewords:20 }}</p>
  {{ post.created_at }}
{% endfor %}
```

Settings (Box 7.71)

```
python
DATABASES = {
  'default': {
    'ENGINE': 'django_mongodb_engine',
    'NAME': 'myapp',
    'HOST': '<mongodb-hostname>',
    'PORT': <mongodb-port>,
  }
}
```

HAProxy Setup (Box 7.72 & 7.73)

```
bash
haproxy -f /etc/haproxy.cfg
python manage.py runserver 0.0.0.0:8000
```

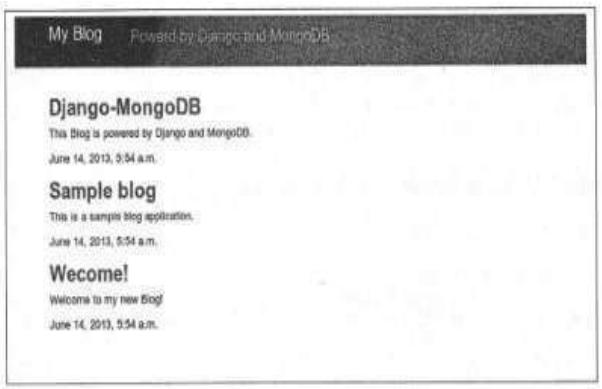


Figure 7.10: Screenshot of the Blog application

Figure 7.10: Blog application interface

7.7 Designing a RESTful Web API

Overview

- RESTful APIs follow the principles of Representational State Transfer (REST).
- Django REST Framework (DRF) simplifies building REST APIs in Django.
- Installation:

```
pip install djangorestframework
pip install markdown
pip install django-filter
```

Project Setup

```
bash
django-admin.py startproject restfulapi
cd restfulapi
python manage.py startapp myapp
```

Book REST API Implementation

Book Model (Box 7.74)

```
python
class Book(models.Model):
    name = models.CharField(max_length=50)
    isbn = models.CharField(max_length=50)
    pub_date = models.DateField()
    pub_name = models.CharField(max_length=50)
```

Views with ViewSet (Box 7.75)

```
python
class BookViewSet(viewsets.ModelViewSet):
    queryset = Book.objects.all()
    serializer_class = BookSerializer
```

Serializers (Box 7.76)

```
python
class BookSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Book
        fields = ['url', 'name', 'isbn', 'pub_date', 'pub_name']
```

URL Configuration (Box 7.77)

```
python
router = routers.DefaultRouter()
router.register(r'books', views.BookViewSet)

urlpatterns = [
    url(r'^', include(router.urls)),
    url(r'^api-auth/', include('rest_framework.urls', namespace='rest_framework')),
    url(r'^admin/', admin.site.urls),
]
```

Django Project Settings

Basic Settings (Box 7.7B)

```
python
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}

REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': ['rest_framework.permissions.IsAdminUser'],
    'PAGINATE_BY': 10
}

INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'books',
)
```

Server Setup

```
python manage.py syncdb
python manage.py runserver
```

Interacting with the API Using CURL

POST Request (Box 7.79)

```
curl -H "Content-Type: application/json" -H "Accept: application/json; indent=4" \
-X POST -d '{"name": "Cloud Computing", "isbn": "0-1988118-1", "date": "2013-12-04",
"pub_name": "XYZ Publishers"}' \
-u admin:admin http://127.0.0.1:8000/books/
```

```
json
{
  "books": "http://localhost:8000/books/"
}
```

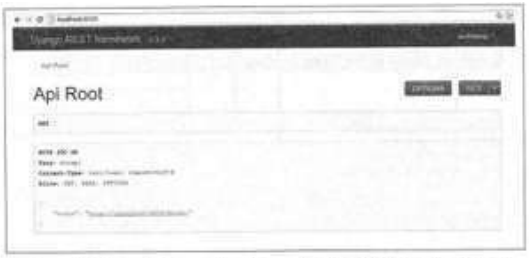


Figure 7.11: Screenshot from the web browsable Books REST API - API root

Figure 7.12: Screenshot from the web browsable Books REST API - viewing a particular book

Figure 7.12: Book Instance Response

```
json
{
  "id": 1,
  "title": "Django for Beginners",
  "author": "William S. Vincent",
  "price": "39.99"
}
```

Additional CURL Examples (Box 7.59)

POST (Create)

```
curl -X POST -H "Content-Type: application/json" -H "Accept: application/json; indent=4" \
-d '{"name":"Cloud Computing: Concepts & Technologies", "isbn":"978-1-49881818",
"pub_date":"2012-12-01", "pub_name":"XYZ Publishers"}' \
http://127.0.0.1:8000/books/1/
```

PUT (Update)

```
curl -X PUT -H "Content-Type: application/json" -H "Accept: application/json; indent=4" \
-d '{"name":"Cloud Computing: Concepts & Technologies", "isbn":"978-1-49881818",
"pub_date":"2012-12-01", "pub_name":"XYZ Publishers"}' \
http://127.0.0.1:8000/books/1/
```

DELETE

```
curl -X DELETE -H "Accept: application/json; indent=4" \
http://127.0.0.1:8000/books/2/
```

8.1 Design Approaches

This section explores application design methodologies tailored to cloud service models: Infrastructure-as-a-Service (IaaS) and Platform-as-a-Service (PaaS).

8.1.1 Design Methodology for IaaS Service Model

Traditional vs. Cloud-Oriented Design

- Traditional designs (e.g., SOA) use multi-tier components (web, app, DB), which are hard to map to cloud architectures.
- Cloud Component Model (CCM) offers a refined approach:
  - Dissects components by function and resource type.
  - Ideal for IaaS environments.

CCM Design Steps

1. **Identify Building Blocks**: Define functional units of the application.
2. **Group Components**: Based on function and required cloud resources.
3. **Define Interactions**:
   - Use asynchronous messaging queues.
   - Employ REST APIs for loosely coupled UI and performance interfaces.

Benefits of CCM

- **Improved Performance**: Loosely coupled components enhance scalability.
- **Reduced Testing & Maintenance Time**: Independent components simplify testing.
- **Lower Costs**: Enables horizontal and vertical scaling; helps isolate performance bottlenecks.

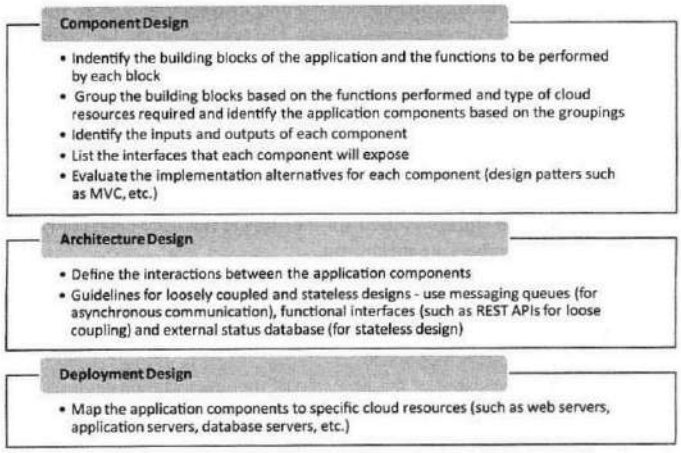


Figure 8.1: Design methodology for IaaS service model (component-based approach)

8.1.2 Design Methodology for PaaS Service Model

Component-Based Design Phases

1. Component Design

- Identify application building blocks and their functions.
- Classify components:
  - Functional (based on grouped functions)
  - Infrastructure (based on platform services)
- Define interfaces between components.
- Evaluate design patterns (e.g., MVC).

2. Architecture Design

- Map components to cloud resources:
  - Web servers
  - Application servers
  - Database servers

3. Deployment Design

- Finalize deployment mapping to cloud infrastructure.

Key Considerations

- PaaS platforms (e.g., Google App Engine, Azure Web Services) offer SDKs and platform-specific features.
- Design must account for:
  - Portability challenges
  - Platform-specific constraints
  - Scalability and flexibility

8.2 Image Processing App

This section describes the design and implementation of a cloud-based image processing application using Django. Users can upload images, apply filters, and download the processed results.

Application Workflow

1. **Image Submission**

html

```
<form action="/demo_app/upload/" method="post" enctype="multipart/form-data">
  <input type="file" name="file">
  <select name="filter">
    <option value="1">gray</option>
    <option value="2">hue</option>
    ...
    <option value="12">enhance</option>
  </select>
  <input type="submit" value="Submit">
</form>
```

2. **Component Design**

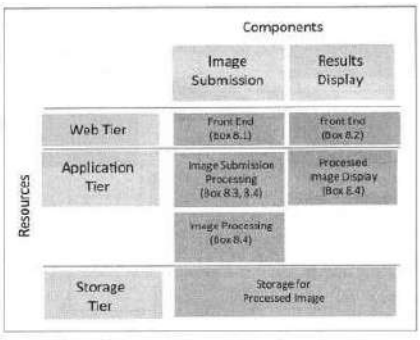


Figure 8.2: Component design for Image Processing App

Figure 8.2: Component design for Image Processing App

  - **Web Tier**: Front-end forms (Box 8.1, 8.2)
  - **Application Tier**: Submission processing, image display, filter logic (Box 8.3, 8.4)
  - **Storage Tier**: Stores processed images

3. **Architecture Design**

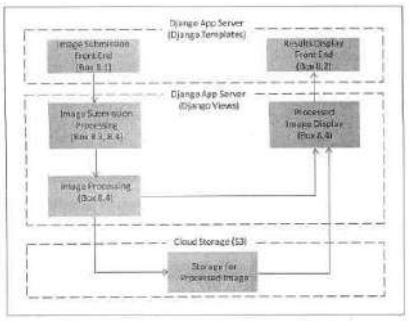


Figure 8.3: Architecture design for Image Processing App

Figure 8.3: Architecture design for Image Processing App

- Django App Server handles templates and views
- Cloud Storage (e.g., S3) stores output images
- Flow: Submission → Processing → Storage → Display

4. **Deployment Design**

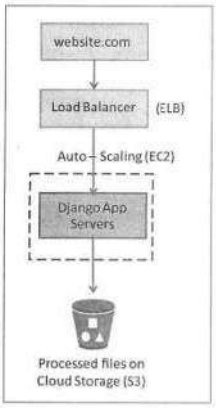


Figure 8.4: Deployment design for Image Processing App

Figure 8.4: Deployment design

- Hosted on cloud infrastructure
- Components:
  - website.com
  - Load Balancer (ELB)
  - Auto Scaling (EC2)
  - Django App Servers
  - Cloud Storage (S3)

User Interface Screenshots

Figure 8.5: Screenshot of Image Processing App - choosing image

Figure 8.5: Choosing image

- Step 1: Upload image
- File selected: lru.jpg
- Navigation: Previous / Next

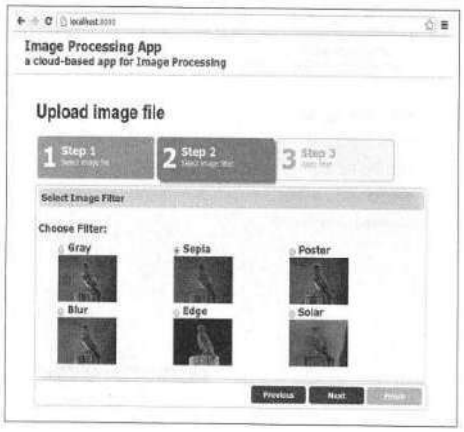


Figure 8.6: Screenshot of Image Processing App - selecting filter

Figure 8.6: Selecting filter

- Step 2: Choose from filters like Gray, Sepia, Poster, Blur, Edge, Solar
- Filter previews shown
- Navigation: Previous / Next

Figure 8.8: Screenshot of Image Processing App - processed image

Figure 8.8: Processed image

- Step 3: Filter applied
- Message: “Image filter has been successfully applied”
- Output: Grayscale image displayed

Front-End Templates

📍 *Box 8.2: Results display front end (Django template)* Includes stylesheets and JavaScript for wizard interface:

html

```
<link href="/static/css/bootstrap.css" rel="stylesheet">
<script src="/static/js/jquery.smartWizard.js"></script>
```

📍 *Wizard UI HTML* Three-step wizard:

html

```
<div id="step-1">Select image file</div>
<div id="step-2">Select image filter</div>
<div id="step-3">Apply filter</div>
```

📍 *Filter Selection HTML*

html

```
<input type="radio" name="preset" value="gray"> Gray
```

```
<img src="/static/images/gray.jpg">
...
<input type="radio" name="preset" value="solar"> Solar
<img src="/static/images/solar.jpg">
```

📍 *Results Display HTML*

```
html
<h4>Image filter has been successfully applied.</h4>
<img src="/static/output/{{outputfilename}}" width="800px">
```

Backend Logic

📍 *Box 8.3: Django form*

```
python
class UploadFileForm(forms.Form):
    myfilefield = forms.FileField()
```

📍 *Box 8.4: Django View*

```
python
def applyfilter(filename, preset):
    if preset == "gray":
        im = ImageOps.grayscale(im)
    ...
    im.save(outputfilename)
```

📍 *Additional Filters*

```
python
if preset == "edge":
    im = ImageOps.grayscale(im)
    im = im.filter(ImageFilter.FIND_EDGES)
if preset == "poster":
    im = ImageOps.posterize(im, 3)
if preset == "solar":
    im = ImageOps.solarize(im, threshold=80)
if preset == "blur":
    im = im.filter(ImageFilter.BLUR)
if preset == "sepia":
    sepia = Image.new("RGB", im.size, (239, 224, 185))
    im = ImageOps.colorize(im.convert("L"), "gray", (239, 224, 185))
    im = Image.blend(sepia, im, 0.5)
```

📍 *File Upload Handler*

```
python
def handle_uploaded_file(request):
    uploadfilename = "media/" + tname
    with open(uploadfilename, 'wb+') as destination:
        for chunk in tchunks:
            destination.write(chunk)
    return applypreset(tname, preset)
```

📍 *Home View*

```
python
def home(request):
    if request.method == "POST":
        form = UploadFileForm(request.POST, request.FILES)
        if form.is_valid():
            preset = request.POST['preset']
```

```
        outputfilename = handle_uploaded_file(request.FILES['myfilefield'], preset)
        return render_to_response('done.html', {'outputfilename': outputfilename})
    else:
        form = UploadFileForm()
    return render_to_response('index.html', {'form': form})
```

📍 *Box 8.5: URL Patterns*

```
python
urlpatterns = patterns('',
    url(r'^$', 'myapp.views.home'),
)
```

8.3 Document Storage App

A cloud-based document storage application built using Django and Amazon S3. Users can register, log in, upload, view, and delete files securely.

Component Design

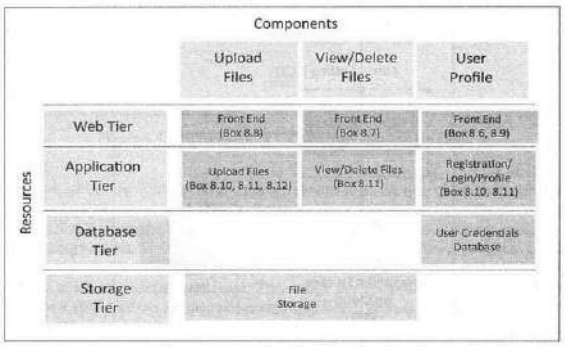


Figure 8.9: Component design for Cloud Drive App

Figure 8.9: Component design for Cloud Drive App

- **Web Tier**:
  - Upload Files (Box 8.10–8.12)
  - View/Delete Files (Box 8.13–8.14)
  - User Profile (Box 8.15–8.16)
- **Application Tier**:
  - Django Views for file handling and user management
- **Database Tier**:
  - MySQL for storing user credentials
- **Storage Tier**:
  - Amazon S3 for storing uploaded documents

Architecture Design

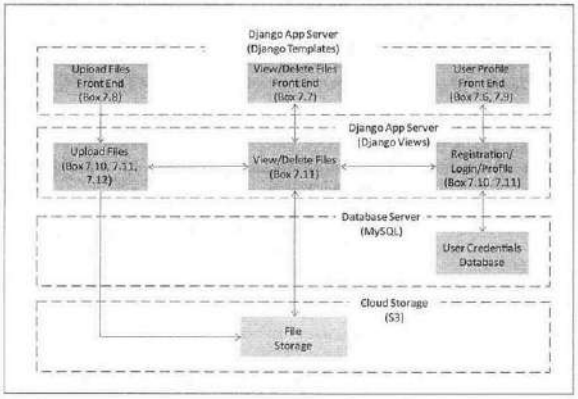


Figure 8.10: Architecture design for Cloud Drive App

Figure 8.10: Architecture design for Cloud Drive App

- Django App Server handles:
  - Upload, View/Delete, and Profile components
- MySQL stores user credentials
- Amazon S3 stores user files

Deployment Design

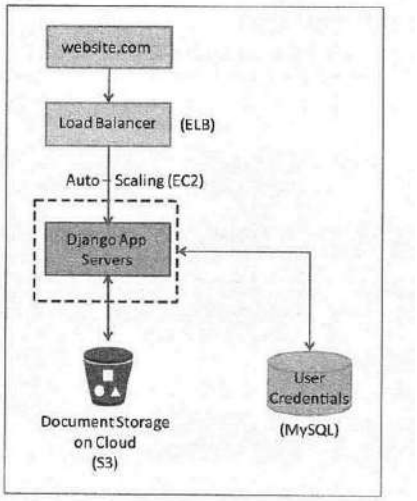


Figure 8.11: Deployment design for Cloud Drive App

Figure 8.11: Deployment design

- Components:
  - website.com
  - Load Balancer (ELB)
  - Auto Scaling (EC2)
  - Django App Servers
  - S3 for file storage
  - MySQL for user data

User Interface Screenshots

Figure 8.12: Screenshot of Cloud Drive App - registration page

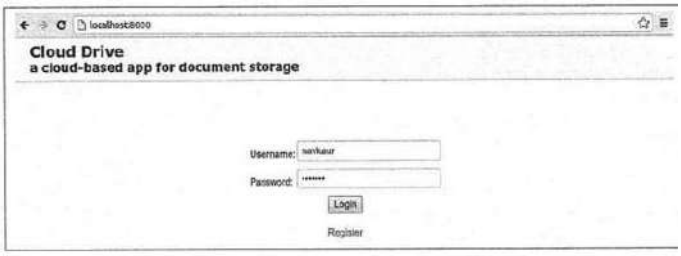


Figure 8.13: Screenshot of Cloud Drive App - login page

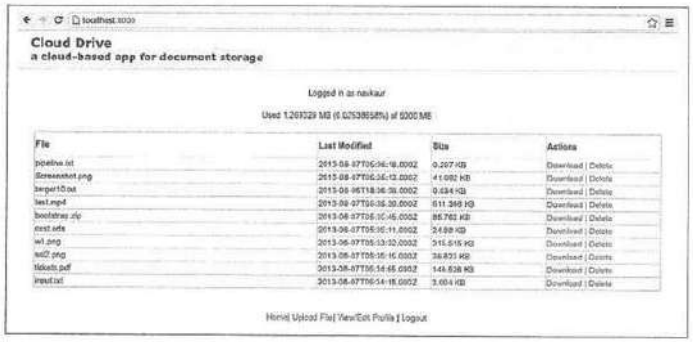


Figure 8.14: Screenshot of Cloud Drive App - uploaded files

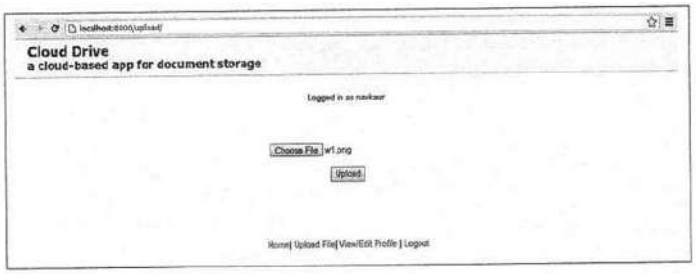


Figure 8.15: Screenshot of Cloud Drive App - file upload page

Figure 8.16: Screenshot of Cloud Drive App - profile page

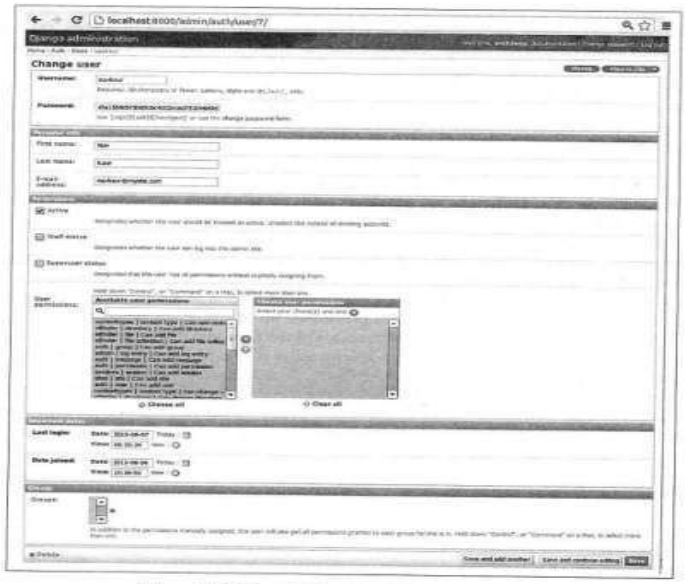


Figure 8.17: Cloud Drive App - Django admin site

Figure 8.12: Registration Page 📍 Figure 8.13: Login Page 📍 Figure 8.14: Uploaded Files View 📍 Figure 8.15: File Upload Page 📍 Figure 8.16: Profile Page 📍 Figure 8.17: Django Admin Site

Django Templates

📍 *Box 8.6: Registration Page Template*

html

```
<form action="" method="post">{% csrf_token %}
  {{ form.as_p }}
  <input type="submit" value="Create the account">
</form>
```

📍 *Box 8.7: View Files Page*

html

```
Logged in as {{username}}
Used: {{total}} MB ({{percentused}}%) of {{limit}} MB
{% for key,val in userfiles.items %}
<tr>
  <td>{{key}}</td>
  <td>{{val.1}}</td>
  <td>{{val.0}}</td>
  <td><a href="/download/{{key}}">Download</a> | <a
href="/delete/{{key}}">Delete</a></td>
</tr>
{% endfor %}
```

📍 *Box 8.8: Upload File Page*

html

```
<form action="/upload/" method="post" enctype="multipart/form-data">
  {% csrf_token %}
  <input type="file" name="filefield" />
  <input type="submit" value="Upload" />
</form>
```

📍 *Box 8.9 & 8.10: Profile Page Template*

html

```
<table>
```

```
<tr><td>Username:</td><td>{{username}}</td></tr>
<tr><td>Email:</td><td>{{email}}</td></tr>
...
<form action="" method="post">{% csrf_token %}
  {{form.as_p}}
  <input type="submit" value="Save">
</form>
```

Django Forms

📍 *Box 8.10: Forms*

```
python
class LoginForm(forms.Form):
    username = forms.CharField()
    password = forms.CharField(widget=forms.PasswordInput())

class UploadFileForm(forms.Form):
    myfile = forms.FileField()
```

Django Views

📍 *Box 8.11–8.12: Views*

```
python
def logout_view(request):
    logout(request)
    return redirect('/')
@login_required
def profile_view(request):
    form = PasswordChangeForm(request.user, data=request.POST)
    ...
    return render_to_response("profile.html", {...})
def register_view(request):
    form = UserCreationForm(request.POST)
    ...
    return render_to_response("register.html", {'form': form})
@login_required
def upload_view(request):
    form = UploadFileForm(request.POST, request.FILES)
    ...
    return render_to_response('index.html', {...})
def login(request):
    user = authenticate(username=username, password=password)
    ...
    return render_to_response('index.html', {...})
```

Amazon S3 Integration

📍 *Box 8.12: S3 Functions*

```
python
def upload_to_s3_bucket_path(bucketname, path, filename):
    key.set_contents_from_filename(fullpathname, cb=percent_cb, num_cb=10)
    key.set_acl('public-read')

def getuserfiles(bucketname, username):
    keys = mybucket.list(username)
    ...
```

```
def delete_from_s3(bucketname, username, filename):
    mybucket.delete_key(username+'/media/'+filename)
```

URL Patterns

📍 *Box 8.13: Basic URLs*

```
python
url(r'^$', 'myapp.views.index'),
url(r'^logout/$', 'myapp.views.logout_view'),
```

📍 *Extended URLs*

```
python
url(r'^register/$', 'myapp.views.register_view'),
url(r'^profile/$', 'myapp.views.profile_view'),
url(r'^upload/$', 'myapp.views.upload_view'),
url(r'^delete/(?P<filename>.*\..*)$', 'myapp.views.delete_view'),
url(r'^process/$', 'myapp.views.process'),
url(r'^admin/', include(admin.site.urls)),
url(r'^password_change/$', 'auth_views.password_change'),
url(r'^password_change/done/$', 'auth_views.password_change_done')
```

MapReduce App :

1. Component Design

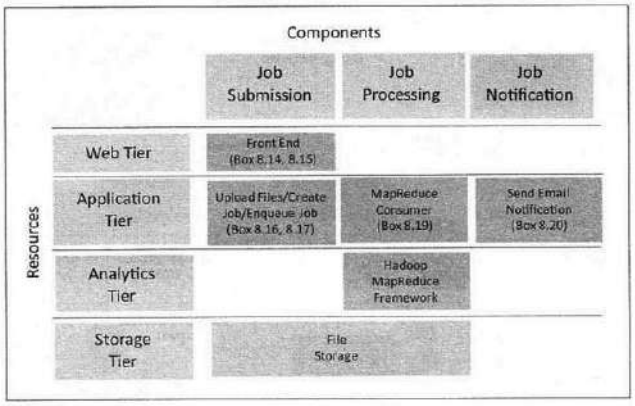


Figure 8.18: Component design for MapReduce App

Job Submission: Front-end UI (Bootstrap, JS) for uploading data and selecting Map/Reduce programs.Job Processing: Django-based application tier handles file uploads, job creation, enqueuing jobs, and execution.Job Notification: Email notifications sent after job completion.
Resources/Tiers:Web Tier → Front-end templates.
Application Tier → Django views, job creation, enqueue logic.
Analytics Tier → Hadoop MapReduce framework.
Storage Tier → Cloud file storage (S3).

2. Architecture Design

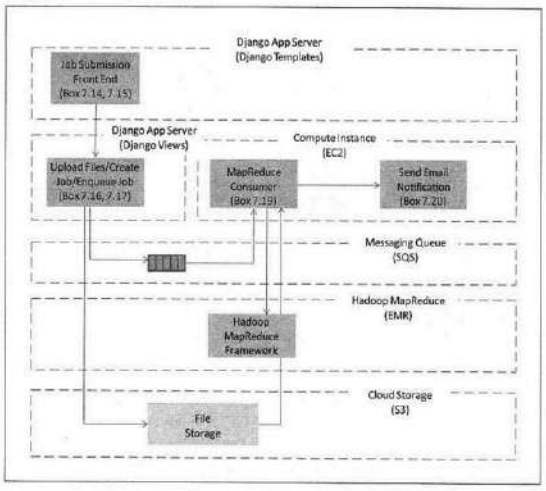


Figure 8.19: Architecture design for MapReduce App

Django App Server:Templates (job submission form, file upload).
Views (handle requests, create jobs).Compute Instance (EC2):
MapReduce Controller (Python scripts).
Email sender.Message Queue (SQS): Stores job requests.
Hadoop MapReduce (EMR): Executes jobs.
Cloud Storage (S3): Stores input/output files.

3.Deployment Design

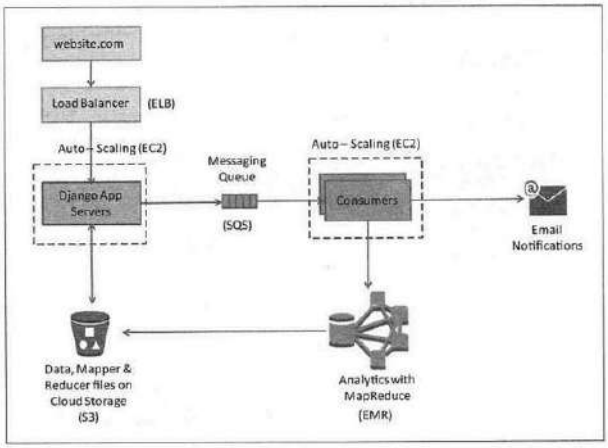


Figure 8.20: Deployment design for MapReduce App

Website + Load Balancer (ELB): Handles user traffic.
Auto-scaling EC2 Instances:
Django App Servers (job submission).
Consumer instances (job execution).
SQS Queue: Connects app servers to consumers.S3 Storage: Holds data, mapper, reducer files.EMR Cluster: Runs MapReduce jobs.
Email Notifications: Sent after job completion.
User Interface Screens
Step 1: Upload data file.

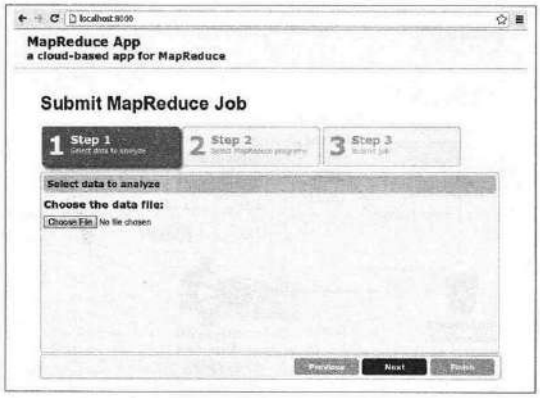


Figure 8.21: Screenshot of MapReduce App - choosing data file

Step 2 : Select predefined/custom Map and Reduce programs.

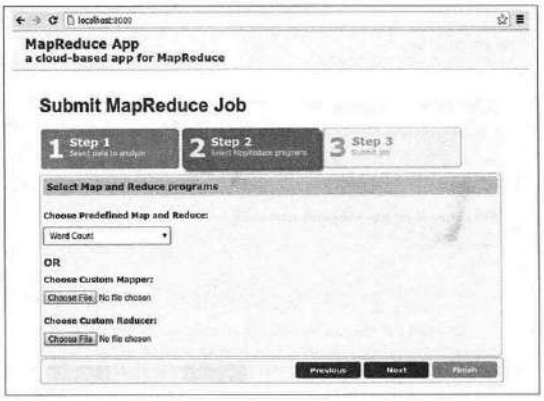


Figure 8.22: Screenshot of MapReduce App - selecting mapper and reducer

Step 3 : Enter email for notifications, submit job.

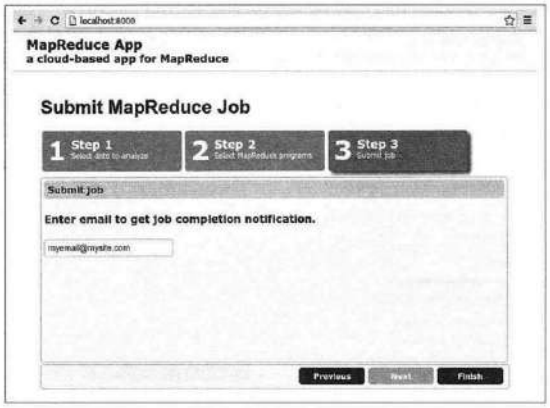


Figure 8.23: Screenshot of MapReduce App - submitting job

Confirmation : Shows job details (email, data file, mapper, reducer).

Figure 8.24: Screenshot of MapReduce App - job confirmation

Notification : Email with download link for results.

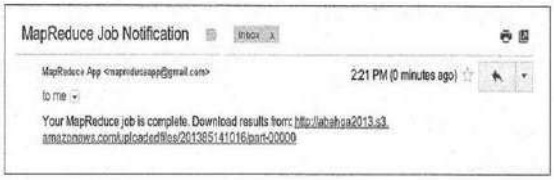


Figure 8.25: MapReduce job completion notification email

Django Forms
UploadFileForm → Handles data file upload.
UploadMapForm → Handles mapper file upload.
UploadReduceForm → Handles reducer file upload.
Django Views
home() function:
Handles POST requests.
Saves uploaded files.
Calls createJob() → builds job definition.
Calls enqueueJob() → pushes job to SQS.
URL Patterns:
Root (^$) mapped to mapreduce.views.home.
Job Creation & Queue Handling
createJob():
Builds S3 paths for data, mapper, reducer.
Supports predefined programs (WordCount, InvertedIndex).
Calls enqueueJob().
enqueueJob():
Connects to SQS.
Serializes job request (datafile, mapper, reducer, email).
Writes message to queue.
Consumer Program:
Reads jobs from SQS (FIFO).
Submits to EMR.
Waits for completion.
Calls sendnotification().
EMR Job Execution
createemrjob():
Connects to EMR.
Creates streaming step (mapper, reducer, input, output).
Runs job flow.

Polls job status until completion.
Generates download link (S3 output).
Sends notification email.
Email Notification (Box 8.20 & Figure 8.25)
SMTP (Gmail) or Amazon SNS used.
Sends job completion message with download link.
Example notification:
Subject: *MapReduce Job Notification*
Body: *Your job is complete. Download results from [S3 link]*.

Social Media Analytics App

- Purpose: Collect Twitter feeds in real-time on specific keywords.
- Analyze sentiment (positive, negative, neutral).
- Provide aggregate results via a dashboard.
- Built using **Tweepy**, **AWS (SQS, EC2, S3)**, **MongoDB**, and **Django**.

2. Component Design

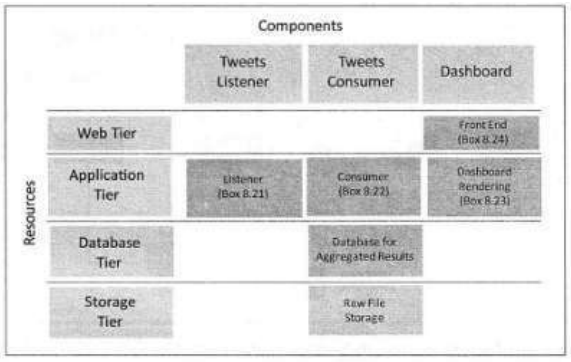


Figure 8.26: Component design for Social Media Analytics App

Web Tier: Front-end dashboard (Django templates).
Application Tier:

- Listener → Collects tweets via Twitter API.
- Consumer → Processes tweets, performs sentiment analysis.
- Dashboard Rendering → Displays results.

Database Tier: MongoDB for aggregated results.
Storage Tier: S3 for raw tweet storage.

3. Architecture Design

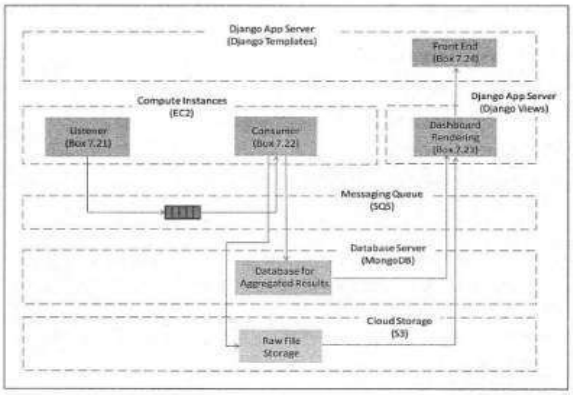


Figure 8.27: Architecture design for Social Media Analytics App

Django App Server → Templates & Views.
EC2 Instances:

- Listener (Box 8.21).

- Consumer (Box 8.22).
- Dashboard Rendering (Box 8.23).

SQS Queue → Connects Listener and Consumer.
MongoDB → Stores aggregated results.
S3 → Stores raw tweet data.

4. Deployment Design

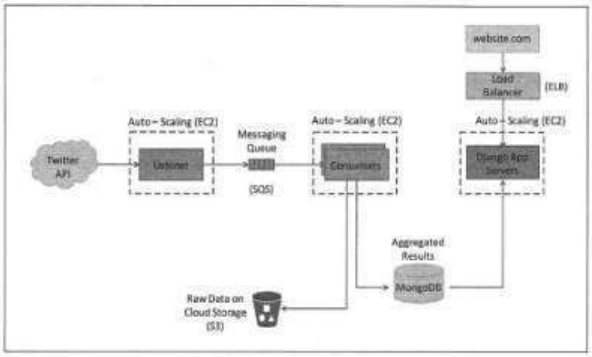


Figure 8.28: Deployment design for Social Media Analytics App

Twitter API → Feeds into Listener (EC2).
Listener → Pushes tweets into SQS.
Consumers (EC2) → Pull tweets from SQS, analyze sentiment, store results in MongoDB.
Web App (Django) → Accessed via ELB, displays results.
S3 → Stores raw tweet data.

5. Listener Component

- Uses **Tweepy** to stream tweets.
- Configured with keywords (e.g., "India").
- On receiving a tweet:
  - Serializes JSON data.
  - Pushes to SQS queue.

6. Consumer Component

- Runs separately from Listener.
- Retrieves tweets from SQS.
- Functions:
  - **parseTweet()** → Extracts fields (text, retweets, hashtags, time).
  - **findSentiment()** → Uses AFINN lexicon (2500 words with sentiment scores).
  - **analyzeTweet()** → Aggregates sentiment counts, hashtags, hourly activity.
- Stores aggregated results in MongoDB.

7. Data Aggregation & Storage

- Sentiment scores: Positive, Negative, Neutral.
- Hourly activity: Tweets grouped by hour.
- Top hashtags: Extracted and ranked.
- Top tweets: Based on retweet counts.
- MongoDB document structure:
  - _id → Date + keyword.
  - Metadata → Date, keyword.
  - Aggregated results → Sentiment counts, hashtags, top tweets.

8. Django Views Connects to MongoDB.

- Retrieves metadata for today's keyword.
- Computes percentages for sentiment distribution.
- Extracts top hashtags and calculates their relative percentages.

- Passes data to template for rendering.

9. Django Templates

- Bootstrap-based UI.
- Displays:
    - Total tweets.
    - Positive, negative, neutral counts & percentages.
    - Hourly activity chart.
    - Top tweets list.
    - Top hashtags.

10. Visualization

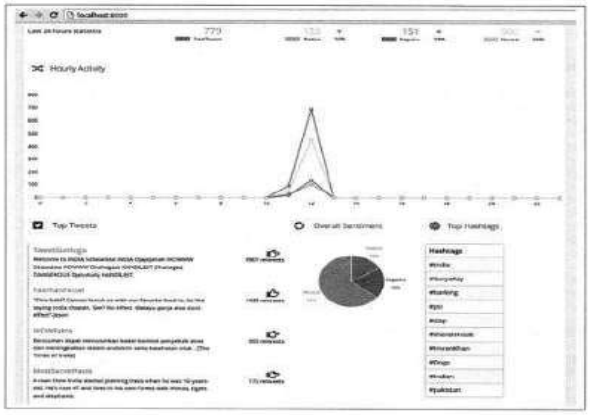


Figure 8.29: Screenshot of Social Media Analytics App

Pie Chart

- Positive, Negative, Neutral distribution.
- Hover shows percentages.

Hourly Activity Chart

- Line chart showing tweet activity over time.

Dashboard Metrics:

- Total tweets, retweets, replies, likes.
- Sentiment breakdown.
- Top hashtags and tweets.

UNIT-1V
INTRODUCTION TO BIG DATA

Introduction:

Big data refers to collections of data sets with high **volume**, **velocity**, and **variety**.

Examples of big data sources:

- Social networks (text, images, audio, video)
- Click-stream data from web applications (e.g., e-Commerce)
- Machine sensor data from industrial and energy systems
- Healthcare data from electronic health record (EHR) systems
- Logs from web applications
- Stock market data

Characteristics:

- **Volume**: Massive scale, difficult to store/process with traditional methods.
- **Velocity**: Speed at which data is generated; modern systems produce data rapidly.
- **Variety**: Different forms (structured/unstructured) — text, image, audio, video, sensor data.

Big data analytics involves: data cleansing, data munging, processing, visualization.

Enabled by: cloud computing, distributed parallel processing frameworks.

9.2 Clustering Big Data

Clustering: Grouping similar data items together.

Useful when data items are not in one cluster.

Applications:

- Finding similar users in social network data
- Grouping similar patients in EHR data
- Grouping similar items in stock market data
- Clustering market research data to group similar customers
- Clustering clickstream data to group similar users

Clustering is achieved by clustering algorithms that belong to a broad category of algorithms called **unsupervised machine learning**. Unsupervised machine learning algorithms find the patterns and hidden structure in data for which no training data is available. Let us now look at some popular clustering algorithms and their applications to big data.

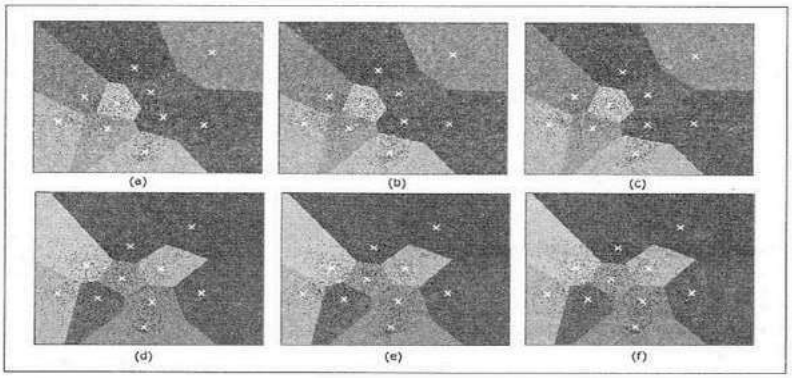


Figure 9.1: Example of clustering 300 points with k-means: (a) iteration 1, (b) iteration 2, (c) iteration 3, (d) iteration 5, (e) iteration 10, (f) iteration 100.

An above example of clustering 300 data points. The centroid points are recomputed after each iteration and as seen in this figure there is little movement of centroids after 10 iterations.

There are various distance measures that can be used for clustering algorithms including:

- **Euclidean distance measure**: This is the simplest of all distance measures. The Euclidean distance between points p and q in N-dimensional space is given as:

$$d(p,q)=\sum_{i=1}^{N}(p_i-q_i)^2 \quad (9.1)$$

- **Cosine distance measure**: Cosine distance measure finds the cosine of angle between two vectors (vectors drawn from origin to the points).

$$d=\cos(\theta)=A\cdot B/\|A\|\,\|B\| \quad (9.2)$$

- **Manhattan distance measure**: Manhattan distance measure is the sum of the absolute differences of the coordinates of two points given as:

$$d(p,q)=\sum_{i=1}^{N}|p_i-q_i| \quad (9.3)$$

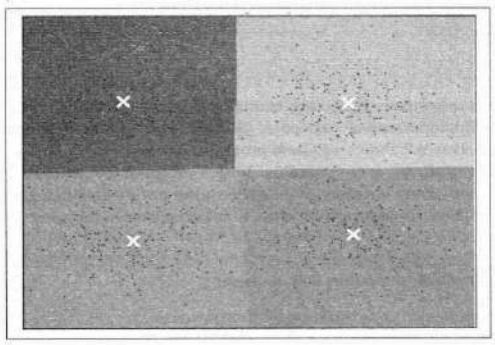


Figure 9.2: Clusters generated using k-means clustering

DBSCAN clustering

DBSCAN (density-based spatial clustering of applications with noise) is another algorithm of interest; See Ester et. al. [67]. DBSCAN is a density clustering algorithm which works based on the notions of **density reachability** and **density connectivity**.

- **Density reachability** is defined on the basis of *Eps-neighborhood*. For any point p in a cluster C, there is a point q so that p is inside the Eps-neighborhood of q and there are at least a minimum number (*MinPts*) of points in the Eps-neighborhood of that point.

- A point p is called **directly density-reachable** from a point q if it is not further away than a given distance *Eps* and if it is surrounded by at least *MinPts* points that may be considered part of a cluster.

- A point p is **density-reachable** from q if there is a chain of points p1,…,pn, with p1=q, pn=p, such that pi+1 is directly density-reachable from pi.

- A point p is **density-connected** to a point q if there is a point o such that both p and q are density-reachable from o with respect to *Eps* and *MinPts*.

A cluster is then defined based on the following two properties:

- **Maximality**: For all points p,q, if p belongs to cluster C and q is density-reachable from p (w.r.t. *Eps* and *MinPts*), then q also belongs to cluster C.

- **Connectivity**: For all points p,q in cluster C, p is density-connected to q (w.r.t. *Eps* and *MinPts*).

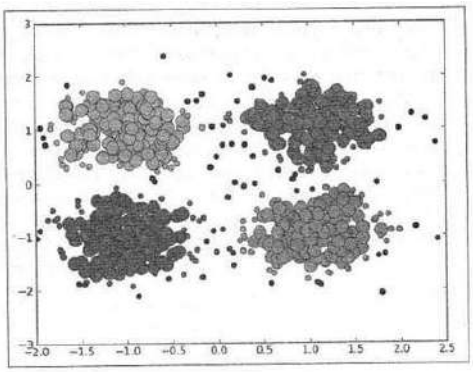


Figure 9.3: Clusters generated using DBSCAN clustering

9.2.3 Parallelizing Clustering Algorithms using MapReduce

- Clustering algorithms (like **k-means**) can be parallelized using **MapReduce** when datasets are too large for a single machine's memory.
- Data is stored in a **distributed file system (HDFS)**, split into blocks, and replicated across nodes.
- Clustering starts with an **initial set of centroids**.
- A **client program** controls the clustering process.
- **Map Phase:** Each data sample is compared to centroids; assigned to the nearest one.
- **Reduce Phase:** Centroids are recomputed as the mean of all points in each cluster.
- New centroids are sent back to the client to check for **convergence**.
- If convergence hasn't occurred, the process repeats until either convergence or the maximum number of iterations is reached.

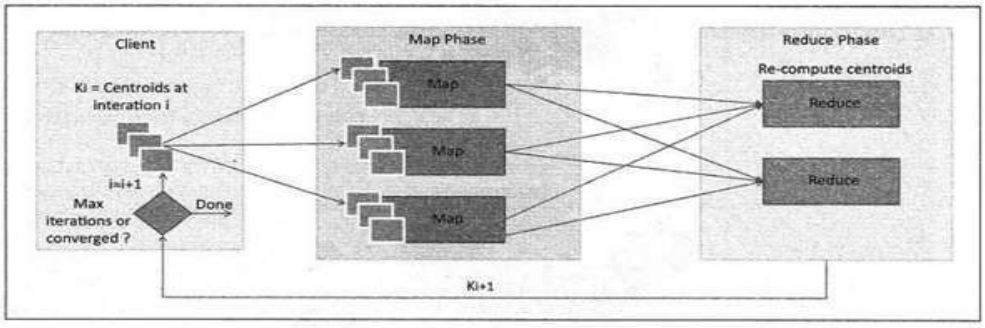


Figure 9.5: Parallel implementation of k-means clustering with MapReduce

9.3 Classification of Big Data

Classification is a key technique in **supervised machine learning**.It involves assigning data samples into predefined categories or classes.

Types of Classification

Binary Classification: Two possible classes (e.g., spam vs. not spam).

Multi-class Classification: More than two classes (e.g., classifying animals into cats, dogs, birds).

Document Classification: Categorizing documents based on content (e.g., news articles into politics, sports, entertainment).

Performance Metrics

1. **Precision**

   Precision=TruePositive/(TruePositive+FalsePositive)

   Measures how many predicted positives are actually correct.
2. **Recall**

   Recall=TruePositive(TruePositive+FalseNegative)

   Measures how many actual positives were correctly identified.
3. **Accuracy**

   Accuracy=(TruePositive+TrueNegative)/(TruePositive+TrueNegative+FalsePositive+FalseNegative)

   Measures overall correctness of predictions.
4. **F1-score**

   F1-score=2(Precision)(Recall)/(Precision+Recall)

   Harmonic mean of precision and recall, balancing both.

9.3.1 Naive Bayes

Naive Bayes is a probabilistic classification algorithm based on **Bayes' theorem** with a naive assumption about the independence of feature attributes.

Given a class variable C and feature variables F1,…,Fn, the conditional probability (posterior) according to Bayes' theorem is:

$P(C/F1,\ldots,Fn)=P(F1,\ldots,Fn/C)\cdot P(C)/P(F1,\ldots,Fn)$

P(C/F1,…,Fn): Posterior probability

P(F1,…,Fn/C): Likelihood

P(C): Prior probability

P(F1,…,Fn): Evidence

Naive Bayes assumes independence among features:

$$P(F1,\ldots,Fn/C)=\prod i=1nP(Fi/C)$$

Since the evidence P(F1,…,Fn) is constant for a given input and does not depend on C, only the numerator matters for classification:

$$P(C/F1,\ldots,Fn)\propto P(C)\prod i=1nP(Fi/C)$$

Thus, classification is performed as:

$$C=\text{argmax}c\in CP(C)\prod i=1nP(Fi|C)$$

Gaussian Naive Bayes

Assumes the likelihood P(F1,…,Fn|C) as:

$$P(F1,\ldots,Fn|C)=\prod iP(Fi|C)(9.18)$$

For each feature:

$$P(Fi|C)=12\pi\sigma 2\exp(-(Fi-\mu C)22\sigma 2)(9.19)$$

Here, μC is the mean and σC is the standard deviation for values in Fi in class C.

Suitable for problems where features are **continuous values** assumed to follow a **Gaussian (normal) distribution**.

Multinomial Naive Bayes

Uses a **multinomial distribution** for each feature variable.

Suitable for problems with **discrete features**, such as **document classification** (e.g., word counts in text).

Bernoulli Naive Bayes

Also suitable for problems with **discrete features**.

The likelihood is defined as:

$$P(Fi|C)=P(Fi|C)Fi\cdot(1-P(Fi|C))(1-Fi)(9.20)$$

Works well when features are **binary-valued** (e.g., presence or absence of a word in a document).

Decision Trees

Decision Trees are a supervised learning method used to predict the value of a target variable based on several attribute variables.They split data recursively based on attributes to maximize **information gain** and reduce uncertainty.

Mathematical Definitions

1. **Information Gain**

$$I(X)=-\sum P(x)\log 2P(x)(9.17)$$

2. **Entropy**

$$H(X)=E[I(X)]=E[-\log 2P(X)]=-\sum \log 2P(x)(9.18)$$

3. **Gini Coefficient**

$$G(X)=1-\sum P(x)2(9.19)$$

- Attributes must be **discrete**. If not, discretize continuous attributes.
- Calculate the **entropy** of every attribute using the dataset.
- Choose the attribute with the **highest information gain**.
- Create branches for each value of the selected attribute.
- Repeat the process with the remaining attributes until the tree is complete

Random Forest

Random Forest is an ensemble learning method based on randomized decision trees [72].

- It trains a number of decision trees.
- The final output is determined by taking the majority vote, using the mode of the class predicted by the individual trees.

Support Vector Machine

Support Vector Machine (SVM) is a supervised machine learning approach used for classification and regression.

- In its basic form, SVM is a binary classifier that classifies data points into one of two classes .
- SVM training involves determining the maximum margin hyperplane that separates the two classes.
- The maximum margin hyperplane is the one with the largest separation from the nearest training data point.

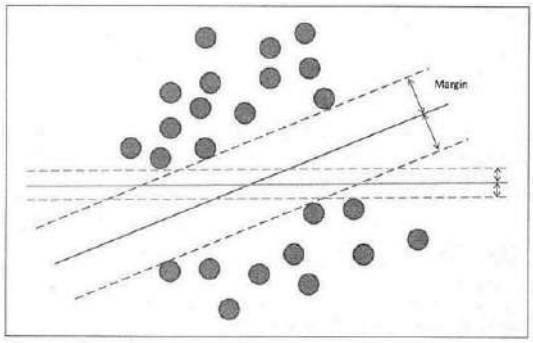


Figure 9.10: Margins for an SVM

Given a training dataset (xi,yi):

- xi is an n-dimensional vector.
- yi=1 if xi is in class 1.
- yi=−1 if xi is in class 2.

A standard SVM finds a hyperplane.

Multimedia Cloud

Introduction

With the development of Web 2.0 and the increasing reach of high-speed internet to wireless applications, multimedia-rich web applications have become widely popular in recent years. There are various types of multimedia web applications including:

- Multimedia storage
- Processing
- Transcoding
- Streaming applications

Because multimedia applications require high resources, cloud computing is proving to be an efficient and cost-effective solution.

With the growing demand for multimedia-rich web applications on wireless platforms, a new paradigm of multimedia cloud is emerging. This provides multimedia storage, processing, and streaming services to millions of mobile users worldwide.

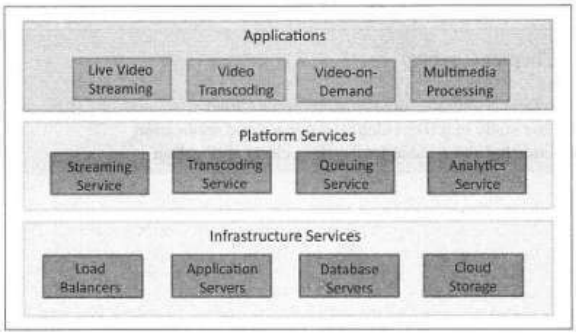


Figure 10.1: Multimedia Cloud reference architecture

Case Study: Cloud-Based Live Video Streaming App

1. Workflow Overview

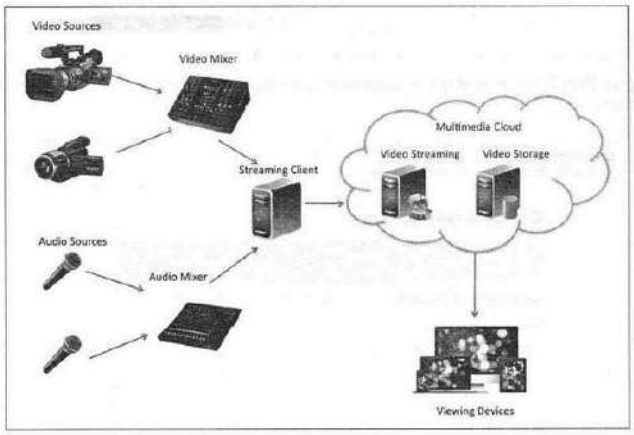


Figure 10.2: Workflow for live video streaming using multimedia cloud

Sources: Video cameras and microphones capture raw feeds.
Mixers: Video and audio mixers combine multiple feeds.
Streaming Client: Encodes the mixed feeds and sends them to the cloud.
Multimedia Cloud:

- Video Streaming Service → broadcasts live streams to viewers.
- Video Storage Service → archives streams for later playback.

Viewing Devices: Laptops, desktops, tablets, smartphones, smart TVs.

Key Point: Cloud enables on-demand creation of streaming instances, scalable broadcasting, and archiving.

2.Stream Creation – Step 1

Figure 10.3: Screenshot of live video streaming application showing step-1 (entering stream details)

User enters stream details:

- Stream Name (e.g., *Cloud Computing*)
- Description (e.g., *Live Stream Session on Cloud Computing*)

- Navigation buttons: Back, Next, Cancel.

Purpose: Define metadata for the stream.

3. Stream Creation – Step 2

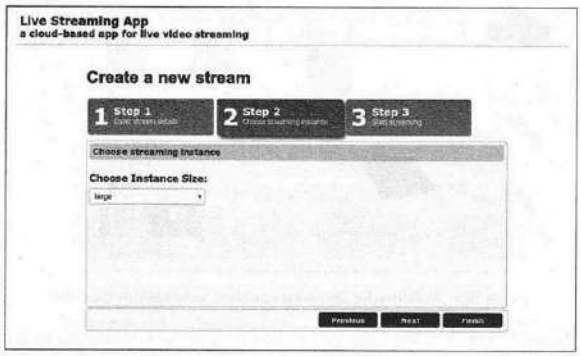


Figure 10.4: Screenshot of live video streaming application showing step-2 (choosing stream instance)

- User chooses instance size (cloud resources allocated for streaming).
- Navigation buttons: Back, Next, Cancel.

Purpose: Select appropriate computing power for the stream.

4. Stream Creation – Step 3

Figure 10.5: Screenshot of live video streaming application showing step-3 (starting streaming instance)

Instruction: *Click finish to start the streaming instance on cloud.*

- Buttons: Back, Next, Finish, Cancel.

Purpose: Launch the streaming instance in the cloud.

5. Streaming Instance Created

Figure 10.6: Screenshot of live video streaming application showing details of streaming instance

- Confirmation screen shows:
  - Stream Title
  - Stream URL (RTMP link)
  - Stream ID
  - Description
- Instructions for encoder setup (Adobe Flash Media Encoder):
  - Use provided Stream URL and Stream ID.
  - Start encoding to push video to the cloud.

Purpose: Provide technical details for connecting encoder → cloud → viewers.

6. Encoder Setup

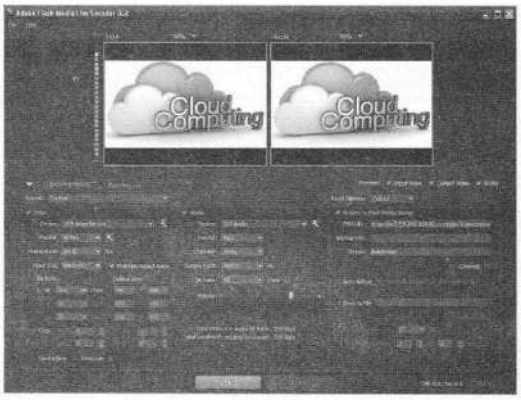


Figure 10.7: Screenshot of Adobe Flash Media Live Encoder used for video streaming

- Screenshot of Adobe Flash Media Live Encoder interface.
- Preview windows show video feed.
- Settings for video/audio encoding: format, bitrate, input devices, stream settings.

Purpose: Configure encoding parameters before sending stream to cloud.

7. Streaming Page

Figure 10.8: Screenshot of live video streaming application showing video streaming page

- Web interface of the Live Streaming App.
- Displays live video (e.g., *Introduction to Cloud Computing*).
- Description area explains the broadcast.

Purpose: Viewer-facing page for consuming the live stream.

Key Takeaways

- The app is built with Django framework and uses Amazon EC2 instances.
- Adobe Flash Media Encoder handles video encoding.
- Workflow: Sources → Mixers → Client → Cloud → Streaming/Storage → Devices.
- Cloud enables scalability, archiving, and global accessibility.
- The UI guides users through 3 steps: stream details → instance size → launch.
- Once launched, encoder settings connect the stream to the cloud for live broadcast.

Streaming Protocols Overview

Streaming protocols are methods used to deliver audio, video, and data over the internet. They differ in transport mechanisms, encryption, and adaptability to bandwidth.

RTMP (Real-Time Messaging Protocol)

Purpose: Streams audio, video, and data between Adobe Flash technologies (Flash Player, AIR).

Transport: Works over TCP; maintains persistent connections for low-latency communication.

Features:

- Bidirectional multiplexing (multiple streams over one connection).
- Chunking mechanism for bandwidth/latency optimization.

- Fragments media into 2-second segments for smooth playback.
- Preserves stream order with timestamps and metadata.

Variations:

- **RTMPS:** Secure RTMP over TLS/SSL.
- **RTMPE:** Encrypted RTMP.
- **RTMFP (P2P):** Peer-to-peer live video delivery.
- **RTMFP (Multicast Fusion):** Combines P2P + multicast for higher QoS.
- **RTMP Dynamic Streaming (Unicast):** Adaptive bitrate, low latency.
- **RTMFP (Multicast):** IP multicast with encryption.

HTTP Live Streaming (HLS)

Proposed by: Apple (part of iOS ecosystem).

Mechanism:

- Uses extended **M3U playlist** to list media segments.
- Server divides streams into segments; client fetches and plays them sequentially.

Features:

- Adaptive bitrate (adjusts quality to bandwidth).
- Supports AES-128 encryption (optional).
- Widely used for iOS and HLS-compatible devices.

HTTP Dynamic Streaming (HDS)

Developed by: Adobe.

Mechanism:

- Combines HTTP progressive download + RTMP streaming.
- Delivers MP4 media (H.264 or VP6) over HTTP.

Features:

- Adaptive bitrate (detects client bandwidth/resources).
- Supports HD video up to 1080p.
- Bitrates: 700 kbps – 6 Mbps+.
- Audio codecs: AAC, MP3.
- Leverages CDNs, caching, and standard HTTP servers.

Protected HDS (PHDS): Adds real-time encryption.

Key Comparisons:

Protocol	Transport	Adaptivity	Encryption	Typical Use
RTMP	TCP	Yes (Dynamic Streaming)	RTMPE, RTMPS	Flash-based live/on-demand streaming
HLS	HTTP	Yes	AES-128 optional	iOS devices, modern streaming apps
HDS	HTTP + RTMP	Yes	PHDS	Adobe ecosystem, HD adaptive streaming

Case Study: Video Transcoding App

Platform: Built using **Amazon Elastic Transcoder**, Python, and Django.

Purpose: Converts video files into formats suitable for playback on mobile devices, tablets, and PCs.

Workflow: Users upload video files → choose transcoding presets → submit job → monitor job status → download transcoded file.

Application Workflow (UI Steps)

1. **Step 1 – Choose Video File**
   - User selects a video file to upload.
   - File is stored in an **Amazon S3 bucket**.
2. **Step 2 – Specify Transcoding Options**
   - Dropdown menu with presets (e.g., Generic 1080p, 720p, 480p, 360p, 240p, 144p, Web).
   - Presets define output resolution and format.
3. **Step 3 – Start Transcoding**
   - User clicks **Finish** to submit the transcoding job.
   - Job is queued in Amazon Elastic Transcoder.

Django Templates & Code

- Home Page Template
- Uses **jQuery Smart Wizard** for step-by-step input.
- Form submission triggers transcoding job creation.

File Submission Confirmation

- Displays message: *"File has been successfully queued for transcoding."*
- Shows **Job ID** and link to job status page.

Job Status Page

- Displays job details:
  - Job ID
  - Status (e.g., queued, processing, complete)
  - Input file
  - Output file
  - Download link
- If status = complete → shows confirmation message.

Key Features

- **Cloud-based scalability:** Uses Amazon Elastic Transcoder for handling multiple jobs.
- **User-friendly UI:** Step-by-step wizard for file upload and preset selection.
- **Integration with Django:** Templates handle job submission, confirmation, and status tracking.
- **Dynamic job monitoring:** Job status page updates with progress and provides download link.

Summary

The Video Transcoding App demonstrates:

- How cloud services (Amazon Elastic Transcoder + S3) integrate with web frameworks (Django).
- A practical workflow for video processing: **Upload → Configure → Submit → Track → Download**.
- Use of HTML, JavaScript (jQuery), and Django templates to build a smooth user interface.

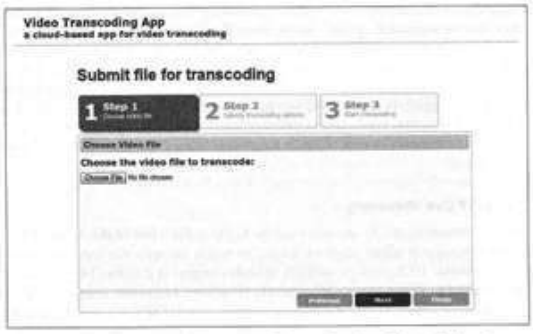


Figure 10.9: Screenshot of video transcoding app showing video uploading form

Figure 10.10: Screenshot of video transcoding app showing preset options

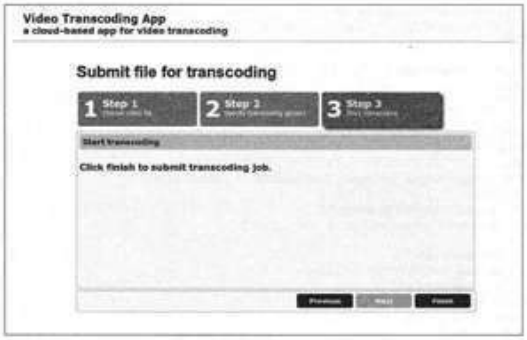


Figure 10.11: Screenshot of video transcoding app showing job submission form

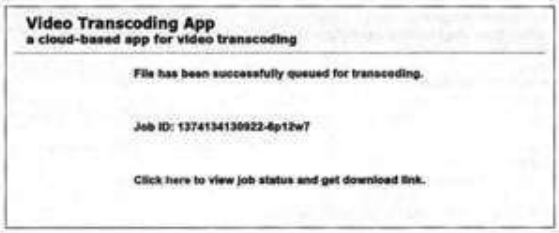


Figure 10.12: Screenshot of video transcoding app showing submitted job

Cloud Application Benchmarking and Tuning:

Introduction:

- Cloud-based multi-tier applications face rapidly changing workloads.
- Proper resource provisioning and capacity planning are required to meet performance and cost goals.
- Over-provisioning helps maintain performance by enabling dynamic scaling.
- Performance testing helps identify bottlenecks before deployment.
- Bottlenecks can be resolved by scaling up (more powerful resources) or scaling out (more instances).
- Synthetic workloads are used to simulate real user behavior for testing.

Benchmarking of cloud applications is important for the following reasons:

- Supports capacity planning and provisioning.
- Helps choose cost-effective deployment architectures.
- Ensures optimal resource utilization (avoids under/over-provisioning).
- Ensures market readiness by testing application under different workload types.
- Helps manage seasonal workload variations.

The steps involved in benchmarking of cloud applications are described below.

11.1.1 Trace Collection/Generation

The first step in benchmarking cloud applications is to collect/generate traces of real application workloads. For generating a trace of workload, the application is instrumented to log information such as the requests submitted by the users, the time-stamps of the requests, etc.

11.1.2 Workload Modeling

Workload modeling involves generation of mathematical models that can be used for generating synthetic workloads. Workloads of applications are often recorded as traces of workload related events such as arrival of requests along with the time-stamps, details about the requesting the service, etc. Analysis of such traces can provide insights into the workload characteristics which can be used for formulating mathematical models for the workloads.

11.1.3 Workload Specification

Since the workload models of each class of cloud computing applications can have different workload attributes, a Workload Specification Language (WSL) is often used for specifying the workload attributes that are critical to the performance of the application. WSL can be used by synthetic workload generators for generating workloads with slightly varying characteristics. This can be used to perform sensitivity analysis of the application performance to the workload attributes by generating synthetic workloads.

11.1.4 Synthetic Workload Generation

Synthetic workloads are used for benchmarking cloud applications. An important requirement for a synthetic workload generator is that the generated workloads should represent the real workloads and should preserve the important characteristics of real workloads such as inter-session and inter-session intervals, etc. There are two approaches to synthetic workload generation:

- **Empirical approach:** In this approach, traces of applications are sampled and replayed to generate the synthetic workloads.
- **Analytical approach:** This approach uses mathematical models to define the workload characteristics that are used by a synthetic workload generator.

11.1.5User Emulation vs Aggregate Workloads

User Emulation

- Mimics real user behavior
- Includes think time and dependencies
- Cannot control exact request arrival times

Aggregate Workload

- Controls exact arrival times
- No individual user behavior
- Cannot model request dependencies well

11.2 Workload Characteristics

Each class of multi-tier applications have their own characteristic workloads. A successful performance evaluation methodology should be able to accurately model the

application workloads. Workload modeling involves creation of mathematical models that can be used for generation of synthetic workloads. Characteristics of application workloads include:

- **Session:** A set of successive requests submitted by a user constitute a session.
- **Inter-Session Interval:** Inter-session interval is the time interval between successive sessions.
- **Think Time:** In session, a user submits a series of requests in succession. The time interval between two successive requests is called think time. Think time is the inactive period between successive requests in a session. It is the time taken by the user to review the response of a request and decide what the next request should be.
- **Session Length:** The number of requests submitted by a user in a session is called the session length.
- **Workload Mix:** Workload mix defines the transition between different pages in an application and the proportion in which the pages are visited. Multi-tier cloud applications can experience varying changes in their workloads. Workload characteristics of an application may also vary based on the kind of users interacting with the application. An e-Commerce application, for example, can experience database write-intensive workloads during the times of a year when a larger number of users are purchasing products online. Whereas, the same application can be read-intensive at workload mix when users are only browsing for products.

11.3 Application Performance Metrics

The most commonly used performance metrics for cloud applications are as follows:
Response Time Response time is the time interval between the moment when the user submits a request to the application and the moment when the user receives a response. Response time includes various components as given in the following equation:

$$R=D/B+T_{RTT}+T_S+T_C$$

- D: Amount of data transfer required to serve the user request
- B: Minimum bandwidth across all links in the network from the user to the application deployment
- T_{RTT}: Round trip time for user-application interactions needed to generate the response
- T_S: Total processing time required by all tiers of the application deployment
- T_C: Total processing time required by the user's device

Throughput Throughput is the number of requests that can be serviced per second. If an application receives N requests per second and only (N−M) can be serviced in a second, then M requests will wait in a queue.
Performance requirements are typically specified as a series of service level objectives (SLOs) that define response time or throughput requirements for each request in the application.

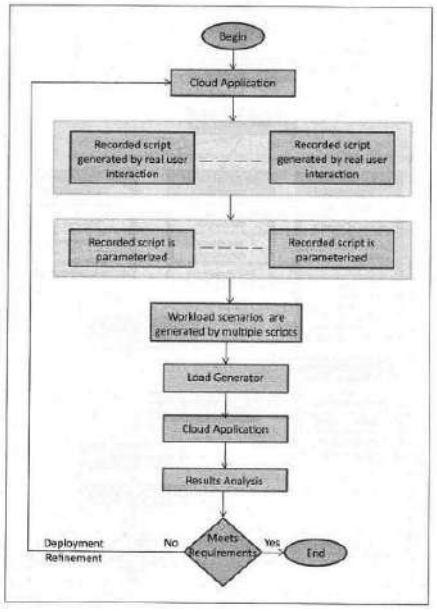
Design Considerations for Benchmarking Methodology

- **Accuracy:**
  - Synthetic workloads must mimic realistic workloads.
  - Aggregated metrics (e.g., average response time) may be insufficient; detailed workload characteristics are needed.
- **Ease of Use:**
  - Should minimize manual coding effort.
  - Must account for dependencies between requests.
- **Flexibility:**

- Fine-grained control over workload attributes (think time, session length, workload mix).
- Enables sensitivity analysis by varying one parameter at a time.

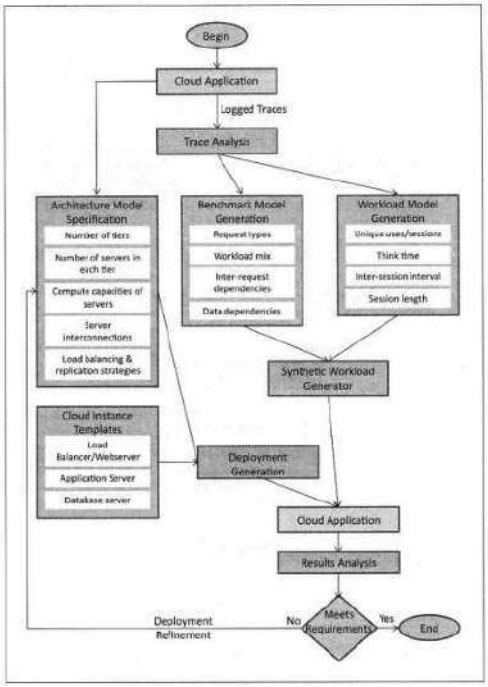
Benchmarking Tools

- **Traditional Tools:** httperf, SURGE, SWAT, HP LoadRunner, SPECweb99.
- **Approaches:**
  - **Traditional**
    - Record user interactions as scripts.
    - Parameterize scripts to simulate anomalies.
    - Execute scripts from load generation machines.
    - Limitations: Time-consuming, less accurate.



Traditional performance evaluation workflow based on a semi-automated approach. A real user has to first manually interact with the application to record scripts and then parameterize recorded scripts to generate scripts for creating virtual users.

Automated :



Automated Performance Evaluation Workflow Used by Recent Tools (e.g., GT-CAT)

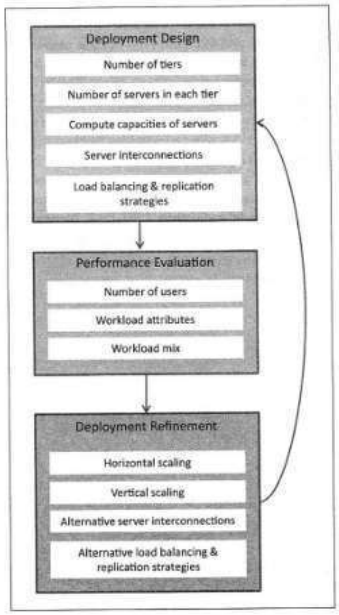
- Use traces (sessions, request times, DB queries).
- Build workload models to generate synthetic loads.
- Captures user behavior more accurately.
- Faster and higher accuracy compared to traditional methods.

11.5.1Types of Tests :

- **Baseline Tests:**
  - Collect performance metrics (response time, throughput, CPU/memory/network usage).
  - Usually with small user counts (1–100).
  - Used to compare performance before/after changes.
- **Load Tests:**
  - Evaluate performance under expected production workloads.
- **Stress Tests:**
  - Push application beyond peak load.
  - Identify failure conditions and warning metrics.
- **Soak Tests:**
  - Run application under fixed workload for long periods.
  - Assess stability and performance degradation over time.

11.6 Deployment Prototyping

From the standpoint of a user, the cloud computing resources should look limitless, however due to complex dependencies that exist between servers in various tiers, applications can experience performance bottlenecks. Deployment prototyping can help in making deployment architecture design choices. By comparing performance of alternative deployment architectures, deployment prototyping can help in choosing the best and most cost-effective deployment architecture that can meet the application performance requirements. Since multi-tier applications can experience seasonal variations in workloads, deployment prototyping may need to be done on a regular basis to ensure the market readiness of such applications at all times.



Steps involved in deployment prototyping for a cloud application

- **Deployment Design:** Create the deployment with various tiers as specified in the deployment configuration and deploy the application.
- **Performance Evaluation:** Verify whether the application meets the performance requirements with the deployment.
- **Deployment Refinement:** Deployments are refined based on the performance evaluations. Various alternatives can exist in this step such as vertical scaling, horizontal scaling, for instance.

11.7 Load Testing & Bottleneck Detection Case Study

This section provides a case study of load testing and benchmarking an application with a multi-tier cloud deployment as shown in **Figure 11.4**. For this case study, the **Rice University Bidding System (RUBiS)** benchmark is used. RUBiS is an auction site prototype loosely modeled after eBay.

- RUBiS PHP application is deployed on the application servers.
- A MySQL database is set up on the database server.
- **collectd** is used to measure system statistics.
    - collectd is a daemon that collects system performance statistics periodically.
    - Stores values in formats such as RRD files, CSV files.
    - Provides plugins for monitoring CPU, Disk, Memory, Network, etc.
- The collected statistics are used to identify performance bottlenecks in the system.

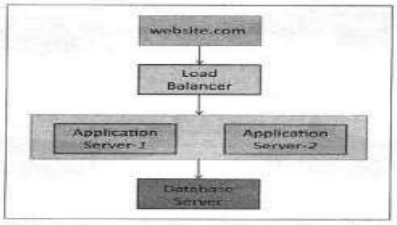
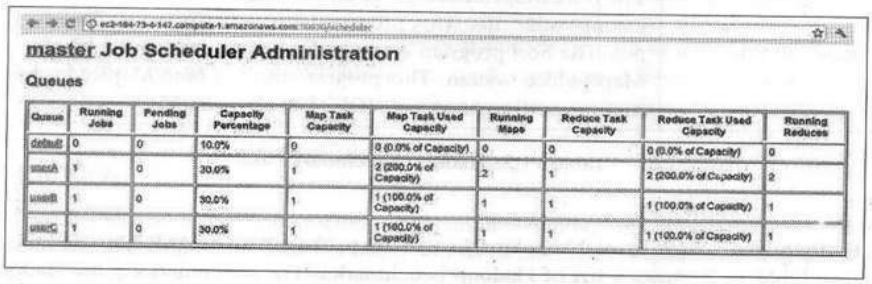


Figure 11.4: Multi-tier deployment architecture for sample application for case study

11.8 Hadoop Benchmarking casestudy

With the Capacity Scheduler configured, the Hadoop cluster is launched. The Capacity Scheduler administration page can be viewed at the URL:
http://JobTracker-URL:50030/scheduler, as shown in



master Job Scheduler Administration

Queues

Queue	Running Jobs	Pending Jobs	Capacity Percentage	Map Task Capacity	Map Task Used Capacity	Running Maps	Reduce Task Capacity	Reduce Task Used Capacity	Running Reduces
default	0	0	10.0%	0	0 (0.0% of Capacity)	0	0	0 (0.0% of Capacity)	0
userA	1	0	30.0%	1	2 (200.0% of Capacity)	2	1	2 (200.0% of Capacity)	2
userB	1	0	30.0%	1	1 (100.0% of Capacity)	1	1	1 (100.0% of Capacity)	1
userC	1	0	30.0%	1	1 (100.0% of Capacity)	1	1	1 (100.0% of Capacity)	1

Figure 11.6: Hadoop Capacity Scheduler administration page

Hadoop Capacity Scheduler administration page

The administration page displays a table under **master Job Scheduler Administration** with columns such as:

- Queue
- Capacity
- Used Capacity
- Running Tasks
- Active Users
- Max Capacity
- State
- Jobs Accepted
- Jobs Completed
- Jobs Failed
- Jobs Killed

The table shows data for queues **A, B, and C**, including capacity percentages, number of running tasks, active users, and job statistics.

The administration page of the Capacity Scheduler shows:

- Queues and their allocated capacities
- Map and reduce task capacities
- Number of running map and reduce tasks

For monitoring the performance of the nodes in the Hadoop cluster, performance monitoring tools such as collectd or mon can be used. For the experiments shown in this section, the collectd tool is used.

Benchmark	Description
TestDFSIO	TestDFSIO tests the I/O performance of HDFS. This benchmark is useful for stress testing HDFS. TestDFSIO uses a MapReduce job to read or write files in parallel. Each file is read or written in a separate map task, and the output of the map is used for collecting statistics relating to the file just processed. The reduce phase accumulates the statistics to produce a summary.
MRBench	MRBench runs a small job a number of times. MRBench is a complimentary benchmark to the Sort benchmark. MRBench checks whether small job runs are responsive and running efficiently on the cluster.
NNBench	NNBench is useful for load testing the NameNode hardware and configuration. This benchmark generates a lot of HDFS-related requests with normally very small payloads for the sole purpose of putting a high HDFS management stress on the NameNode. The benchmark can simulate requests for creating, reading, renaming and deleting files on HDFS.
Gridmix	GridMix is a suite of benchmarks that model that characteristics of realistic workloads for submitting a mix of synthetic jobs. GridMix requires a MapReduce job trace describing the job mix for a given cluster, and input data from which the synthetic jobs will read bytes.
Sort	The Sort MapReduce program is included in the hadoop-*-examples.jar file. This program does a partial sort of its input. The Sort program is useful for benchmarking the whole MapReduce system. This program can test both MapReduce runtime performance and HDFS I/O performance.

Table 11.2: Hadoop benchmarks

PART-1: Cloud Security

12.1 Introduction:

More and more organizations are moving their applications and associated data to the cloud to reduce costs and reduce the operational and maintenance overheads, and one of the predominant considerations is that of security of the data in the cloud. Most cloud service providers implement advanced security features similar to those in on-premise environments. However, due to the outsourced nature of the cloud, resource pooling and multi-tenanted architectures, security remains an important concern in adoption of cloud computing. In addition to the traditional vulnerabilities that exist for web applications, cloud applications have additional vulnerabilities because of the shared usage of resources and virtualized resources.

Key security challenges for cloud applications include:

1. Authentication

- Confirms digital identity of the entity requesting access.
- Traditional IT: controlled internally, usually employees only.
- Cloud: accessed over the internet → more complex.
- Requires involvement of cloud provider's systems and services.

2. Authorization

- Defines access rights to protected resources via policies.
- Traditional IT: organization controls and modifies policies easily.
- Cloud: access policies depend on cloud provider's services.

3. Security of Data at Rest

- Cloud servers host applications from multiple organizations side-by-side.
- Increases complexity of securing data.
- Requires **isolation mechanisms** to separate data/applications of different organizations.

4. Security of Data in Motion

- Traditional IT: data exchange stays within organization boundaries → full visibility.
- Cloud: data moves in/out over the internet.
- Requires **strong security mechanisms** to protect data while being transmitted.

5.Data Integrity

Data integrity ensures that the data is not altered in an unauthorized manner after it is created, transmitted, or stored.

- In cloud computing environments, where data storage is outsourced, ensuring integrity is critical.
- Mechanisms are required to detect accidental or intentional changes in data.

6.Auditing

Auditing is very important for applications deployed in cloud environments.

- In traditional in-house IT, organizations have full visibility of applications and protected information.
- In cloud applications, auditing mechanisms are needed to provide visibility into:
    - Application data

- Access patterns
- Modifications performed by users (including mobile devices like laptops and smartphones).

12.2 CSA Cloud Security Architecture:

The Cloud Security Alliance (CSA) provides a Trusted Cloud Initiative (TCI) Reference Architecture [46].

- It is a methodology and set of tools for developers and security architects.
- Helps assess whether internal IT and cloud providers adhere to security objectives.
- Provides a roadmap to meet business security needs.

Security Risk Management (SRM) Domain within TCI Reference Architecture:

- Core component of an organization's information security program.
- Safeguards assets and data.
- Assesses and prioritizes risks in operating activities.
- Figure 12.1 shows the SRM domain.

Sub-domains of SRM:

1. **Governance, Risk Management, and Compliance**
   - Structures, processes, and controls for effective governance, risk management, and compliance.
2. **Information Security Management**
   - Measurements such as maturity models, benchmarking, security architectures, and roadmaps.
   - Helps assess priorities and minimize/eliminate threats and risks.
3. **Privilege Management Infrastructure**
   - Ensures users have required access and privileges.
   - Uses Identity and Access Management (IAM): identity management, authentication, authorization, accountability.
4. **Threat and Vulnerability Management**
   - Addresses vulnerability management, threat management, compliance testing, and penetration testing.
5. **Infrastructure Protection Services**
   - The objective of this sub-domain is to secure Server, End-Point, Network and Application layers.
6. **Data Protection**
   - This sub-domain deals with data lifecycle management, data leakage prevention, intellectual property protection with digital rights management, and cryptographic services such as key management and PKI/symmetric encryption.
7. **Policies and Standards**
   - Security policies and standards are derived from risk-based business requirements and exist at a number of different levels including Information Security policy, Physical Security Policy, Business Continuity Policy, Infrastructure Security Policies, Application Security Policies as well as the over-arching Business Operational Risk Management Policy.

12.3 Authentication

Authentication refers to confirming the digital identity of the entity requesting access to some protected information. The process of authentication involves, but is not limited to, validating at least one factor of identification of the entity to be authenticated. A factor can be something the entity or the user knows (passcode or pin), something the user has (such as a smart card), or something that can uniquely identify the user (such as fingerprints). In multifactor authentication more than one of these factors are used for authentication. In this section you will learn about authentication mechanisms such as SSO, SAML-Token, OTP, etc.

Single Sign-On (SSO)

- User logs in once and gains access to multiple systems.
- Avoids repeated login prompts.
- Translates credentials across different systems.
- Benefits:
    - Saves time
    - Reduces password fatigue
    - Minimizes human error

SAML (Security Assertion Markup Language)

- XML-based open standard.
- Exchanges authentication and authorization data.
- Works between:Identity Provider (IdP)
  Service Provider (SP)
- Uses SAML tokens.

o Security features: SSL encryption

- Digital signatures
- Prevents replay and man-in-the-middle attacks

Kerberos

- Developed at MIT.
- Ticket-based authentication protocol.
- Works over insecure networks.
- Provides **mutual authentication** (client and server verify each other).
- Uses tickets instead of sending passwords over network.

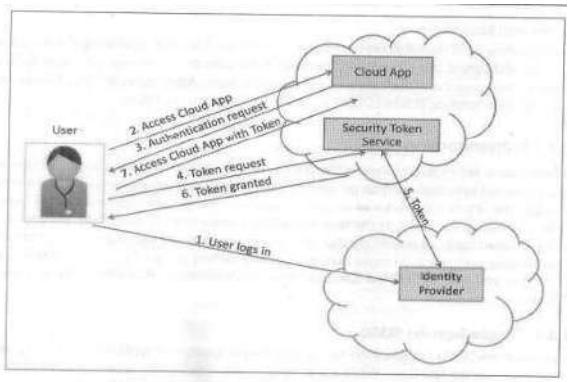


Figure 12.2: SAML-token based SSO authentication

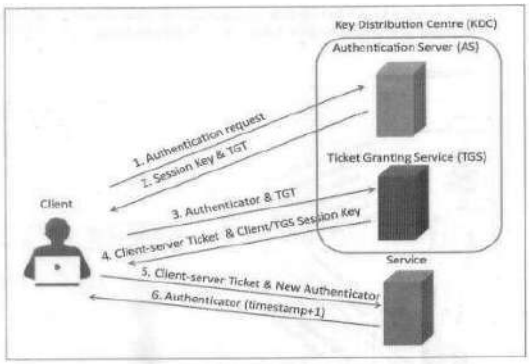


Figure 12.3: Kerberos authentication flow

12.4 Authorization

OAuth is an open standard for authorization that allows resource owners to share their private resources stored on one site with another site without handing out the credentials [49, 50]. OAuth 1.0 protocol was published as an RFC in 2010 and the OAuth 2.0 framework was published in 2012. OAuth 2.0 is not backward compatible with OAuth 1.0. In the OAuth model, an application (which is not the resource owner) requests access to resources permitted by the resource owner (but hosted by the server). Resource owner grants permission to access the resources in the form of a token and the shared-secret. Tokens make it unnecessary for the resource owner to share its credentials with the application. Tokens can be issued with a restricted scope and limited lifetime, and revoked independently.

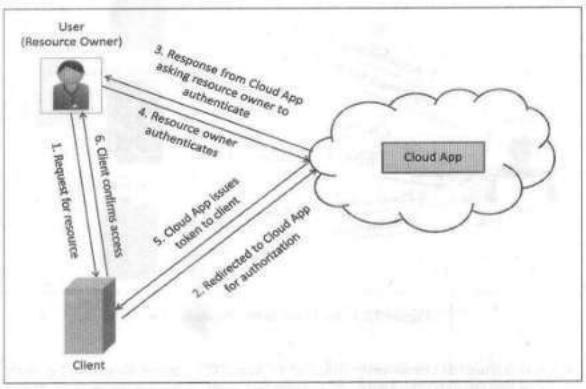


Figure 12.4: OAuth authorization flow

12.5 Identity & Access Management

Step 1: Identity Management

- Provides methods to digitally identify users.
- Maintains user identity attributes across multiple organizations.
- Ensures consistent identification of persons.

Step 2: Access Management

- Deals with user privileges.
- Controls what resources a user can access.
- Works along with identity management.

Step 3: Identity & Access Management (IAM)

- Combines:
  - User identity
  - Authentication
  - Authorization
  - Access policies
- Ensures secure access to systems and data.

Step 4: Federated Identity Management

- Allows users from one domain to access systems of another domain.
- No need to maintain separate identity information for each domain.
- Enables seamless cross-domain access.

Step 5: How Federation is Enabled

- Uses Single Sign-On (SSO) mechanisms.
- Technologies used:
  - SAML Token
  - Kerberos
- Identity credentials remain with a trusted Identity Provider (IdP).
- Multiple organizations can use the same credentials for authentication.

Step 6: Standardized Access Control Policies

- Ensure data confidentiality.
- Define how access is granted or restricted.

Step 7: Role-Based Access Control (RBAC)

- Access is assigned based on roles, not individuals.
- Different roles for different departments.
- Example:
    - HR role
    - Finance role
    - Admin role

Step 8: RBAC in Cloud Environment

1. User requests access to cloud application data.
2. User sends request to the System Administrator.
3. Administrator assigns:
    - User roles
    - Access permissions
4. Information is stored in:
    - User Roles database
    - Data Access Policies database
5. System grants access based on:
    - Assigned role
    - Defined policies

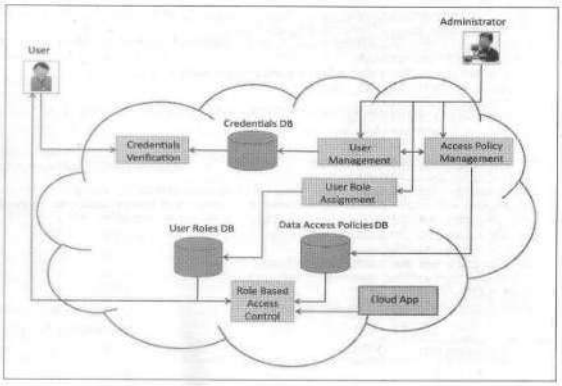


Figure 12.5: Role-based access control in the cloud

12.6 Data Security

Securing data in the cloud is critical for cloud applications as the data flows from applications to storage and vice versa. Cloud applications deal with both data at rest and data in motion. There are various types of threats that can exist for data in the cloud such as denial of service, replay attacks, man-in-the-middle attacks, unauthorized access/modification, etc.

12.6.1 Securing Data at Rest

Data at rest is the data that is stored in database in the form of tables/records, files on a file server or raw data on a distributed storage or storage area network (SAN). Data at rest is secured by encryption. Encryption is the process of converting data from its original form (i.e.,

plaintext) to a scrambled form (ciphertext) that is unintelligible. Decryption converts data from ciphertext to plaintext. Encryption can be of two types:

Symmetric Encryption (symmetric-key algorithms)

Symmetric encryption uses the same secret key for both encryption and decryption. The secret key is shared between the sender and the receiver. Symmetric encryption is best suited for securing data at rest since the data is accessed by known entities from known locations. Popular symmetric encryption algorithms include:

- **Advanced Encryption Standard (AES):** AES is the data encryption standard established by the U.S. National Institute of Standards and Technology (NIST) in 2001. AES uses Rijndael cipher (developed by Joan Daemen and Vincent Rijmen)
- **Twofish:** Twofish is a symmetric key block cipher with a block size of 128 bits and key sizes up to 256 bits. Twofish was one of the top candidates for the AES contest.
- **Blowfish:** Blowfish has a 64-bit block size and a variable key length from 32 bits up to 448 bits.
- **Triple Data Encryption Standard (3DES):** This algorithm is a variation of Data Encryption Standard (DES) developed by IBM. 3DES uses a key-bundle comprising three keys each of 56 bits. In the first step, DES is used to encrypt plaintext using the first key, then the data is decrypted using the second key and finally, the third key is used to encrypt the data using DES.
- **Serpent:** Serpent is a symmetric key block cipher that uses a block size of 128 bits and supports a key size of 128, 192 or 256 bits. Serpent was one of the top candidates for AES contest and came second.
- **RC6:** RC6 is a symmetric key block cipher designed by RSA Security and was a candidate for the AES contest. RC6 uses a block size of 128 bits and supports key sizes of 128, 192 and 256 bits.
- **MARS:** MARS is a block cipher that was designed by IBM as a candidate for the AES contest. It uses a 128-bit block size and a variable key size of between 128 and 448 bits.

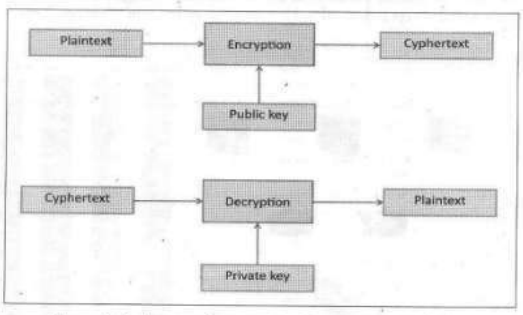


Figure 12.6: Asymmetric encryption using public/private keys

Asymmetric Encryption (public-key algorithms)

Asymmetric encryption uses two keys, one for encryption (public key) and the other for decryption (private key). The two keys are linked to each other such that one key encrypts plaintext to ciphertext and the other decrypts ciphertext back to plaintext. The public key can be

shared or published while the private key is known only to the user. Figure 12.6 shows the asymmetric encryption approach.
Asymmetric encryption is best suited for securing data that is exchanged between two parties, where symmetric encryption can be unsafe because the secret key has to be exchanged between the parties and anyone who manages to obtain the secret key can decrypt the data. In asymmetric encryption, a separate key is used for decryption which is kept private.

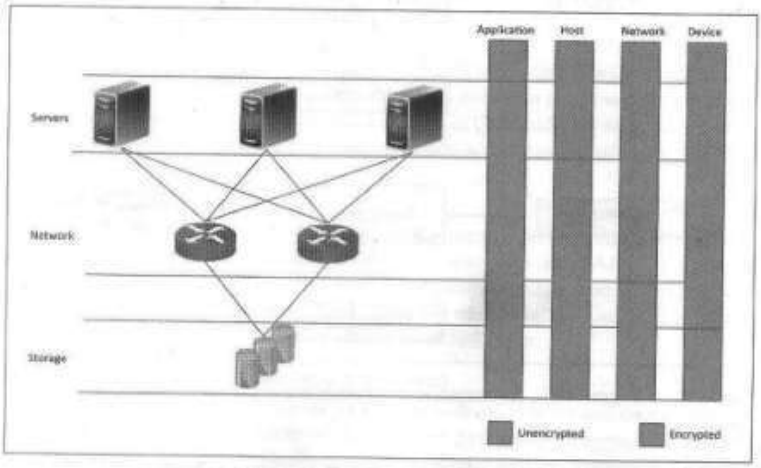


Figure 12.7: Encryption levels

Application

Application-level encryption involves encrypting application data right at the point where it originates, i.e., within the application. Application-level encryption provides security at the level of both the operating system and from other applications. Therefore, one application cannot decrypt data of another application. An application encrypts all data generated in the application before it flows to the lower levels and presents decrypted data to the user.
The advantage of application-level encryption is that it provides security against operating system and network attacks and also data theft. However, key management is a challenging task for application-level encryption. Keys can be stored either in memory, a file, or on a separate key server. The application performance is affected in case of key rotation, where the application reads and decrypts the data using an old key and then encrypts the data using the new key, while it is processing other requests.

Host

In host-level encryption, encryption is performed at the file level for all applications running on the host. Host-level encryption can be done in software, in which case additional computational resources are required for encryption, or it can be performed with specialized hardware such as a cryptographic accelerator card.
The advantage of host-level encryption is that it is highly secure and well-suited for active data files across all applications running on a host. However, like application-level encryption, key management can be challenging. Keys are stored in the host memory or a separate key server.

Network

Network-level encryption is best suited for cases where the threats to data are at the network or storage level and not at the application or host level. Network-level encryption is performed when moving the data from a creation point to its destination using a specialized hardware that encrypts all incoming data, in real-time. The application and host levels remain unencrypted. Network-level encryption using the operating system implemented.

The advantage of this network-level encryption is that it is simple to implement and requires no changes in the existing data infrastructure. Keys are managed in the software. However, the disadvantage of this encryption level is that it is the least scalable and is the slowest. As the data volume increases, a single encryption appliance can become a bottleneck.

Device

Device-level encryption is performed on a disk controller or a storage server. Device-level encryption is easy to implement and is cost-effective for cases where the primary concern about data security is protecting data stored on a single node. Device-level encryption is operating system, application, host, and even transport independent. Encryption is performed in hardware in the device. Device-level encryption is the fastest encryption level and is infrastructure. The disadvantage of this method is that all data that is transmitted to and from the storage module is unencrypted.

12.6.2 Securing Data in Motion

Securing data in motion, when the data flows between a client and a server over a potentially insecure network, is important to ensure data confidentiality and integrity.

- **Data confidentiality** means limiting access to data so that only authorized recipients can access it.
- **Data integrity** means that the data is not changed when moving from sender to receiver.

Data transmitted over unsecured networks can be captured in an unmodified manner and later reused, or the data may be altered during transmission.

Transport Layer Security (TLS) and Secure Socket Layer (SSL) are the mechanisms used for securing data in motion. Though TLS and SSL are different (SSL is the predecessor of TLS), TLS 1.0 resembles SSL 3.0 closely. Both TLS and SSL are used to encrypt web traffic using Hypertext Transfer Protocol (HTTP). When HTTP is upgraded with TLS/SSL, it is conveniently called **HTTPS**. TLS/SSL is also used for confidentiality and message authentication codes for message integrity.

Figure 12.8 shows the TLS handshake in more detail. The TLS handshake protocol is composed of the following steps:

- **ClientHello:** The client sends a ClientHello message specifying the highest TLS protocol version it supports, a random number, and the information on the private key encryption algorithms supported.

- **ServerHello:** The server replies with the ServerHello message, containing the protocol version chosen, a random number, and the settings of the private key encryption algorithm.
- **Server Certificate:** The server sends its Certificate to the client.
- **ServerHelloDone:** The server sends a ServerHelloDone message indicating to the client that it has completed the handshake negotiation.
- **Client Certificate:** The client sends its Certificate to the server.
- **ClientKeyExchange:** The client creates a PreMaster secret (session key) encrypted with the server's public key (which is contained in the server's certificate) and sends the encrypted session key to the server.
- **CertificateVerify:** The client sends a CertificateVerify message, which is a digital signature over the previous handshake messages using the client's certificate's private key. The digital signature can be verified by using the client's certificate's public key.
- **ChangeCipherSpec:** Client and server communicate to each other that the data that will be exchanged from now on will be encrypted with the session key previously exchanged.
- **Finished:** The client sends an encrypted message (containing a hash and MAC over the previous handshake messages) indicating the end of the handshake session. The server responds with its Finished message. At this point, the handshake phase is complete and the client and the server use the session key to encrypt and decrypt the data that they mutually exchange to validate integrity.

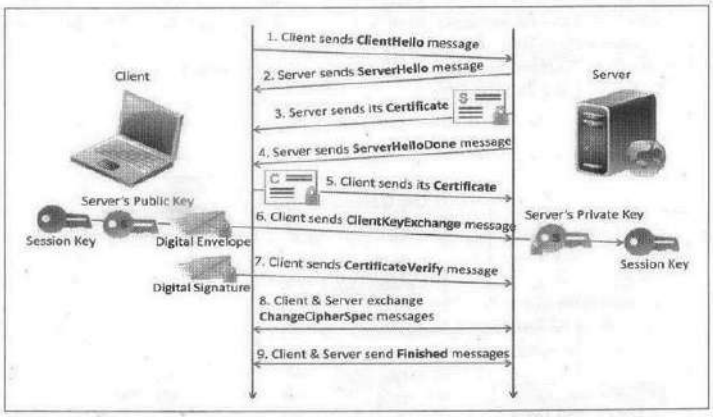


Figure 12.8: TLS Handshake

TLS/SSL use Message Authentication Codes (MAC) MACs detect both accidental or deliberate modifications in the data. A MAC is a cryptographic checksum on the data that provides assurance that the data has not changed. Computation of MAC involves:

1. A secret key known only to the party that generates the MAC and the intended recipient.
2. The data on which the MAC is computed.

12.7 Key Management

Management of encryption keys is critical to ensure security of encrypted data. The key management lifecycle involves different phases including:

- **Creation:** Creation of keys is the first step in the key management lifecycle. Keys must be created in a secure environment and must have adequate length. It is recommended to encrypt the keys themselves with a separate master key.
- **Backup:** Backup of keys must be made before putting them into production because in the event of loss of keys, all encrypted data can become useless.
- **Deployment:** In this phase the new key is deployed for encrypting the data. Deployment of a new key involves re-keying existing data.
- **Monitoring:** After a key has been deployed, monitoring the performance of the encryption environment is done to ensure that the key has been deployed correctly.
- **Rotation:** Key rotation involves creating a new key and re-encrypting all data with the new key.
- **Expiration:** Key expiration phase begins after the key rotation is complete. It is recommended to complete the key rotation process before the expiry of the existing key.
- **Archival:** Archival is the phase before the key is finally destroyed. It is recommended to archive old keys for some period of time to account for scenarios where there is still some data in the system that is encrypted with the old key.
- **Destruction:** Expired keys are finally destroyed after ensuring that there is no data encrypted with the expired keys.

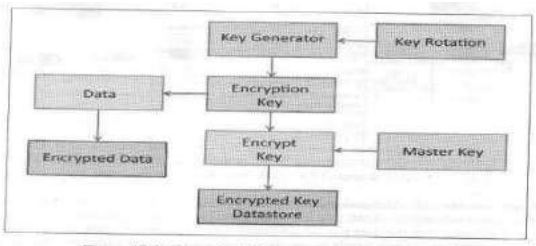


Figure 12.9: Example of a key management approach

12.8 Auditing

Auditing is an important requirement of data security regulations. Auditing requires logging all read and write accesses to data, including:

- User information
- Access type
- Timestamps
- Actions performed
- Records accessed

The primary purpose of auditing is to identify security breaches and enable necessary changes to prevent future breaches. Auditing has become even more important in cloud computing environments because resources are outsourced and the cloud infrastructure is managed by cloud service providers, limiting direct control.

Objectives of Auditing:

- Verifying efficiency and compliance of identity and access management controls as per established access policies.
- Verifying that authorized users are granted access to data and services based on their roles.
- Verifying whether access policies are updated in a timely manner upon changes in user roles.

- Verifying whether the data protection policies are sufficient.
- Assessment of support activities such as problem management.

PART-2 CLOUD FOR INDUSTRY,HEALTHCARE,&EDUCATION

13.1 Cloud Computing for Healthcare

The healthcare ecosystem consists of numerous entities including healthcare providers (primary care physicians, specialists, hospitals, for instance), payers (government, private health insurance companies, employers), pharmaceutical, device and equipment manufacturers, and consumers such as end users, families, and patients.

The process of provisioning healthcare services involves healthcare data that exists in different forms (structured or unstructured), can be stored but the skill and accuracy of that coding varies, and is stored in many different formats and sources (such as relational databases, file servers, for instance) and in different locations.

To promote more coordination of care across the multiple providers involved with patients, **Electronic Health Information** is increasingly aggregated from diverse sources into **Electronic Health Record (EHR)** systems. Physicians receive and analyze this health information from many sources such as laboratory tests and medical devices (such as CT and MRI scans).

In the diagnosis process, physicians critically diagnose patients based on different information. Chronic disease patients are typically seen by multiple physicians at different times. Care is so distributed that the provider network around the average primary care physician includes some 30 other physicians. Information sharing among the physicians is critical to high-quality care. Physicians can avoid diagnosis errors through the stakeholder network and can reduce analysis and time spent after consulting specialists.

Payers

Health payers can increase the effectiveness of their care management programs by providing value-added services and giving access to health information to members.

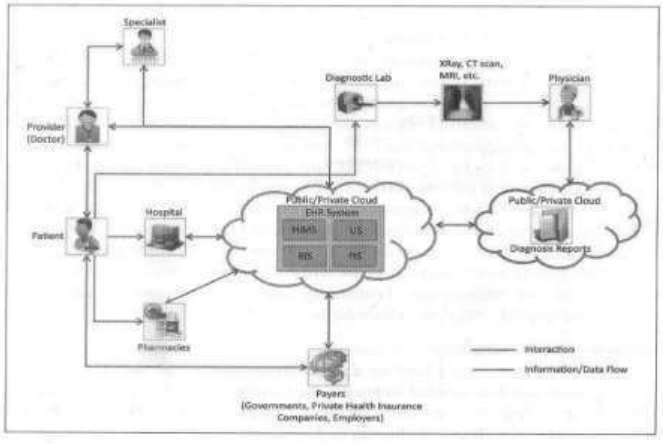


Figure 13.1: Cloud computing for healthcare

EHRs capture and store information on patient health and provider actions including individual-level laboratory results, diagnostic, treatment, and demographic data.

Figure 13.2 shows a screenshot of a cloud-based EHR application. The figure shows a patient summary page of a **Patient Health Record (PHR)** application. The PHR application maintains information such as:

- Patient visits
- Allergies
- Immunizations
- Lab reports
- Prescribed medicines
- Vital signs
- Insurance history

The primary use of EHRs is to maintain all medical data for an individual. In order to avoid duplication of efforts, the stored data can be reused for other purposes. EHR data can be reused for value-added services to show overall patient health.

EHRs can also be the source for aggregated information about overall patient populations. The EHR data can be used to provide healthcare applications like:

- Disease surveillance
- Clinical decision support
- Risk prediction
- Prognosis
- Syndromic diagnosis
- Small-area analysis-based surveillance
- HealthMap

To exploit the potential to aggregate data for advanced healthcare research, an instance is needed for efficiently integrating information from distributed and heterogeneous healthcare IT systems and analyzing the integrated information.

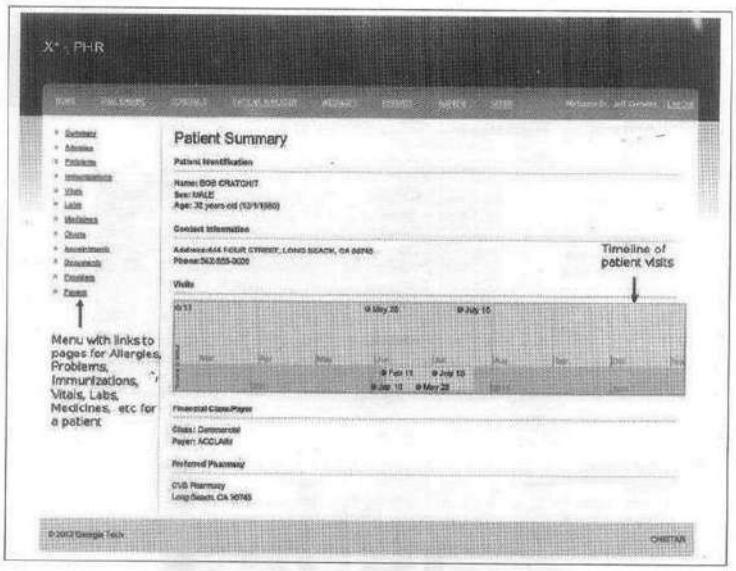


Figure 13.2: Screenshot of a cloud-based Patient Health Record application [10].

13.2 Cloud Computing for Energy Systems

1. Cloud-Based Framework

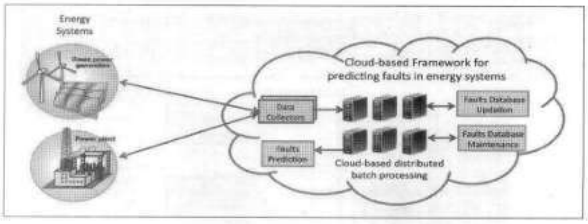


Figure 13.5: Cloud computing for energy systems

- Energy systems (smart grids, turbines, power plants) generate massive sensor data.
- Cloud provides:
  - Database storage for large-scale sensor data.
  - Batch processing for handling big data efficiently.
  - Data analysis for fault prediction and diagnostics.
- Enables predictive maintenance and real-time monitoring.

3. **Workflow for Sensor Data Collection**

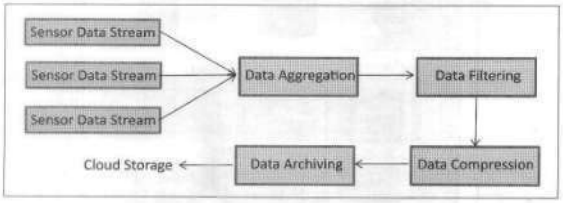


Figure 13.6: Workflow for collecting machine sensor data in a cloud.

Steps in cloud-based sensor data handling:

1. Multiple Sensor Data Streams → continuous input from machines.
2. Data Aggregation → combine streams into manageable formats.
3. Data Filtering → remove bad/missing records.
4. Data Compression → reduce size for efficient storage.
5. Data Archiving → store processed data.
6. Cloud Storage → centralized, scalable repository.

4. **Case-Based Reasoning (CBR) for Failure Prediction**

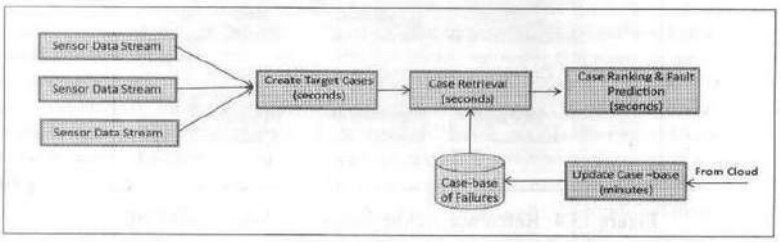


Figure 13.7: Approach for predicting failures in real-time using case-based reasoning demonstrated in [8].

- Sensor Data Stream → input for monitoring.
- Create Target Cases → define exception/fault conditions.
- Case Indexing → organize cases for retrieval.

- Case Retrieval → find similar past cases.
- Case Adaptation → adjust solutions to current context.
- Case Retention & Update → store new experiences in cloud case base.
- Supports real-time fault prediction where mathematical models are hard to establish.

4. Industry Applications
   - GE → Gas turbine diagnostics.
   - SKF WindCon → Wind turbine condition monitoring.
   - OpenPDC → Real-time processing of Phasor Measurement Unit (PMU) data.

Cloud Computing for Smart Grids

- Smart grids integrate **cloud infrastructure** for real-time data analysis and predictive management.
- They handle **data-intensive flows** from:
  - Energy generation (centralized/distributed)
  - Transmission lines
  - Distribution systems
  - Consumption (residential, commercial, industrial)
  - Equipment health monitoring

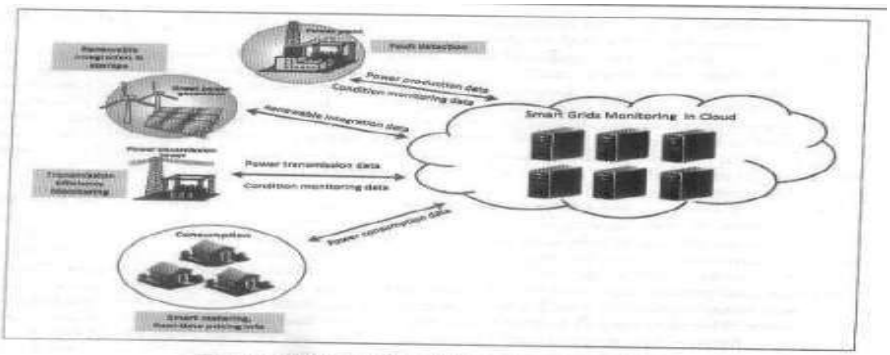


Figure 13.8: Cloud computing for smart grids

Benefits of Cloud Integration

- **Real-time information & power exchange** using sensing and measurement technologies.
- **Prevention of power thefts** via smart metering.
- **Efficiency improvement** across the entire electric system (generation → transmission → consumption).
- **Dynamic optimization** of operations, maintenance, and planning.
- **Energy feedback to users** with real-time pricing information to reduce consumption.
- **Demand response management**:
  - Appliance control
  - Energy storage mechanisms
  - Lowering peak demand

Fault Detection & Risk Management

- Condition monitoring data helps detect faults and predict outages.
- **Probabilistic risk assessments** identify equipment, plants, and transmission lines most likely to fail.

- **Predictive analytics** anticipate problems before they occur, enabling proactive measures.

13.3 Cloud computing for transportation systems

Modern transportation systems are driven by data collected from multiple sources which is processed to provide new services to the stakeholders. By collecting large amount of data from various sources and processing the data into useful information, data-driven transportation systems can provide new services such as advanced route guidance.

Fleet Tracking

Vehicle fleet tracking systems use GPS technology to track the locations of the vehicles in real-time. Cloud-based fleet tracking systems can be scaled up on demand to handle large numbers of vehicles. Alerts can be generated in case of deviations in planned routes. The vehicle location data can be aggregated and analyzed for detecting inefficiencies in the supply chain such as traffic congestions on routes, assignment and manufacturing.

Route Generation & Scheduling

Route generation and scheduling systems can generate end-to-end routes for vehicles based on transportation needs and feasible schedules based on the availability of vehicles. As the transportation network grows in size and complexity, relationship management of route combinations increases exponentially. Route administration needs advanced computing abilities to build and keep routes updated in a timely fashion.

Condition Monitoring

Condition monitoring solutions for transportation systems allow monitoring the conditions inside containers. For example, containers carrying fresh food produce can be monitored to prevent spoilage of food. Condition monitoring with sensors such as temperature, pressure, and humidity, for instance. For a large fleet of vehicles, an enormous amount of data can be difficult to analyze in real time. Cloud-based systems can be used for this purpose that not only detect food spoilage but also suggest alternative routes to prevent spoilage.

Planning, Operations & Services

Different transportation solutions (such as fleet tracking, condition monitoring, route generation, scheduling, cargo operations, fleet maintenance, customer service, order tracking, billing, & collection, for instance) can be moved to the cloud to provide a seamless integration between order management, tactical planning & execution and customer facing processes. Such integrated cloud-based systems for planning, operations and customer facing processes in transportation systems enable the organizations to improve service levels, reduce costs, and reduce network management costs (e.g. toll costs), and improve service levels.

Collecting and organizing location and sensor data from containers and vehicles for raising alerts about violation of certain conditions is a major challenge for the following reasons:

1. Wide coverage is needed for collection of location and sensor data (vehicles carrying fresh food supply).
2. Data needs to be collected in real time for both fresh food and real-time large multiple shipments across the fleet.
3. At massive scale, since the real-time data from a large number of vehicles is collected simultaneously.
4. The collected data needs to be organized and processed in real-time.
5. A massive amount of data needs to be leveraged for cloud solutions deployable to ensure wide popularity.

A cloud-based platform can enable real-time fresh food supply tracking and monitoring. Spoiled fruits and vegetables during transport and distribution not only result in losses to the

distributors but also present a hazard to food safety. Therefore tracking and monitoring of fresh food during the entire transportation process can help in effective tracking and rerouting before spoilage occurs. Alerts can be generated in real-time, e.g., when container temperature exceeds the allowed limit.

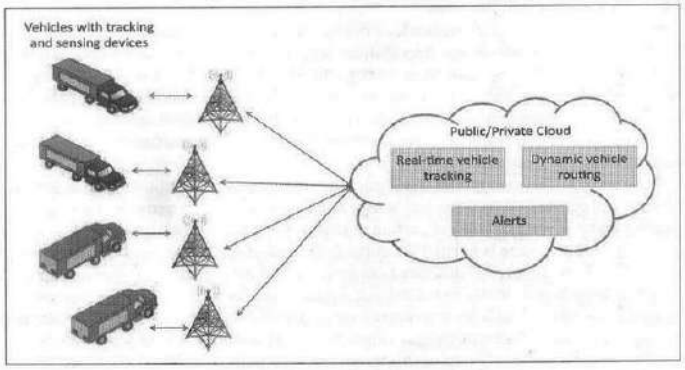


Figure 13.9: Cloud computing for transportation systems

13.4 Cloud Computing for Manufacturing Industry

Manufacturing industry researchers are exploring the potential of utilizing cloud computing for manufacturing that would enable collaborative design, distributed manufacturing and co-creation. There are two forms of cloud manufacturing:

- One involves the use of cloud computing technologies for manufacturing.
- The other involves service-oriented manufacturing that replicates the cloud computing environment using physical manufacturing resources.

Cloud computing is well suited for the manufacturing industry, where computing needs vary significantly with the product lifecycle phase. It can provide computational, storage, and software services on demand, allowing improved resource sharing, rapid prototyping, and reduced manufacturing costs.

An application of cloud computing technologies for manufacturing is in **Industrial Control Systems (ICS)**, which include SCADA systems, distributed control systems (DCS), and programmable logic controllers (PLC). These systems continuously generate monitoring and control data. Real-time collection, management, and analysis of production operations data in the cloud can help estimate system states, improve plant and personnel safety, and prevent catastrophic failures.

Wu et al. proposed expanding cloud computing to computer-aided design and manufacturing, introducing the concept of **cloud-based design and manufacturing (CBDM)**.

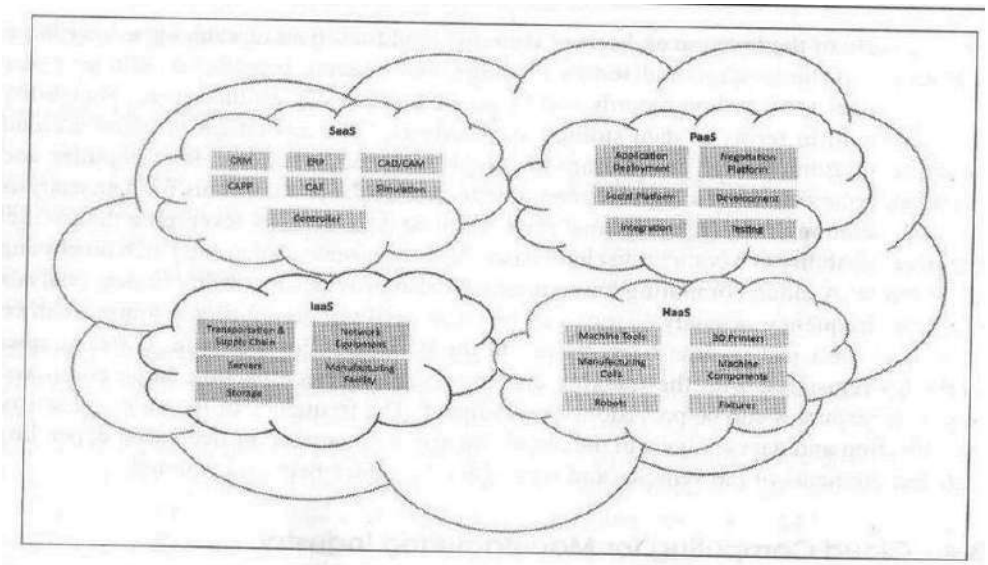


Figure 13.10: Cloud-based design and manufacturing

Figure 13.10 shows examples of services in CBDM, including:

- **Software-as-a-Service (SaaS):** Provides applications such as CRM, ERP, CAD/CAM hosted in the cloud, accessible via thin clients like web browsers.

Applications can be accessed by multiple teams spread across different locations working in a collaborative development environment.

- **Platform-as-a-Service (PaaS):** Allows deployment of applications without buying or managing infrastructure. Provides services for developing, testing, and deploying applications in an integrated environment. Design teams can use PaaS for collaboration and productivity.
- **Infrastructure-as-a-Service (IaaS):** Provides physical resources such as servers and storage that can be provisioned on demand.
- **Hardware-as-a-Service (HaaS):** Provides access to machine tools, 3D printers, manufacturing cells, and industrial robots. Hardware can be rented through the CBDM environment. For example, a HaaS service for 3D printing can be used by multiple organizations to print parts. Cloud-connected 3D printers enable rapid tooling and scalability for traditional manufacturing processes.

13.5 Cloud Computing for Education

Cloud computing is bringing a transformative impact in the field of education by improving the reach of quality education to students through the use of online learning platforms and collaboration tools.In recent years, the concept of **Massively Online Open Courses (MOOCs)** appears to be gaining popularity worldwide, with large numbers of students enrolling for online courses.

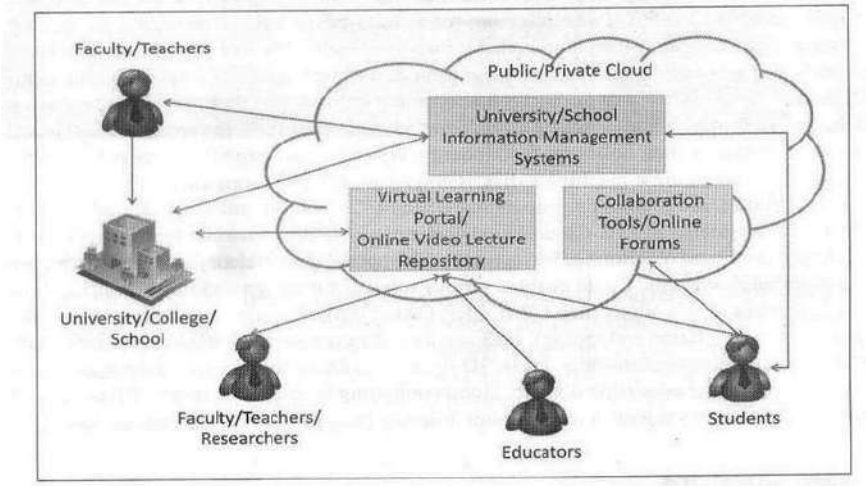


Figure 13.11: Cloud computing for education

Many universities across the world are using cloud platforms for providing online degree programs. Lectures are delivered through live/recorded video using cloud-based content delivery networks to students across the world.

- Online proctoring for distance learning programs is becoming popular through cloud-based live video management systems, where online proctors observe test takers remotely.
- Virtual labs are provided to distance learning students through cloud-based platforms, enabling remote access to the same software and applications used by on-campus students.
- Cloud-based assessment platforms are used for sharing reading material and providing assignment feedback.

Cloud-based education guidance systems and online forums help students discuss common problems and seek guidance from faculty.

Universities, colleges, and schools can use cloud-based school management systems for:

- Online admission
- Student data collection
- Distance education programs
- Online exams
- Tracking student progress
- Collecting feedback

Cloud-based systems can help universities, colleges, and schools cut down costs.

PART-3 MIGRATING OF A CLOUD

INTRODUCTION:

The promise of cloud computing has raised the IT expectations of small and medium enterprises beyond measure. Large companies are deeply debating it. Cloud computing is a disruptive model of IT whose innovation is part technology and part business model—in short a *"disruptive techno-commercial model"* of IT.

This tutorial chapter focuses on the key issues and dilemmas faced by decision makers, architects, and systems managers in trying to understand and leverage cloud computing for their IT needs. Questions discussed include:

- When and how to migrate applications into a cloud.

- Which parts or components of IT applications should migrate, and which should not.
- What kinds of customers benefit most from migrating IT into the cloud.

The chapter describes key factors underlying these questions and introduces a **Seven-Step Model of Migration into the Cloud**.

Cloud computing has been a hotly debated topic among IT professionals and researchers in both industry and academia. Discussions appear in blogs, websites, and research efforts, leading to entrepreneurial initiatives to help enterprises migrate into the cloud despite challenges, benefits, and limitations.

Large vendors such as Google, Amazon, and Microsoft began offering cloud services on what seemed like a demonstration and trial basis. They charged lower fees, sometimes unsustainable, as part of these trial offerings.

Cloud computing is described as:

"It is a techno-business disruptive model of using distributed large-scale data centers—either private, public, or hybrid—offering customers a scalable virtualized infrastructure or an abstracted set of services qualified by service-level agreements (SLAs) and charged only by the abstracted IT resources consumed."

Key points:

- Cloud computing shifts enterprises away from traditional IT investments toward scalable, efficient services.
- It allows enterprises to provision IT capacity based on demand, which may vary seasonally or cyclically.
- Defining cloud computing is challenging due to rapid technological advancements and evolving business models.

The Promise of the Cloud

Most users of cloud computing services offered by large-scale data centers are least bothered about the complexities of the underlying systems or their functioning. Given the heterogeneity of systems or software running on them, users are impressed by the **simplicity, uniformity, and ease of use** of Cloud Computing Service abstractions.

- For small and medium enterprises, cloud computing usage for cyclical IT needs has yielded **substantial economic savings**. Many success stories have been documented and discussed online.
- This economics and the associated trade-offs of leveraging cloud computing services are now popularly called **"cloudonomics."**

Cloudonomics

- "Pay as you use" – Lower Cost of Ownership
- No upfront CAPEX – No asset ownership (CAPEX), only operational expense (OPEX)
- SLA-driven services – Measurable Service Level Agreements (SLA)
- Enhanced ROI, competitive advantage

Technology

- "Virtual" unlimited availability – Computing/Storage/Bandwidth
- Time-to-market reduction – Rapid provisioning
- Improved utilization – Multi-tenancy
- "Greener" IT – Energy efficiency and reduced carbon footprint
- Increasing innovation at minimal IT cost

The Cloud Service Offerings and Deployment Models

Service Types:

1. IaaS (Infrastructure as a Service)
    - Abstract Computing/Storage/Bandwidth Infrastructure
    - Examples: Amazon Web Services (AWS): EC2, S3, RDS, CDNs, CloudWatch
2. PaaS (Platform as a Service)
    - Abstracted Programming Platforms with Development Interfaces
    - Examples: Google App Engine, Windows Azure/.NET, Microsoft Azure/.NET
3. SaaS (Software as a Service)
    - Applications with encapsulated interfaces & platforms
    - Examples: Google Apps; Gmail; Yahoo Mail; Facebook; Twitter

Deployment Models:

- Public Clouds
- Hybrid Clouds
- Private Clouds

Microsoft and other companies classify cloud services into IaaS, PaaS, and SaaS.

- Administrators often prefer IaaS.
- Programmers use PaaS for development.
- End users find SaaS most popular.

Cloud computing has evolved to include hybrid cloud models, integrating public and private infrastructures.

Challenges in the Cloud

- Cloud services (IaaS, PaaS, SaaS) look simple on the surface:
    - IaaS → makes IT infrastructure look easy
    - PaaS → makes programming look easy
    - SaaS → makes using applications look easy
- But behind the scenes, things are very complex:
    - Systems can fail often
    - They are made up of different, mixed technologies (heterogeneous)
    - They consume a lot of resources
    - Security problems are common
- Cloud providers often give the impression of:
    - Perfect network reliability
    - Zero latency (instant response)
    - Infinite bandwidth
- In reality, these are fallacies (false assumptions). Good distributed systems are designed by avoiding these unrealistic ideas.
- The irony:
    - Cloud computing pretends to be simple and perfect for users.
    - But providers must manage the messy, failure-prone systems underneath.

BROAD APPROACHES TO MIGRATING INTO THE CLOUD

Cloud computing is increasingly relevant as a techno-business disruptive model. Gartner ranked cloud computing among the top 10 strategic technologies to watch for in 2010. The concept of *Cloudonomics* deals with the economic rationale for leveraging the cloud. It raises questions about IT costs, total cost of ownership (TCO), and strategic parameters for enterprise IT. Decision-makers, IT managers, and software architects face dilemmas when planning new enterprise IT initiatives.

Why Migrate?

Economic and business reasons drive migration of enterprise applications to the cloud. Technological reasons include integration of enterprise applications and adoption of cloud technologies. Integration with new applications developed on the cloud and adoption of cloud computing services are key use cases of migration.

Migration of an application into the cloud can occur in different ways: migrating the application as-is, modifying the code, or redesigning the architecture. Migration can happen at five levels: application, code, design, architecture, and usage.

Mathematical model of migration:

$$P=P_C+P_I-P_{OC}+P_I$$

Where:

- P: application before migration running in a captive data center
- PC: application part after migration into a (hybrid) cloud
- PI′: part of the application running in the captive local data center
- POC: application part optimized for cloud
- PI: null when the entire application is migrated onto the cloud

Hybrid cloud usage implies partial migration. Migration use-case scenarios:

- 30 scenarios for IaaS migration
- 20 scenarios for PaaS migration
- SaaS migration is described as importing usage only, with no enterprise application migration.

Deciding on the Cloud Migration Evaluating whether to migrate enterprise applications to the cloud involves factors such as operational expenses, licensing issues, SLA compliance, and pricing variability of cloud services. A weightage-based decision-making technique uses a questionnaire to assess migration impact.

Mathematical model:

$$C_i = \sum_{i=1}^{M} B_i \left(\sum_{j=1}^{N} A_{ij} x_{ij} \right) \equiv C_k$$

Where:

- Ci: lower weightage threshold
- Ck: higher weightage threshold
- Bi: weightage assigned to class i
- Aij: expected impact for question j of class i
- xij: fraction (0–1) representing relevance/applicability of the answer

Since all except one class of questions do not have all N questions, the corresponding row has a null value.

Seven-Step Model of Migration into the Cloud

A structured, phased approach to ensure smooth migration.

Steps:

1. **Conduct Cloud Migration Assessment**
   - Evaluate applications, architecture, code, design, and usage.
   - Assess tools, test cases, configurations, functionalities, and non-functional requirements (NFRs).
2. **Isolate the Dependencies**
   - Identify interdependencies among applications, databases, and services.

- Helps avoid migration failures due to overlooked links.

3. **Map the Messaging & Environment**
   - Align communication flows and environment setups.
   - Ensures compatibility between on-premises and cloud systems.
4. **Re-architect & Implement the Base Cloud Foundation**
   - Redesign applications for cloud-native architecture.
   - Establish the foundational infrastructure.
5. **Leverage Cloud Functionalities & Features**
   - Use elasticity, autoscaling, cloud storage, and other cloud-native features.
   - Augment lost functionalities with cloud services.
6. **Test the Migration**
   - Validate with extensive test suites.
   - Ensure performance, reliability, and compliance.
7. **Go-live**
   - Deploy migrated applications into production.
   - Monitor closely for stability and performance.

AWS-Specific Seven-Step Model

Amazon's proprietary version emphasizes practical migration strategies.

Phases:

1. **Cloud Migration Assessment** – Identify dependencies and strategies.
2. **Proof of Concepts** – Build reference migration architecture.
3. **Data Migration Phase** – Segment, cleanse, and move data using cloud storage.
4. **Application Migration** –
   - *Forklift strategy*: Move entire application with dependencies.
   - *Hybrid strategy*: Keep critical parts on-premises, move noncritical parts to cloud.
5. **Leverage AWS Features** – Elasticity, autoscaling, storage.
6. **Optimization** – Fine-tune for efficiency.

Migration Risks and Mitigation

Risks are categorized into **general** and **security-related**.

General Risks:

- Performance monitoring & tuning
- Business continuity & disaster recovery
- Compliance with standards & governance

- IP and licensing issues
- Quality of Service (QoS) & SLAs
- Data ownership, transfer, and storage concerns
- Portability & interoperability challenges
- Complexity leading to migration failure

Security Risks:

- Compliance with Cloud Security Alliance guidelines
- Trust & privacy issues
- Legal compliance & audit requirements
- Multi-tenancy risks (data leakage)
- Incident response quality
- Identity management & access control

PART-4 ORGANIZATIONAL READINESS AND CHANGE MANAGEMENT IN THE CLOUD AGE

INTRODUCTION

Studies by the OECD (2002) found a strong correlation between organizational/workplace changes and investment in IT. Canadian government studies also showed that firms investing more in IT experience more organizational change than those investing less.

A study by Bresnahan, Brynjolfsson, and Hitt found IT investment positively linked to:

- Process re-engineering
- Organizational structure changes
- Employee empowerment
- Firm value

Gartner research (2009) highlighted cloud computing as the fastest-growing IT trend, alongside data center power/cooling and mobile device technologies.

To support enterprise goals, IT must adapt and embrace emerging technologies (e.g., cloud computing, IaaS, PaaS, SaaS). These can disrupt IT service strategy, design, transition, and operation.

The Context

- Cloud adoption forces clarity of data ownership.
- Protecting intellectual property and copyrights is critical.
- Organizations must balance innovation with adoption.
- Managers must handle complexity, uncertainty, and maintain flexibility.

The Take Away Transitioning to change management maturity requires enhancing:

1. **Managing the Environment –** Understand people, processes, culture.
2. **Recognizing Trends –** Observe business and technology drivers.
3. **Leading for Results –** Assess readiness and select solutions with clear business value.

BASIC CONCEPT OF ORGANIZATIONAL READINES

Change brings fear, uncertainty, and doubt (FUD syndrome).Employees get comfortable in stable roles. Change requires clear communication of reasons and alignment with new vision. Leadership, HR, and reward systems must support new goals.

Human behavior: people resist change, preferring stability.

IBM Case Study – "Making Change Work"

- 60% of projects fail to meet objectives.

- Key barriers: changing mindsets (58%), corporate culture (49%).
- "Soft stuff" (people issues) is hardest to manage.

Forrester Survey (2009) Large enterprises lean toward private clouds due to:

1. Protecting existing infrastructure investments.
2. Managing security risks (data integrity, privacy).

Case Study: Waiting in Line for a Special Concert Ticket Scenario: You wait in freezing weather for tickets, only to find the concert sold out. The case study asks how you react and what you do next.

Five Stages of Reaction to Change (Kübler-Ross Model)

1. **Denial –** Reject the reality.
2. **Anger –** Blame external factors.
3. **Bargaining –** Try to negotiate.
4. **Depression –** Feel disappointed and lost.
5. **Acceptance –** Accept reality and move on.

Originally proposed by Dr. Elizabeth Kübler-Ross in *On Death and Dying*. Applied to change management, these stages describe emotions felt during major organizational changes.

What Do People Fear?

- Fear of leaving comfort zone: "That's not how we do things here."
- Fear of risk: losing position, power, benefits, or jobs.
- Some organizations reward risk-avoidance, promoting longevity over innovation.

Concerns in Cloud Computing Common concerns include:

- Security and privacy protection
- Loss of control (paradigm shift)
- New model of vendor relationship management
- More stringent contract negotiation and SLA
- Availability of an executable exit strategy

DRIVERS FOR CHANGES: A FRAMEWORK TO COMPREHEND THE COMPETITIVE ENVIRONMENT

Five driving factors for change:

1. Economic (global/local, external/internal)
2. Legal, political, and regulatory compliance
3. Environmental (industry structure and trends)
4. Technology developments and innovation
5. Sociocultural (markets and customers)

This framework helps organizations analyze trends, forecast plausible futures, identify uncertainties, and prepare for unexpected decisions. Internal factors (like financial management, talent, innovation) and external factors (like competitor strategies) both shape organizational direction.

Economic (Global and Local, External and Internal)

- Managers face paradoxes: growth vs. downturns, shrinking markets, declining margins, rising competition.
- Sample questions:
  - Current economic situation?
  - Future outlook (1–5 years)?
  - Factors influencing economy?
  - Is capital accessible?

- Buy vs. build?
- Total cost of ownership (TCO)?

Legal, Political, and Regulatory Compliance

- Focus: transparency, compliance, conformity.
- Sample questions:
  - Regulatory requirements?
  - Implications of noncompliance?
  - Global geopolitical issues?

Environmental (Industry Structure and Trends)

- Impact of environment on business: natural environment, health, safety.
- Sample questions:
  - Implications of global warming?
  - Is a green data center over-hyped?

Technology Developments and Innovation

- Scientific discoveries drive economic growth.
- Emerging technologies (bioscience, nanotech, IT) can transform lives.
- Sample questions:
  - When will IT standards be finalized?
  - Who leads cloud computing technologies?
  - Virtualization: write once, run anywhere?
  - How does cloud computing open innovation?
  - Can apps dynamically configure in real time?
  - Infrastructure limits for future IaaS?
  - Will data execute correctly?

Sociocultural (Markets and Customers)

- Sociocultural factors = human side of markets.
- Example: U.S. defense industry lost 90% of workforce in a decade.
- Survival depends on transforming competitors into collaborators and adapting to new realities.

Creating a Winning Environment At the cultural level, change requires planning and resources. Senior management and employees often interpret change differently. To succeed, executives must communicate clearly:

1. New direction of the firm
2. Urgency of change
3. Risks of status quo vs. change
4. Potential rewards

Building a Business-Savvy IT Organization

- Are infrastructure burdens unnecessary?
- Which IT roles matter most to business?
- Should focus shift to high-value product issues?

Cultivating an IT-Savvy Business Organization

- Do users need new skills and expertise?

Cloud Computing Value Proposition

- Buy, don't build
- Avoid large upfront capital investment

COMMON CHANGE MANAGEMENT MODELS There are many approaches to change management. Two common ones plus a proposed model are:

- **Lewin's Change Management Model**
- **Deming Cycle (Plan, Do, Study, Act)**
- **CROPS Change Management Framework**

Lewin's Change Management Model Created in the 1950s, Lewin's model has three stages:

1. **Unfreeze –** Break the comfort zone, motivate people to accept change.
2. **Transition –** Support people with training and coaching as they learn new behaviors.
3. **Refreeze –** Stabilize the environment again until the next change.

Deming Cycle (Plan, Do, Study, Act) Also called the **PDCA cycle**, originally conceived by Walter Shewhart (1930s) and adopted by Edward Deming (1950s). It is a continuous improvement model.

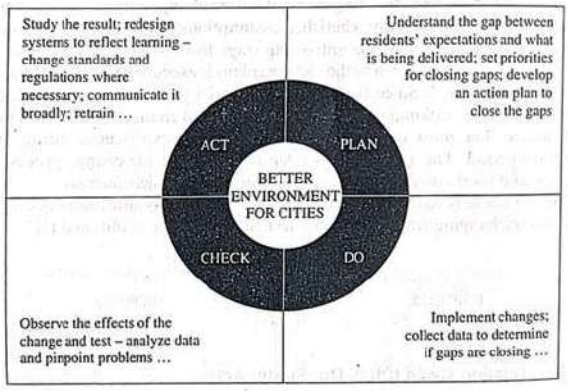
Steps:

- **PLAN –** Recognize opportunity, plan change.
- **DO –** Execute on a small scale.
- **CHECK –** Evaluate performance, analyze data.
- **ACT –** Standardize successful changes, redesign if needed.

The cycle is evergreen: each pass flows into the next, supporting continuous quality improvement.

Diagram highlights:

- PLAN → Understand gaps, develop action plan
- DO → Implement changes, collect data
- CHECK → Analyze effects, pinpoint problems
- ACT → Standardize, communicate results



GURE 22.1. Deming's PDCA cycle.

A Proposed Working Model: CROPS Change Management Framework Focuses on people as the most valuable asset in the knowledge economy.

Challenges:

- Employees resist change due to fear, comfort with status quo, or lack of skills.
- Best approach: transparency, sustained communication, education.
- Leaders (often CEOs) must articulate the vision and roadmap.

CROPS Framework Components:

- **Culture –** Shared values, beliefs, customs, rituals, stories.
- **Rewards –** Systems that measure performance and determine compensation.
- **Organization & Structures –** Job roles, alignment with vision and strategy.
- **Processes –** Structured activities producing services/products.
- **Skills & Competencies –** Specialized skills enabling innovation and competitiveness

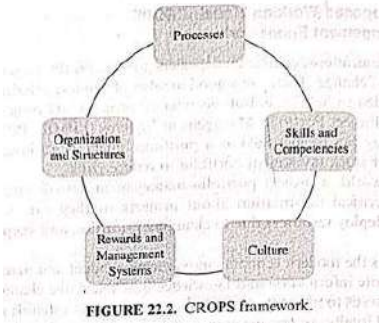


FIGURE 22.2. CROPS framework.

Rewards and Management System – Ensures employees have skills/tools, measures performance, sets compensation.
Organization and Structures – Aligns processes with vision, mission, and strategies.
Process – Defined as structured activities that transform inputs into outputs of value.
CHANGE MANAGEMENT MATURITY MODEL (CMMM) Helps organizations:

- Analyze strengths/weaknesses of change management.
- Identify opportunities for improvement.
- Build competitiveness.

Based on **Capability Maturity Model (CMM)**, adapted for business maturity. Describes a five-level process maturity model.
Business Value of CMMM Measured in terms of **ROI**:

$$ROI(CMMM) = \frac{\text{Estimated total business performance improvement}}{\text{Total CMMM investment (TCO)}}$$

Where:

- ROI = observed business value or return on investment from IT initiative (CMMM)
- Business performance improvement includes reducing error rate, etc.

A Case Study: AML Services Inc. AML (A Medical Laboratory Services Inc.) is a medical lab provider serving a city of one million people, with 150 employees. The industry requires high startup investment (equipment, IT, skilled staff).

- In 2009, demand grew due to H1N1 flu and aging population.
- AML needed to upgrade outdated IT infrastructure.
- They moved toward web-based mail servers and appointment booking systems to reduce courier workload and support mobile devices.
- Since 2008, patients could book appointments online.

The CIO hired consultants to assess cloud computing. Key questions:

- Should AML consider cloud computing?
- Is AML ready for it?
- What does “done” look like?
- How can challenges of change be overcome?

ORGANIZATIONAL READINESS SELF-ASSESSMENT (WHO, WHEN, WHERE, HOW) Organizational assessment seeks to understand the current state and define strategies to move toward the desired future state.

Organizational assessment process:

- Complete strategic analysis first.
- Formulate future goals and objectives.
- CEO must articulate vision and rally support.

Effective readiness assessment should:

- Articulate and reinforce reason for change.
- Determine current (as-is) state.
- Identify gaps between current and future state.
- Anticipate barriers to change.
- Establish action plans to remove barriers.

Stakeholder involvement:

- Involve all affected people, not just management.
- Cross-section representation is critical.

Key questions to ask:

- How big is the gap?
- Does the organization have capacity to execute changes?
- Will employees respond positively?
- Are employees ready to adopt changes?
- Are there critical barriers?
- Are business partners ready to support changes?

TABLE 22.2. Working Assessment Template

Nontechnical	Agree	Don't Know	Disagree
Does your organization have a good common understanding of why business objectives have been met or missed in the past?			
Does your organization have a good common understanding of why projects have succeeded or failed in the past?			
Does your organization have a change champion?			
Does your organization perceive change as unnecessary disruption to business?			
Does your organization view changes as the management fad of the day?			
Does your organization adopt an industry standard change management best practice and methodology approach?			
Does your organization adopt and adapt learning organization philosophy and practice?			
How familiar is your organization with service provisioning with an external service provider?			
Technical			
Does your organization implement any industry management standards? • ITIL • COBIT • ITSM • others			
Does your organization have a well-established policy to classify and manage the full lifecycle of all corporate data?			
Can you tell which percentage of your applications is CPU-intensive, and which percentage of your applications is data-intensive?			

PART-5 Legal Issues in Cloud Computing

Cloud computing legal context:

- In cloud environments, businesses outsource application hosting and data management to third parties.
- New aspects include:
  - Cloud-based service models
  - Relocation of system control to third parties
  - Virtualization
  - Multi-vendor integration
  - Borderless globalization

Definition of Cloud Computing Defined by NIST as:
A model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (networks, servers, storage, applications, services) that can be rapidly provisioned and released.
It includes:

- 5 essential characteristics
- 3 service models
- 4 deployment models

Overview of Legal Issues

- Privacy and data security (PII and sensitive business info)

- Licensing concerns (shrink wrap, click wrap agreements)
- Service agreements and risk models
- Forensic investigation: location of data, mix of on-premise/off-premise
- Jurisdictional issues: data stored in multiple regions, cross-border laws

Jurisdictional issues:

- Physical server locations affect which country's law applies in case of breach.
- Laws vary when data moves geographically among centers.
- Business concerns: what happens if a provider goes bankrupt or ceases operations?

Distinguishing Cloud Computing from Outsourcing and ASP

Outsourcing:

- Outsourcer runs entire business/IT process for customer.
- Software usually belongs to customer.
- Location of data is known and agreed.
- Pricing negotiated based on scope.

ASP (Application Service Provider):

- Provides third-party software service.
- Not customized, less complex than outsourcing.
- Software treated like a product.
- Location not fixed, pricing usually uniform.
- Licensing shifted from traditional models to service arrangements.
- No inherent ability to scale on demand.

Cloud Computing vs. Outsourcing/ASP (continued)

- Cloud computing places less emphasis on location of data.
- Covers multiple service models (SaaS, IaaS, PaaS).
- Pricing often unit-based, similar to utility computing.
- Providers may have multiple geographically dispersed data centers.
- Customers often don't know exact data location at any given time.
- Cloud allows rapid scaling up/down to meet demand.

DATA PRIVACY AND SECURITY ISSUES

U.S. Data Breach Notification Requirements

- Data breach = loss of unencrypted personal info (name + financial data).
- Causes: compromised servers, lost devices, theft.
- Risks: identity theft, fraud (user side); lawsuits, FTC investigations, lost customers, reputation damage (provider side).
- Serious breaches in U.S. have exceeded $50 billion in costs.
- Almost all U.S. states require notification of affected individuals.
- Congress considering nationwide notification requirements.

U.S. Federal Law Compliance

Gramm–Leach–Bliley Act (GLB): Financial Privacy Rule

- Requires financial institutions to ensure confidentiality of personal information and protect against unauthorized access.
- Applies to service providers, including cloud providers handling financial data.
- Duties include:
    1. Cloud provider must meet confidentiality requirements.
    2. Demonstrate security to prevent unauthorized access.
    3. Control who has access.

FTC Role: Safeguards Rule & Red Flags Rule

- **Safeguards Rule** requires businesses engaged in financial services to have a written information security plan.
    - Must designate employees to coordinate security.
    - Identify and assess risks to customer information.
    - Design, implement, monitor, and test safeguards.

Additional requirements under **Safeguards Rule**:

- Select service providers that maintain safeguards.
- Adjust program based on changes in business or monitoring results.

Red Flag Rules (FACTA, 2007)

- Intended to curb identity theft.
- Require financial institutions and organizations with credit accounts (banks, retailers, telecoms) to implement fraud detection programs.
- Cloud providers must also comply, with written plans and monitoring systems.

Health Insurance Portability and Accountability Act (HIPAA) & HITECH Act

- Require notification of breaches of unencrypted health records.
- Apply to covered entities and cloud providers handling health data.

USA PATRIOT Act (2001)

- Allows installation of devices to capture data (computer taps).
- Extends government access to financial and student information stored remotely.
- Requires only certification that data is relevant to an investigation.
- Cloud providers may face privacy challenges under this Act.

International Data Privacy Compliance

European Union Data Privacy Directive (1995)

- Requires EU countries to pass laws protecting personal/business data.
- Covers written, oral, electronic, and Internet-based data.
- Has **extraterritorial effect**: any organization receiving EU personal data must meet EU standards of "adequate protection."

Article 17 Requirements:

- Data controllers must implement technical and organizational safeguards.
- Written contracts required between controllers and processors.
- Cloud providers acting as processors must comply.

Adequate Protection Methods:

1. EU-recognized national laws (e.g., Argentina, Canada, Guernsey, Isle of Man, Switzerland).
2. Safe harbor provisions.
3. EU model contract clauses.
4. Binding corporate rules.

International Data Privacy Compliance (continued)

- Cloud users must ask about geographic placement of data and compliance methods before using EU-based services.

Other Jurisdictions:

- **Argentina –** similar to EU approach.
- **Brazil –** constitutional right to privacy, but patchwork laws.
- **China –** few privacy rules.
- **Hong Kong –** Personal Data Ordinance applies broadly.

- **India –** limited provisions, mainly public sector.

Canada – PIPEDA (Personal Information Protection and Electronic Documents Act)

- Supports electronic commerce by protecting personal information.
- Transfers accountability to contractual arrangements between organizations.
- Service providers (including cloud providers) are accountable for data protection.
- Principle 6 of CSA Model Code: organizations are responsible for personal information under their control.
- Compliance enforced through contracts, regardless of geography.

Australia Privacy Act

- Based on:
  - 11 *Information Privacy Principles* (public sector)
  - 10 *National Privacy Principles* (private sector)
- Governs use, disclosure, and management of personal data.
- Restrictions on sending personal data abroad:
  1. Recipient will uphold principles
  2. Consent from data subject
  3. Transfer necessary for contractual obligations

Office of the Privacy Commissioner Expectations

- Cloud providers must comply with National Privacy Principles 4 and 9:
  - Ensure personal information is accurate, up-to-date, and secure.
  - Protect personal information transferred outside Australia.

CLOUD CONTRACTING MODELS

Licensing Agreements vs. Services Agreements

License Agreement

- Licensor provides a copy of software for use (non-exclusive).
- License terms bind the licensee.
- Remedies for breach: stop use, seek damages, fines.
- Protects against infringement and allows recovery of damages.

Service Agreement

- Provider delivers services related to software for a fee.
- No transfer of ownership or copy of software.
- Focused on service delivery, not licensing.

Australia Privacy Act (continued)

- Same principles reiterated: public vs. private sector obligations.
- Cloud providers must comply with National Privacy Principles 4 and 9.

CLOUD CONTRACTING MODELS

License Agreement (expanded)

- Software used on licensee's site.
- Licensor retains ownership.
- License terms prohibit copying, unauthorized use, or resale.
- Breach remedies: refund, replace, or remove software.

Service Agreement (expanded)

- Provider delivers services, not copies of software.
- Software accessed via licensor's site.
- No transfer of ownership.
- No need to cover infringement risk.

Service Agreements in Cloud Arrangements

- SaaS, PaaS, IaaS are all service-based.
- Access provided as a service, not ownership.
- Service agreements protect cloud users without exposing providers to licensing risks.

On-Line Agreements vs. Standard Contracts

- **On-line agreements (click wrap):**
  - User clicks "I Agree."
  - Non-negotiable, most common model.
- **Standard contracts:**
  - Negotiated agreements.
  - Used for mission-critical applications.
  - More complex, tailored to customer needs.

Importance of Privacy Policies

- Privacy policy explains how provider handles user data.
- Must disclose protections and compliance with laws.
- If inadequate, users should consider other providers.
- FTC may audit privacy practices; discrepancies can lead to sanctions.

Privacy Policies & FTC Oversight

- Providers must disclose security and safety mechanisms.
- FTC audits can enforce compliance.
- Sanctions possible if practices differ from stated policies.

Risk Allocation & Limitations of Liability

- Agreements set maximum liability for claims.
- Risk often falls more on providers than users.
- Many contracts disclaim provider liability, even for negligence.
- Cloud users must carefully review contracts for liability clauses.
- Providers may assume responsibility through:
  - Security procedures
  - Standards/best practices
  - Next-gen security protocols
  - Employee training

Jurisdictional Issues Raised by Virtualization and Data Location

- Jurisdiction = court's authority over acts in a territory.
- Data location in cloud computing raises complex jurisdictional questions.

Virtualization and Multi-tenancy

Virtualization

- One physical server simulates multiple separate servers (virtual machines).
- Example: payroll, sales support, and asset management all run on one server.
- Benefits:
  - Less hardware needed
  - Lower power consumption
  - Better utilization of processing power
  - Reduced data center expenses
- Challenge: difficult to know exactly where information is housed at any given time.

Multi-tenancy

- Cloud provider delivers software as a service to multiple organizations from a single shared instance.
- Data is isolated, but software is shared.
- Risks:
  - One user's data may be accessed by another.
  - Backup and restore difficulties.

Flexibility of Data Location

- Providers move data among virtualized data centers to maximize efficiency.
- Same data may be hosted in multiple locations simultaneously (e.g., live storage in one region, backup in another).
- Agreements often don't specify where data is housed.
- Example: Amazon allows customers to choose U.S. or European data centers.

Issues Raised by Data Location

- **EU Data Protection Directive (Article 4):**
  - Once EU law applies to personal data, it remains subject to EU rules.
  - Transfers to third countries are limited.
- **Legal Conflicts:**
  - If contract specifies one country's law (e.g., Thailand), but data resides elsewhere (e.g., Poland), conflicts arise.
  - Applies whether storage is temporary or permanent.

Other Jurisdiction Issues

- **Confidentiality & Government Access:**
  - Each jurisdiction has its own rules for protecting data.
  - Governments may access data depending on local laws.
  - Example: USA PATRIOT Act allows law enforcement to access financial info, emails, and other data stored in the cloud with minimal certification.
- **Subcontracting:**
  - Providers may use third-party subcontractors.
  - Cloud users often cannot determine subcontractor location or responsibilities, creating risk.

International Conflicts of Laws

- Different countries' laws may conflict over the same subject matter.
- Sovereignty means each nation's laws apply within its territory.
- If contracts lack governing law clauses, or if public policy overrides them, courts decide which law applies.
- In cloud environments, conflicts are heightened by:
  - Cross-geographic virtualization
  - Multi-tenancy
  - Lack of transparency about data location
  - Subcontractor involvement

COMMERCIAL AND BUSINESS CONSIDERATIONS—A CLOUD USER'S VIEWPOINT

Minimizing Risk

- **Maintaining Data Integrity:**
  - Ensures data at rest is not corrupted.
  - Multi-tenancy creates efficiency but increases risks of corruption, contamination, or unauthorized access.

- Cloud users should demand contractual provisions obligating providers to protect data.
- Users must be entitled to remedies if data integrity is not maintained.

Accessibility and Availability of Data / SLAs

- SLA = provider's contractual performance commitment.
- Availability should be high (e.g., >99.7%) with minimal downtime.
- Users must understand provider's performance record.
- Remedies for SLA failure (credits, fees) are usually contractual.
- Many providers offer weak SLAs, leaving burden on users to prove failures.
- No law mandates SLAs; they are business-driven.
- Users must carefully review SLA terms to ensure compensation for failures.

Disaster Recovery

- Users should know provider's disaster recovery plan.
- Important for continuity if provider's data centers face disaster events.

Viability of the Cloud Provider

- Providers range from startups to global corporations.
- Viability matters because:
  - Users invest in integrating and migrating data.
  - Lack of standardization makes switching providers costly.
- Early-stage providers pose higher risks for users.

Special Topics

- **Open Cloud Manifesto (2009)**
  - Advocates openness in cloud computing.
  - Challenges: security, interoperability, portability, governance, monitoring.
  - Benefits: easier transitions, collaboration, integration, talent pool.
- **Litigation & e-Discovery**
  - Cloud complicates electronic discovery.
  - Agreements must address retention, preservation, disclosure timelines.
  - Cross-border discovery is complex; U.S. courts enforce compliance regardless of foreign laws.