Unit-2

CLOUD SERVICES AND FILE SYSTEM

Types of Cloud services

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

2.1 Software-as-a-Service

Software as a Service (SaaS) is defined as software that is deployed over the internet. With SaaS, a provider licenses an application to customers either as a service on demand through a subscription, in a ―pay-as-you-go‖ model, or (increasingly) at no charge when there is opportunity to generate revenue from streams other than the user, such as from advertisement or user list sales. SaaS is a rapidly growing market as indicated in recent reports that predict ongoing double digit growth.

SaaS is often referred to as software-on-demand and utilising it is akin to renting software rather than buying it. With traditional software applications you would purchase the software upfront as a package and then install it onto your computer. The software‘s licence may also

limit the number of users and/or devices where the software can be deployed. Software as a Service users, however, subscribe to the software rather than purchase it, usually on a monthly basis. Applications are purchased and used online with files saved in the cloud rather than on individual computers.

Applications

- Google
- Twitter
- Facebook
- Flickr

With users able to access the services via any internet enabled device. Enterprise users are able to use applications for a range of needs, including accounting and invoicing, tracking sales, planning, performance monitoring and communications including webmail and instant messaging.

Benefits

Some defining benefits of SaaS include;

- No additional hardware costs; the processing power required to run the applications is supplied by the cloud provider.
- No initial setup costs; applications are ready to use once the user subscribes.
- Pay for what you use; if a piece of software is only needed for a limited period then it is only paid for over that period and subscriptions can usually be halted at any time.
- Usage is scalable; if a user decides they need more storage or additional services, for example, then they can access these on demand without needing to install new software or hardware.
- Updates are automated; whenever there is an update it is available online to existing customers, often free of charge. No new software will be required as it often is with other types of applications and the updates will usually be deployed automatically by the cloud provider.

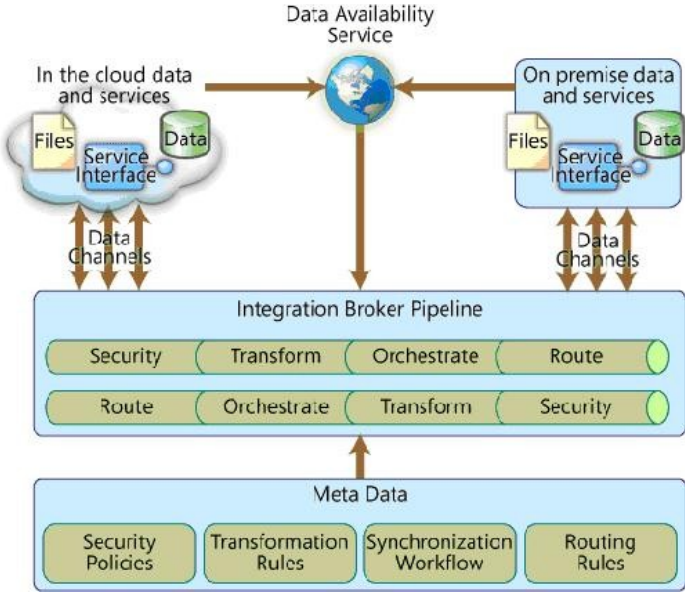
- Cross device compatibility; SaaS applications can be accessed via any internet enabled device, which makes it ideal for those who use a number of different devices, such as internet enabled phones and tablets, and those who don‘t always use the same computer.
- Accessible from any location; rather than being restricted to installations on individual computers, an application can be accessed from anywhere with an internet enabled device.
- Applications can be customized and white-labeled; with some software, customization is available meaning it can be altered to suit the needs and branding of a particular customer.

SaaS Integration Architecture

An integration broker is used to manage data movement and system integration.

Integration Broker

Many enterprises already are using some kind of integration broker for exposing application functions, orchestrating business processes, and integrating with internal backend systems. In many cases, the same integration broker can be customized and configured to perform integration and routing functions for a variety of internal and external data sources, including SaaS applications.



Data can originate from different sources, using different protocols and a variety of mutually incompatible formats. The job of the integration broker is to take data from a variety of

sources, determine how and where the data needs to be processed and routed, and send each piece of data to its destination in a form that the target system can use. The broker takes the form of a pipeline architecture to which you can add and remove modules that perform specific integration operations. Multiple logical pipelines can be used to process data traveling in different directions. In a typical case, for example, one pipeline would integrate data from sources on the Internet with local data sources, and another pipeline would take local data and integrate it with SaaS data on the Internet.

Data enters and exits the pipeline through data channels that define the protocols used to communicate with data sources. For example, one channel might be established to transmit data from a particular Web service to the broker using SOAP; another might transmit the data from the broker to a SaaS application using FTP. The modules plugged into the pipeline determine how the data is processed, routed, and integrated with data at the destination. A metadata service provides the configurable rules that each module uses to do its job. **Common integration operations** include the following:

- Security—Incoming data typically is processed by a security module, which performs operations such as authenticating the data source or digital signature, decrypting the data, and examining it for security risks, such as viruses. Security operations can be coordinated with existing security policies to control access.
- Validation—A validation module can compare the data to relevant schemas, and either reject noncompliant data or hand it off to a transformation component to be converted to the correct format.
- Synchronization workflow—A synchronization component uses workflow and rules to determine how data changes are propagated to destinations, and in what order. In cases where one of these workflow sequences cannot be completed successfully, the synchronization component can use transactional or compensation logic to "unwind" the data transfer gracefully, to guarantee data consistency across different systems.
- Routing—finally, routing rules define the destination for each piece of data. Routing might involve simply transmitting all data from a specific source to a designated target; or it might involve more complex logic, such as determining a destination from content information, such as a customer ID number.

A data availability service provides the means by which the integration broker can detect when new data is available.

Data-Availability Patterns

Synchronizing data involves transferring new and changed data from the source to the target (the data sink), either at regular intervals or when precipitated by an event. Three basic patterns are used to trigger data synchronization between a local source and a SaaS application:

- Polling—With polling, one source queries the other for changes, typically at regular intervals.
- Push—Push is the opposite of polling. In a push relationship, the source with the changed data communicates changes to the data sink. A data source can initiate a push every time data in a data source changes, or at regular intervals.
- Publish and subscribe—Event based publication and subscription is a hybrid approach that combines aspects of both polling and pushing. When a change is made to a data source, it publishes a change notification event, to which the data sink can subscribe.

The correct approach to use for detecting data changes can depend on a number of different factors, including whether data changes must be reflected at or near real time, and how many data sinks must be integrated with the data update. In some cases, one might have to seek a compromise that balances opposing interests. For example, a push approach is usually best for data that must always be kept up to date; but pushing data out to a large number of interested sources can be computationally and network intensive, and might degrade application performance. Whichever approach one chooses, he/she must develop rules to govern implementation details, such as polling frequency, syndication format, and so forth.

Data-Transfer Patterns

Data can be transferred between two endpoints using synchronous or asynchronous communication techniques.

- A synchronous transfer is akin to an interface: When one party requires information, it connects to the other party and requests it, expecting to receive the result immediately. This connection can take place in a variety of ways. Synchronous transfers can be simple file transfers, or they can take place through FTP, HTTP, or some other method.

- In an asynchronous transfer, the information can be transmitted by the sender and processed by the receiver at different times. Asynchronous transfers are typically message- based: One party sends a message to the other party requesting information, without expecting an immediate response. When the second party has processed the request, it sends a response back to the first party in another message. Messages can be sent by e-mail protocols such as SMTP, for example, or by message-queuing technologies.

Data-Transformation Patterns

Data transformation means taking data from one source, and altering its format and/or content so that it can be used by the data sink. Exchanging data with a SaaS application can involve some degree of data transformation. For example, one of the existing on-premise systems might exchange data using the EDIFACT standard, while the SaaS application are integrating, uses an incompatible XML-based format to send and receive data. Data emanating from an on-premise system must be transformed before it is sent to the SaaS application, and vice versa.

Transforming data is a multi-step process. Firstly, the incoming data should be validated against the appropriate data formats and schemas, to ensure that it will be usable after transformation. Optionally, the data can be enhanced by combining it with data from another source. Finally, the data itself is converted to the target format.

Identity Integration

Applications in multiple locations should be made accessible in a convenient and consistent way. One very significant component of this consistent user experience is single sign on: Users enter their user name and password when signing on to the Microsoft Windows operating system at the beginning of the day, and thereafter can access applications and network resources without having to present their credentials separately to each one. In addition to convenience, single sign-on means that users have fewer sets of credentials to keep track of, and reduces the security risk of lost or misplaced passwords.

From the IT management and governance perspective, single sign-on means that support staff will not have to manage independent sets of credentials. It also facilitates identity integration in other ways, such as enabling the reuse of existing application-access policies to control access to SaaS applications.

SaaS applications can provide single sign-on authentication through the use of a federation server within the customer's network that interfaces with the customer's own enterprise user-directory service. This federation server has a trust relationship with a corresponding federation server located within the SaaS provider's network. When an end user attempts to access the application, the enterprise federation server authenticates the user locally and negotiates with the SaaS federation server to provide the user with a signed security token, which the SaaS provider's authentication system accepts and uses to grant the user access.

Implementing a federation server that uses well-known standards for remote authentication, such as WS-Federation or Security Assertion Markup Language (SAML), will help to ease the process of implementing single sign-on with a wide range of SaaS providers.

2.2 Platform as a Service

Platform-as-a-Service (PaaS) solutions provide a development and deployment platform for running applications in the cloud. Platform as a Service (PaaS) brings the benefits that SaaS bought for applications, but over to the software development world. PaaS is analogous to SaaS except that, rather than being software delivered over the web, it is a platform for the creation of software, delivered over the web.

PaaS can be defined as a computing platform that allows the creation of web applications quickly and easily and without the complexity of buying and maintaining the software and infrastructure underneath it. They constitute the middleware on top of which applications are built.

PaaS makes the development, testing, and deployment of applications quick, simple, and cost-effective, eliminating the need to buy the underlying layers of hardware and software.

PaaS Features

- Operating system
- Server-side scripting environment
- Database management system
- Server Software
- Support

- Storage
- Network access
- Tools for design and development
- Hosting
- Services to develop, test, deploy, host and maintain applications in the same integrated development environment. All the varying services needed to fulfil the application development process
- Web based user interface creation tools help to create, modify, test and deploy different UI scenarios Multi-tenant architecture where multiple concurrent users utilize the same development application
- Built in scalability of deployed software including load balancing and failover
- Integration with web services and databases via common standards
- Support for development team collaboration – some PaaS solutions include project planning and communication tools
- Tools to handle billing and subscription management

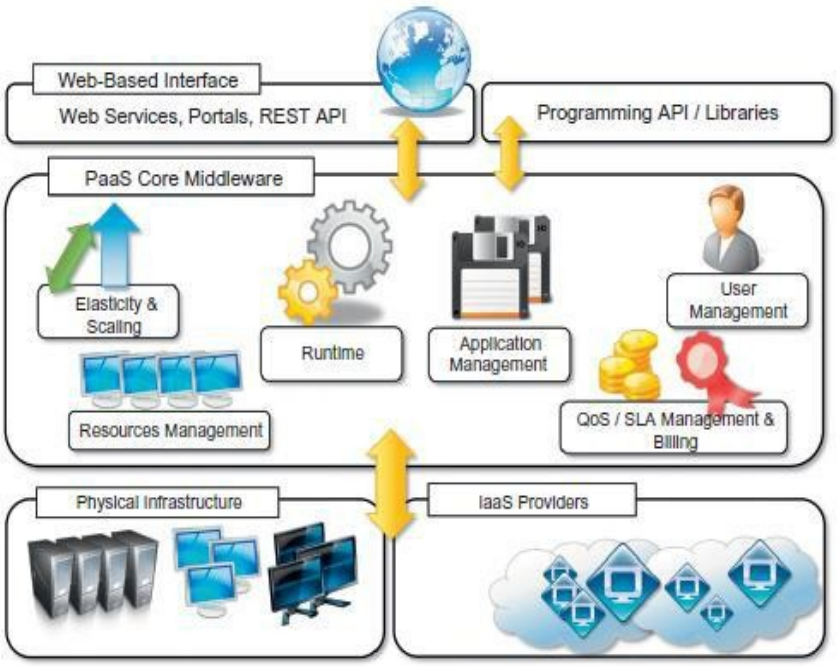
Paas Benefits

- Makes development possible for _non-experts‘; with some PaaS offerings anyone can develop an application. They can simply do this through their web browser utilizing one-click functionality. Salient examples of this are one-click blog software installs such as WordPress.
- Flexibility; customers can have control over the tools that are installed within their platforms and can create a platform that suits their specific requirements. They can _pick and choose‘ the features they feel are necessary.
- Adaptability; Features can be changed if circumstances dictate that they should.
- Teams in various locations can work together; as an internet connection and web browser are all that is required, developers spread across several locations can work together on the same application build.
- Security; security is provided, including data security and backup and recovery.

Applications

- Google App Engine
- Microsoft Azure Services
- Force.com platform.

Paas Architecture Model



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

From a user point of view, the core middleware exposes interfaces that allow programming and deploying applications on the cloud. These can be in the form of a Web-based interface or in the form of programming APIs and libraries.

The specific development model decided for applications determines the interface exposed to the user. Some implementations provide a completely Web-based interface hosted in the cloud and offering a variety of services. It is possible to find integrated developed environments based on 4GL and visual programming concepts, or rapid prototyping environments where applications are built by assembling mash-ups and user-defined components and successively customized. Other implementations of the PaaS model provide a complete object model for representing an application and provide a programming language-based approach. This approach generally offers more flexibility and opportunities but incurs longer development cycles.

Developers generally have the full power of programming languages such as Java, .NET, Python, or Ruby, with some restrictions to provide better scalability and security. In this case the traditional development environments can be used to design and develop applications, which are then deployed on the cloud by using the APIs exposed by the PaaS provider. Specific components can be offered together with the development libraries for better exploiting the services offered by the PaaS environment. Sometimes a local runtime environment that simulates the conditions of the cloud is given to users for testing their applications before deployment. This environment can be restricted in terms of features, and it is generally not optimized for scaling.

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

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

There are some essential characteristics that identify a PaaS solution

- Runtime Framework- This framework represents the ―software stack‖ of the PaaS model and the most intuitive aspect that comes to people‘s minds when they refer to PaaS solutions. The runtime framework executes end-user code according to the policies set by the user and the provider

- Abstraction- PaaS solutions are distinguished by the higher level of abstraction that they provide. Whereas in the case of IaaS solutions the focus is on delivering ―raw‖ access to virtual or physical infrastructure, in the case of PaaS the focus is on the applications the cloud must support. This means that PaaS solutions offer a way to deploy and manage applications on the cloud rather than a bunch of virtual machines on top of which the IT infrastructure is built and configured.
- Automation - PaaS environments automate the process of deploying applications to the infrastructure, scaling them by provisioning additional resources when needed. This process is performed automatically and according to the SLA made between the customers and the provider. This feature is normally not native in IaaS solutions, which only provide ways to provision more resources.
- Cloud Services- PaaS offerings provide developers and architects with services and APIs, helping them to simplify the creation and delivery of elastic and highly available cloud applications. These services are the key differentiators among competing PaaS solutions and generally include specific components for developing applications, advanced services for application monitoring, management, and reporting.

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

Even though a platform-based approach strongly simplifies the development and deployment cycle of applications, it poses the risk of making these applications completely dependent on the provider. Such dependency can become a significant obstacle in retargeting the application to another environment and runtime if the commitments made with the provider cease.

One of the major concerns of leveraging PaaS solutions for implementing applications is vendor lock-in. Differently from IaaS solutions, which deliver bare virtual servers that can be fully customized in terms of the software stack installed, PaaS environments deliver a platform for developing applications, which exposes a well-defined set of APIs and, in most cases, binds the application to the specific runtime of the PaaS provider. The impact of the vendor lock-in on

applications obviously varies according to the various solutions. Some of them, such as Force.com, rely on a proprietary runtime framework, which makes the retargeting process very difficult. Others, such as Google App Engine and Microsoft Azure, rely on industry-standard runtimes but utilize private data storage facilities computing infrastructure. In this case it is possible to find alternatives based on PaaS solutions implementing the same interfaces, with perhaps different performance. Others, such as Appistry Cloud IQ Platform, Heroku, and Engine Yard, completely rely on open standards, thus making the migration of applications easier.

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

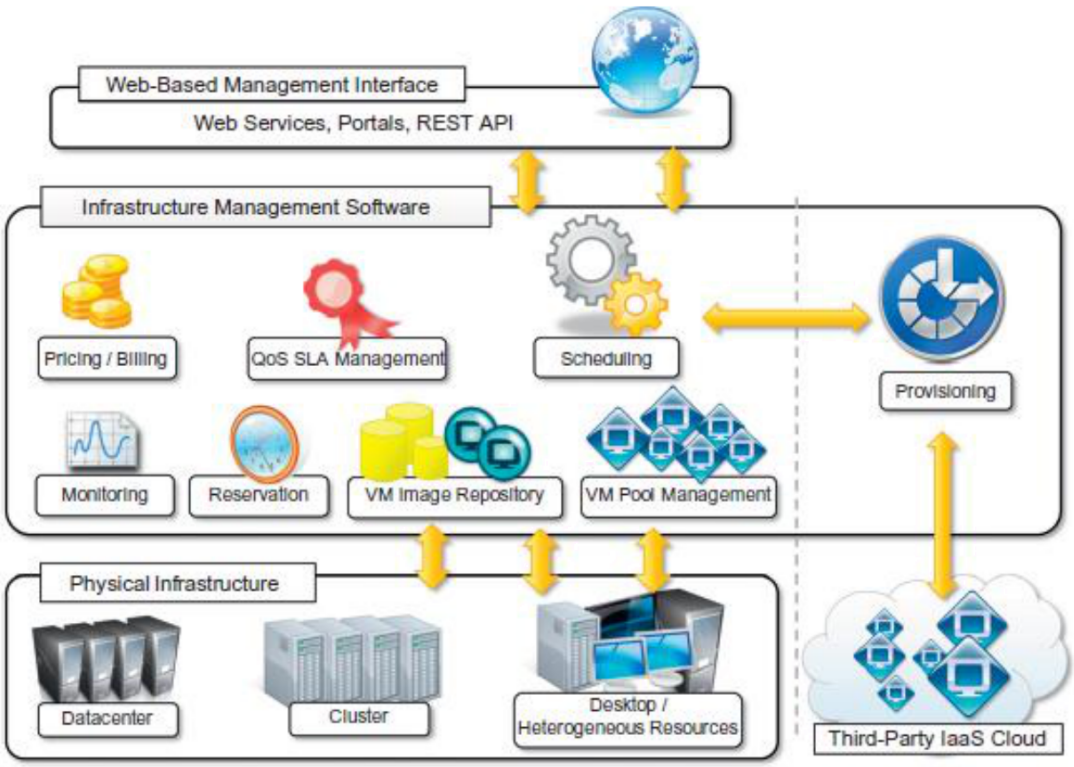
2.3 Infrastructure/ Hardware as a Service

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

The main technology used to deliver and implement these solutions is hardware virtualization: one or more virtual machines opportunely configured and interconnected define the distributed system on top of which applications are installed and deployed. Virtual machines also constitute the atomic components that are deployed and priced according to the specific features of the virtual hardware: memory, number of processors, and disk storage. IaaS/HaaS solutions bring all the benefits of hardware virtualization: workload partitioning, application isolation, sandboxing, and hardware tuning. From the perspective of the service provider, IaaS/HaaS allows better exploiting the IT infrastructure and provides a more secure environment where executing third party applications.

From the perspective of the customer it reduces the administration and maintenance cost as well as the capital costs allocated to purchase hardware. At the same time, users can take advantage of the full customization offered by virtualization to deploy their infrastructure in the cloud; in most cases virtual machines come with only the selected operating system installed and the system can be configured with all the required packages and applications. Other solutions provide prepackaged system images that already contain the software stack required for the most common uses: Web servers, database servers, or LAMP stacks.

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



Besides the basic virtual machine management capabilities, additional services can be provided, generally including the following: SLA resource-based allocation, workload management, support for infrastructure design through advanced Webinter- faces, and the ability to integrate third-party IaaS solutions.

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

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

The bottom layer is composed of the physical infrastructure, on top of which the management layer operates. As previously discussed, the infrastructure can be of different types; the specific infrastructure used depends on the specific use of the cloud. A service provider will most likely use a massive datacenter containing hundreds or thousands of nodes.

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

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

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

IaaS implementations provide computing resources, especially for the scheduling component. If storage is the main service provided, it is still possible to distinguish these three layers. The role of infrastructure management software is not to keep track and manage the execution of virtual machines but to provide access to large infrastructures and implement storage virtualization solutions on top of the physical layer.

2.4 Database as a Service

Database as a Service (DBaaS) is a cloud-based approach to the storage and management of structured data. DBaaS delivers database functionality similar to what is found in relational database management systems (RDBMSes) such as SQL Server, MySQL and Oracle. Being cloud-based, DBaaS provides a flexible, scalable, on-demand platform that's oriented toward self-service and easy management, particularly in terms of provisioning a business' own environment. DBaaS products typically provide enough monitoring capabilities to track performance and usage and to alert users to potential issues. The products can also generate at least some degree of data analytics.

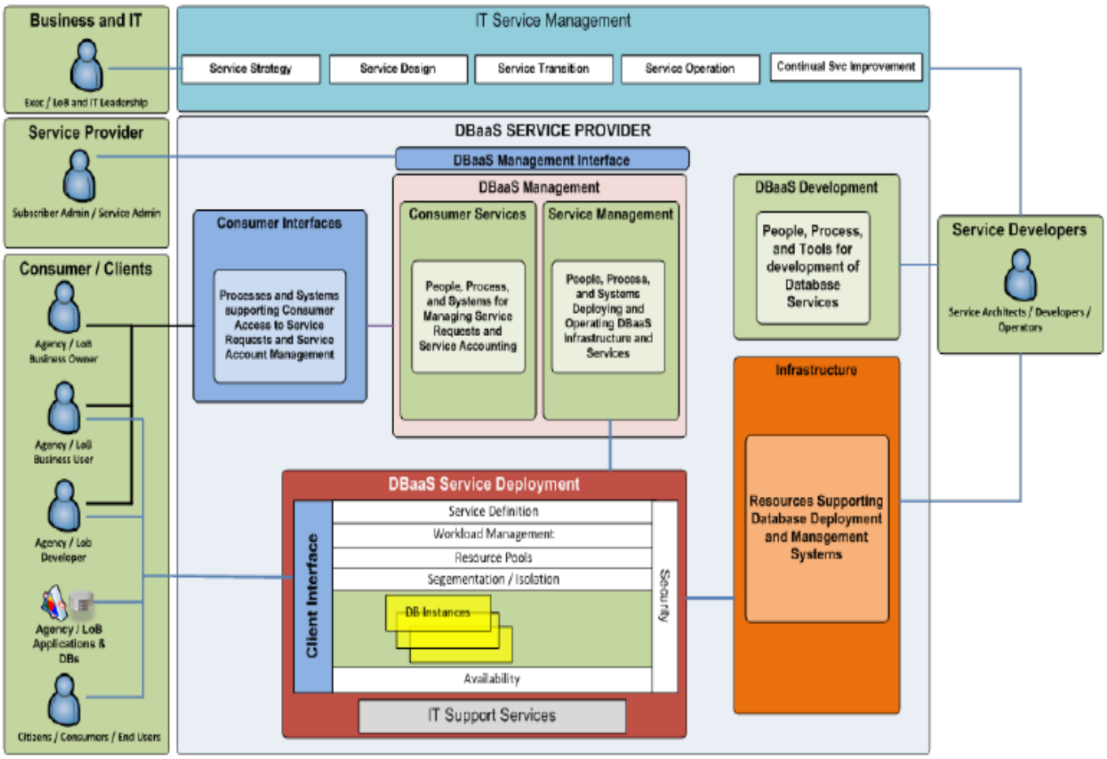
Benefits

- A shift from capital expense for hardware and software to operating expense for the database service. Companies can realize significant cost savings by purchasing database capacity and functionality as needed, and don‘t have to invest in advance of future requirements.
- Rapid or on-demand, self-service-based database provisioning. DBaaS allows for provisioning an environment in a very short period in contrast to days or weeks, thus reducing time to market.
- The ability of the customer to leverage existing servers and storage through automated resource management across standalone, clustered, virtualized and non-virtualized.
- The ability to outsource the administration and monitoring of databases such as backup, recovery, tuning, optimization, patching, upgrading and creation. Based on the policies that are defined by DBAs, database administration tasks can be automated — scheduled or proactively initiated to support various database activities.
- Granular metering of database usage that can be used for chargeback to various database users. Tracking is typically based on usage time, space, availability guarantees and resource consumption and provides an aggregated view per database.
- Freeing up of IT staff to focus on the logical administration of the database and the application data. The DBaaS provider provides a comprehensive database operating environment and a service-level agreement (SLA). Internal IT staffs don‘t have to build and

manage the physical environment, and physical database administration tasks are offloaded to the DBaaS provider.

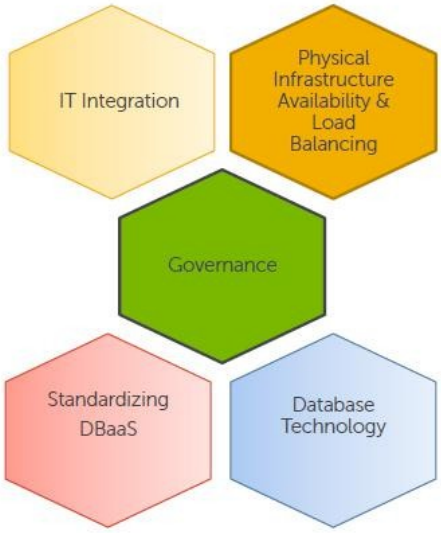
- Repurposing servers and storage. Servers and storage are often underutilized; DBaaS offers the ability to repurpose system resources more efficiently, resulting in significant cost savings.
- Support for faster application development and testing. DBaaS enables faster provisioning of new databases and automates the administration process, which helps organizations deliver database instances faster to developers, testers and architects.
- Improved availability for various applications. DBaaS can improve high availability of databases, especially for non-critical applications, by enabling failover of databases to available system resources. Typically these best practice architectures would not be cost effective in a dedicated environment.

The DBaaS conceptual model contains the core capabilities supporting the delivery of database services to an organization through a service-oriented and self-service driven model. The following is an overview of these capabilities and their relationships:



- DBaaS Development – The people, process and tools used to define the DBaaS service offerings, the designyy and implementation of the infrastructure required to support the DBaaS services offered, and the design and development of the systems and services required to deploy and administer the services within the DBaaS deployment architecture. DBaaS Development is responsible for defining the service offerings and management of the service catalog.
- DBaaS Management – The people, process and systems supporting the organization‘s ability to request, manage, operate, and account for database services and their utilization. Within DBaaS Management there are two sets of sub-capabilities: Subscriber Services which supports the interactions between the subscriber and delivery of database services, and Service Management which implements database services and manages the resources and systems supporting their delivery.
- DBaaS Service Deployment – The instantiated physical resources and their configuration in support of the DBaaS service offerings, and the interfaces required to manage the deployment, monitoring, and management of the services. This would include the configuration of servers, networking, and software to support the specific database deployment models and database instances within the shared resource pool.
- Infrastructure – The physical resources required to support the systems and services supporting the management and deployment of the DBaaS architecture. This would include servers, networking, software and facilities.
- IT Service Management – Is the framework for service definition, design and operational practices and policies. IT Service Management provides capabilities for defining and managing the services required by the organization, the design process, change management process definition, service operation policy structure, and process improvement framework.
- Subscriber Interfaces – The methods, systems and procedures for interacting with the DBaaS management capabilities as well as the database service instances deployed for the specific subscriber.

Challenges



2.5 Monitoring as a Service

MaaS is a framework that facilitates the deployment of monitoring functionalities for various other services and applications within the cloud. The most common application for MaaS is online state monitoring, which continuously tracks certain states of applications, networks, systems, instances or any element that may be deployable within the cloud.

Monitoring as a Service (MaaS) in the Cloud is a concept that combines the benefits of cloud computing technology and traditional on-premise IT infrastructure monitoring solutions. MaaS is a new delivery model that is suited for organizations looking to adopt a monitoring framework quickly with minimal investments.

On premise monitoring is the traditional deployment model for monitoring private networks (internal IT infrastructure). This has been a very effective model over the years and works well for organization that can afford to implement this monitoring framework. On-premise monitoring involves purchase of software tools and investing in monitoring infrastructure and skilled IT personnel.

MaaS offerings consist of multiple tools and applications meant to monitor a certain aspect of an application, server, system or any other IT component. There is a need for proper data collection, especially of the performance and real-time statistics of IT components, in order to

make proper and informed management possible. Monitoring system is therefore required to alert the cloud user about the current state of their infrastructure. Also MaaS would also deal with the predictive analysis of their source utilization on the cloud server.

MaaS is capable of monitoring all aspects of IT infrastructure assets:

- **Servers and Systems Monitoring:** Server Monitoring provides insights into the reliability of the server hardware such as Uptime, CPU, Memory and Storage. Server monitoring is an essential tool in determining functional and performance failures in the infrastructure assets.
- **Database Monitoring:** Database monitoring on a proactive basis is necessary to ensure that databases are available for supporting business processes and functions. Database monitoring also provides performance analysis and trends which in turn can be used for fine tuning the database architecture and queries, thereby optimizing the database for your business requirements.
- **Network Monitoring:** Network availability and network performance are two critical parameters that determine the successful utilization of any network – be it a LAN, MAN or WAN network. Disruptions in the network affect business productivity adversely and can bring regular operations to a standstill. Network monitoring provides pro-active information about network performance bottlenecks and source of network disruption.
- **Storage Monitoring:** A reliable storage solution in your network ensures anytime availability of business critical data. Storage monitoring for SAN, NAS and RAID storage devices ensures that your storage solution are performing at the highest levels. Storage monitoring reduces downtime of storage devices and hence improves availability of business data.
- **Applications Monitoring:** Applications Monitoring provides insight into resource usage, application availability and critical process usage for different Windows, Linux and other open source operating systems based applications. Applications Monitoring is essential for mission critical applications that cannot afford to have even a few minutes of downtime. With Application Monitoring, you can prevent application failures before they occur and ensure smooth operations.
- **Cloud Monitoring:** Cloud Monitoring for any cloud infrastructure such as Amazon or Rackspace gives information about resource utilization and performance in the cloud. While

cloud infrastructure is expected to have higher reliability than on-premise infrastructure, quite often resource utilization and performance metrics are not well understood in the cloud. Cloud monitoring provides insight into exact resource usage and performance metrics that can be used for optimizing the cloud infrastructure.

- **Virtual Infrastructure Monitoring:** Virtual Infrastructure based on common hypervisors such as ESX, Xen or Hyper-V provides flexibility to the infrastructure deployment and provides increased reliability against hardware failures. Monitoring virtual machines and related infrastructure gives information around resource usage such as memory, processor and storage.

Benefits

The following are the benefits of a monitoring as a service (MaaS):

1) **Ready to Use Monitoring Tool Login:** The vendor takes care of setting up the hardware infrastructure, monitoring tool, configuration and alert settings on behalf of the customer. The customer gets a ready to use login to the monitoring dashboard that is accessible using an internet browser. A mobile client is also available for the MaaS dashboard for IT administrators.
2) **Inherently Available 24x7x365:** Since MaaS is deployed in the cloud, the monitoring dashboard itself is available 24x7x365 that can be accessed anytime from anywhere. There are no downtimes associated with the monitoring tool.
3) **Easy Integration with Business Processes:** MaaS can generate alert based on specific business conditions. MaaS also supports multiple levels of escalation so that different user groups can get different levels of alerts.
4) **Cloud Aware and Cloud Ready:** Since MaaS is already in the cloud, MaaS works well with other cloud based products such as PaaS and SaaS. MaaS can monitor Amazon and Rackspace cloud infrastructure. MaaS can monitor any private cloud deployments that a customer might have.
5) **Zero Maintenance Overheads:** As a MaaS, customer, you don‘t need to invest in a network operations centre. Neither do you need to invest an in-house team of qualified IT engineers to run the monitoring desk since the MaaS vendor is doing that on behalf of the customer

When to Use Monitoring as a Service (MaaS)?

Monitoring as a service (MaaS) is an attractive choice for the following scenarios:

- **Price Sensitive Customers:** For small and medium enterprises, MaaS provides cost effective pay per use pricing model. Customers don‘t need to make any heavy investments neither in capital expenditures (capex) nor in operating expenditures (opex).
- **Cloud Based SaaS and PaaS offering Add-On:** MaaS provides a better technology fit for monitoring cloud based SaaS and PaaS offerings. MaaS can be provided as an add-on product offering along with SaaS and PaaS.
- **Distributed Infrastructure Assets:** In scenarios where the IT infrastructure assets are distributed across different locations and branch offices, MaaS is a good option since the monitoring infrastructure is centralized in the cloud and can easily monitor all distributed infrastructure assets.
- **Mixture of Cloud and On-Premise Infrastructure:** MaaS is already in the cloud. Hence in deployments where customer has a mix of on-premise and cloud infrastructure, MaaS provides good monitoring options for the hybrid environment.
- **Multitenant Monitoring Requirements:** For vendors offering multi-tenant functionality on their hosted services, MaaS provides a strong backend framework for monitoring the multi-tenant services and their availability.

2.6 Communication as a Service

Communications as a Service (CaaS) is an outsourced enterprise communications solution that can be leased from a single vendor. This is communications functionality that may include telephony, messaging, conferencing, presence and notification, based on assets owned, managed and colocated by third parties. Such communications can include voice over IP (VoIP or Internet telephony), instant messaging (IM), collaboration and video conference applications using fixed and mobile devices. CaaS has evolved along the same lines as Software as a Service (SaaS).

The CaaS vendor is responsible for all hardware and software management and offers guaranteed Quality of Service (QoS). CaaS allows businesses to selectively deploy communications devices and modes on a pay-as-you-go, as-needed basis. This approach

eliminates the large capital investment and ongoing overhead for a system whose capacity may often exceed or fall short of current demand.

CaaS offers flexibility and expandability that small and medium-sized business might not otherwise afford, allowing for the addition of devices, modes or coverage on demand. The network capacity and feature set can be changed from day to day if necessary so that functionality keeps pace with demand and resources are not wasted. There is no risk of the system becoming obsolete and requiring periodic major upgrades or replacement.

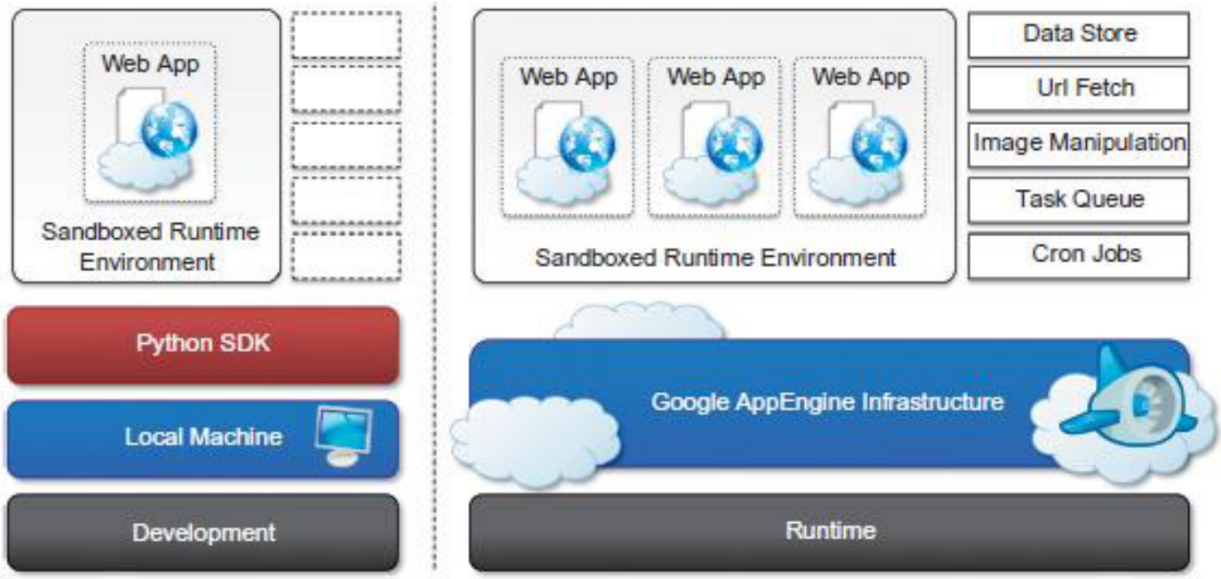
With the service provider responsible for the management and running of these services also, the other advantage the consumer has is that they needn‘t require their own trained personnel, bringing significant OPEX as well as CAPEX costs.

2.7 Service Providers

Google App Engine

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

applications in Java, Python, and Go, a new programming language developed by Google to simplify the development of Web applications. Application usage of Google resources and services is metered by App Engine, which bills users when their applications finish their free quotas.



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

Infrastructure

App Engine hosts Web applications, and its primary function is to serve users requests efficiently. To do so, App Engine‘s infrastructure takes advantage of many servers available within Google datacenters.

For each HTTP request, App Engine locates the servers hosting the application that processes the request, evaluates their load, and, if necessary, allocates additional resources (i.e., servers) or redirects the request to an existing server. The particular design of applications, which does not expect any state information to be implicitly maintained between requests to the same application, simplifies the work of the infrastructure, which can redirect each of the requests to any of the servers hosting the target application or even allocate a

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

Runtime Environment

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

Sandboxing

One of the major responsibilities of the runtime environment is to provide the application environment with an isolated and protected context in which it can execute without causing a threat to the server and without being influenced by other applications. In other words, it provides applications with a sandbox. Currently, App Engine supports applications that are developed only with managed or interpreted languages, which by design require a runtime for translating their code into executable instructions. Therefore, sandboxing is achieved by means of modified runtimes for applications that disable some of the common features normally available with their default implementations.

Supported runtimes

Currently, it is possible to develop App Engine applications using three different languages and related technologies:

- **Java-** App Engine currently supports Java 6, and developers can use the common tools for Web application development in Java, such as the Java Server Pages (JSP), and the applications interact with the environment by using the Java Servlet standard. Furthermore, access to App Engine services is provided by means of Java libraries that expose specific interfaces of provider-specific implementations of a given abstraction layer.
- **Python-** Support for Python is provided by an optimized Python 2.5.2 interpreter. As with Java, the runtime environment supports the Python standard library, but some of the modules that implement potentially harmful operations have been removed, and attempts

to import such modules or to call specific methods generate exceptions. To support application development, App Engine offers a rich set of libraries connecting applications to App Engine services. In addition, developers can use a specific Python Web application framework, called web app, simplifying the development of Web applications.

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

Storage

App Engine provides various types of storage, which operate differently depending on the volatility of the data. There are three different levels of storage: in memory-cache, storage for semi-structured data, and long-term storage for static data.

Static file servers

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

Data Store

Data Store is a service that allows developers to store semi-structured data. The service is designed to scale and optimized to quickly access data. Data Store can be considered as a large object database in which to store objects that can be retrieved by a specified key. Both the type

of the key and the structure of the object can vary. The underlying infrastructure of Data Store is based on Bigtable, a redundant, distributed, and semi-structured data store that organizes data in the form of tables. Developers define their data in terms of entity and properties, and these are persisted and maintained by the service into tables in Bigtable.

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

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

Application Services

- **UrlFetch-** The sandbox environment does not allow applications to open arbitrary connections through sockets, but it does provide developers with the capability of retrieving a remote resource through HTTP/HTTPS by means of the UrlFetch service.
- **MemCache-** AppEngine provides caching services by means of MemCache. This is a distributed in-memory cache that is optimized for fast access and provides developers with a volatile store for the objects that are frequently accessed. The caching algorithm implemented by MemCache will automatically remove the objects that are rarely accessed. The use of MemCache can significantly reduce the access time to data; developers can

structure their applications so that each object is first looked up into MemCache and if there is a miss, it will be retrieved from DataStore and put into the cache for future lookups.

- **Mail and instant messaging**- App Engine provides also another way to communicate with the external world: the Extensible Messaging and Presence Protocol (XMPP). Any chat service that supports XMPP, such as Google Talk, can send and receive chat messages to and from the Web application, which is identified by its own address. Even though the chat is a communication medium mostly used for human interactions, XMPP can be conveniently used to connect the Web application with chat bots or to implement a small administrative console.
- **Account management-** Web applications often keep various data that customize their interaction with users. These data normally go under the user profile and are attached to an account. App Engine simplifies account management by allowing developers to leverage Google account management by means of Google Accounts. The integration with the service also allows Web applications to offload the implementation of authentication capabilities to Google‘s authentication system. Using Google Accounts, Web applications can conveniently store profile settings in the form of key-value pairs, attach them to a given Google account, and quickly retrieve them once the user authenticates.
- **Image manipulation**- Web applications render pages with graphics. Often simple operations, such as adding watermarks or applying simple filters, are required. AppEngine allows applications to perform image resizing, rotation, mirroring, and enhancement by means of Image Manipulation, a service that is also used in other Google products. Image Manipulation is mostly designed for lightweight image processing and is optimized for speed.

2.8 Amazon EC2

This is an Infrastructure as a Service offering. Amazon EC2 allows deploying servers in the form of virtual machines created as instances of a specific image. Images come with a preinstalled operating system and a software stack, and instances can be con- figured for memory, number of processors, and storage. Users are provided with credentials to remotely access the instance and further configure or install software if needed.

Customer segments (CS)

In the Business Model Canvas, ―Customer Segments‖ are the groups of customers that the company ultimately serves, I.e. the ones that consume and pay for the services. In the AWS case, although basically anybody with a credit card can spin up a virtual machine, it looks like Amazon is primarily targeting software developers and (startup) SaaS providers as the main customers. Historically, the Amazon development teams were the first customers. External customers were initially added as an afterthought.

Value Propositions (VP)

The value propositions reflect the customer problems and needs. This is the central element that describes why the customer would ultimately pay for the product or service.The value proposition of cloud computing centers around its five essential characteristics. For example in the AWS EC2 case, the core component of the value proposition is rapid self-service provisioning of virtual machines with pay per use billing. For each individual customer these translate into different business advantages. An example is reduced capital expenditure and reduced risk of over-investing or under-provisioning.

Channels (CH)

Value propositions are delivered to customers through communications, distribution and sales channels. It is often assumed that cloud computing relies solely self-service direct sales, but the reality is much more diverse. SaaS providers in particular are developing extensive partner programs.

AWS primarily employs a self-service direct model, where the delivery is through APIs. AWS also provides a web user interface to those APIs. Interestingly, that interface used to lag in functionality behind the main AWS services, but these days most new features are announced on the API and the Web UI simultaneously. The model is enhanced by premium support.

Customer Relationships (CR)

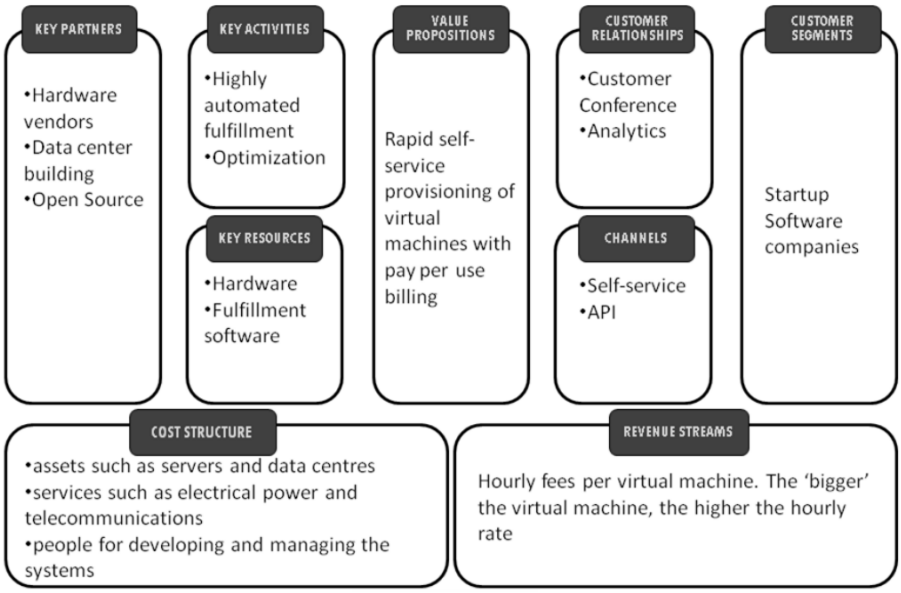
Customer relations are established and maintained with each specific customer segment. One of the ways that AWS maintains relationships with its customer segments is through conferences. The 2013 re:Invent developer conference attracted 9000 visitors. Additionally, there are vibrant on-line communities. Thus AWS does extensive analytics on the activity that customers have on the platform.

Revenue Streams (RS)

Revenue streams are the result of value propositions that are successfully offered to customers.The structure of revenue streams is where cloud computing differs from earlier IT service models, as they are usage based rather than asset based.AWS basically charges hourly fees per virtual machine. The _bigger‘ the virtual machine, the higher the hourly rate.

Key Resources (KR)

Key resources are the assets required to offer and deliver the previously mentioned elements (e.g. value proposition, customer relationships). AWS owns massive amounts of hardware, estimated at 1 million servers or more. That is housed in dozens of data-centers worldwide. But there is more. The service can only be delivered through advanced and unique fulfillment software and processes. Amazon must have invested very substantially in that.



Key Activities (KA)

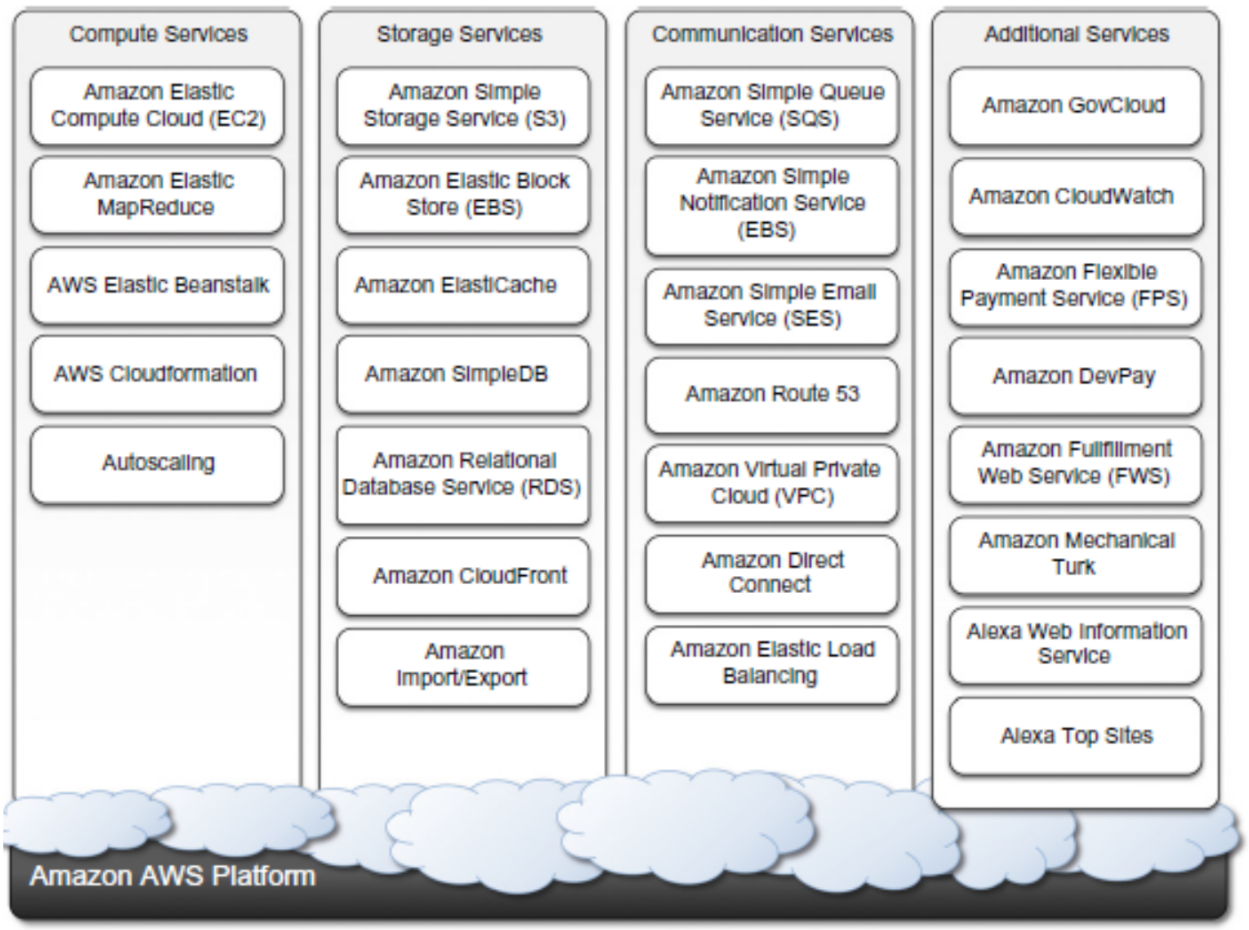
The key resources perform key activities. At AWS the key activity, delivery, is highly automated. But at the AWS scale, oversight and resources planning is still a serious effort. Optimizing assets versus utilization is very essential in the IaaS business model. Through economies of scale, AWS is able to spend a lot of effort on these activities.

Key Partnerships (KP)

Some activities are outsourced, and some resources are acquired outside the enterprise. AWS buys immense amounts of hardware, and uses a lot of (open source) software. Building out data centers is also likely to be outsourced.

Cost Structure (CS)

All business model elements result in a cost structure. In more traditional IT service models the revenue streams are tightly coupled to the cost structure. The cloud computing innovation is also about decoupling these. At AWS the main cost elements are in assets such as servers and data centers, in services such as electrical power and telecommunications, and in people for eveloping and managing the systems.



EC2 Instances

EC2 (Elastic Compute Cloud) instances represent virtual machines. They are created using Amazon Machine Images (AMI) as templates, which are specialized by selecting the number of cores, their computing power, and the installed memory. EC2 instances can be run either by using the command-line tools provided by Amazon, which connects the Amazon Web Service that provides remote access to the EC2 infrastructure, or via the AWS console, which allows the management of other services, such as Simple Storage Service (S3). By default an EC2 instance is created with the kernel and the disk associated to the AMI.

The processing power is expressed in terms of virtual cores and EC2 Compute Units (ECUs). The ECU is a measure of the computing power of a virtual core; it is used to express a predictable quantity of real CPU power that is allocated to an instance. By using compute units instead of real frequency values, Amazon can change over time the mapping of such units to the underlying real amount of computing power allocated, thus keeping the performance of EC2 instances consistent with standards set by the times. Over time, the hardware supporting the underlying infrastructure will be replaced by more powerful hardware, and the use of ECUs helps give users a consistent view of the performance offered by EC2 instances.

- Standard instances: This class offers a set of configurations that are suitable for most applications. EC2 provides three different categories of increasing computing power, storage, and memory.
- Micro instances: This class is suitable for those applications that consume a limited amount of computing power and memory and occasionally need bursts in CPU cycles to process urges in the workload. Micro instances can be used for small Web applications with limited traffic.
- High-memory instances. This class targets applications that need to process huge workloads and require large amounts of memory. Three-tier Web applications characterized by high traffic are the target profile. Three categories of increasing memory and CPU are available, with memory proportionally larger than computing power.
- High-CPU instances. This class targets compute-intensive applications. Two configurations are available where computing power proportionally increases more than memory.

- Cluster Compute instances. This class is used to provide virtual cluster services. Instances in this category are characterized by high CPU compute power and large memory and an extremely high I/O and network performance, which makes it suitable for High Processing Computing (HPC) applications.
- Cluster GPU instances. This class provides instances featuring graphic processing units (GPUs) and high compute power, large memory, and extremely high I/O and network performance. This class is particularly suited for cluster applications that perform heavy graphic computations, such as rendering clusters. Since GPU can be used for general-purpose computing, users of such instances can benefit from additional computing power, which makes this class suitable for HPC applications.

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

Storage Services

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

A bucket is a container of objects. It can be thought of as a virtual drive hosted on the S3 distributed storage, which provides users with a flat store to which they can add objects. Buckets are top-level elements of the S3 storage architecture and do not support nesting. A bucket is located in a specific geographic location and eventually replicated for fault tolerance and better content distribution. Users can select the location at which to create buckets, which by default are created in Amazon‘s U.S. datacenters. Once a bucket is created, all the objects that belong to the bucket will be stored in the same availability zone of the bucket.

Amazon elastic block store

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

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

Amazon ElastiCache

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

ElastiCache cluster can be dynamically resized according to the demand of the client applications. Furthermore, automatic patch management and failure detection and recovery of cache nodes allow the cache cluster to keep running without administrative intervention from AWS users, who have only to elastically size the cluster when needed. ElastiCache nodes are priced according to the EC2 costing model, with a small price difference due to the use of the caching service installed on such instances.

Amazon SimpleDB

Amazon SimpleDB is a lightweight, highly scalable, and flexible data storage solution for applications that do not require a fully relational model for their data. SimpleDB provides support for semi structured data, the model for which is based on the concept of domains, items, and attributes.

Amazon CloudFront

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

Communication services

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

- Virtual Networking

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

Amazon Direct Connect allows AWS users to create dedicated networks between the user private network and Amazon Direct Connect locations, called ports. This connection can be further partitioned in multiple logical connections and give access to the public resources hosted

on the Amazon infrastructure. The advantage of using Direct Connect versus other solutions is the consistent performance of the connection between the users‘ premises and the Direct Connect locations. This service is compatible with other services such as EC2, S3, and Amazon VPC and can be used in scenarios requiring high bandwidth between the Amazon network and the outside world.

- Messaging

Messaging services constitute the next step in connecting applications by leveraging AWS capabilities. The three different types of messaging services offered are

- Amazon Simple Queue Service (SQS) - Amazon SQS constitutes disconnected model for exchanging messages between applications by means of message queues, hosted within the AWS infrastructure. Using the AWS console or directly the underlying Web service AWS, users can create an unlimited number of message queues and configure them to control their access. Applications can send messages to any queue they have access to. These messages are securely and redundantly stored within the AWS infrastructure for a limited period of time, and they can be accessed by other (authorized) applications. While a message is being read, it is kept locked to avoid spurious processing from other applications. Such a lock will expire after a given period.
- Amazon Simple Notification Service (SNS) - Amazon SNS provides a publish-subscribe method for connecting heterogeneous applications. With respect to Amazon SQS, where it is necessary to continuously poll a given queue for a new message to process, Amazon SNS allows applications to be notified when new content of interest is available. This feature is accessible through a Web service whereby AWS users can create a topic, which other applications can subscribe to. At any time, applications can publish content on a given topic and subscribers can be automatically notified. The service provides subscribers with different notification models (HTTP/HTTPS, email/email JSON, and SQS).
- Amazon Simple Email Service (SES) - Amazon SES provides AWS users with a scalable email service that leverages the AWS infra- structure. Once users are signed up for the service, they have to provide an email that SES will use to send emails on their behalf. To activate the service, SES will send an email to verify the given address and provide the users with the necessary information for the activation. Upon verification, the user is

given an SES sandbox to test the service, and he can request access to the production version. Using SES, it is possible to send either SMTP-compliant emails or raw emails by specifying email headers and Multipurpose Internet Mail Extension (MIME) types. Emails are queued for delivery, and the users are notified of any failed delivery. SES also provides a wide range of statistics that help users to improve their email campaigns for effective communication with customers.

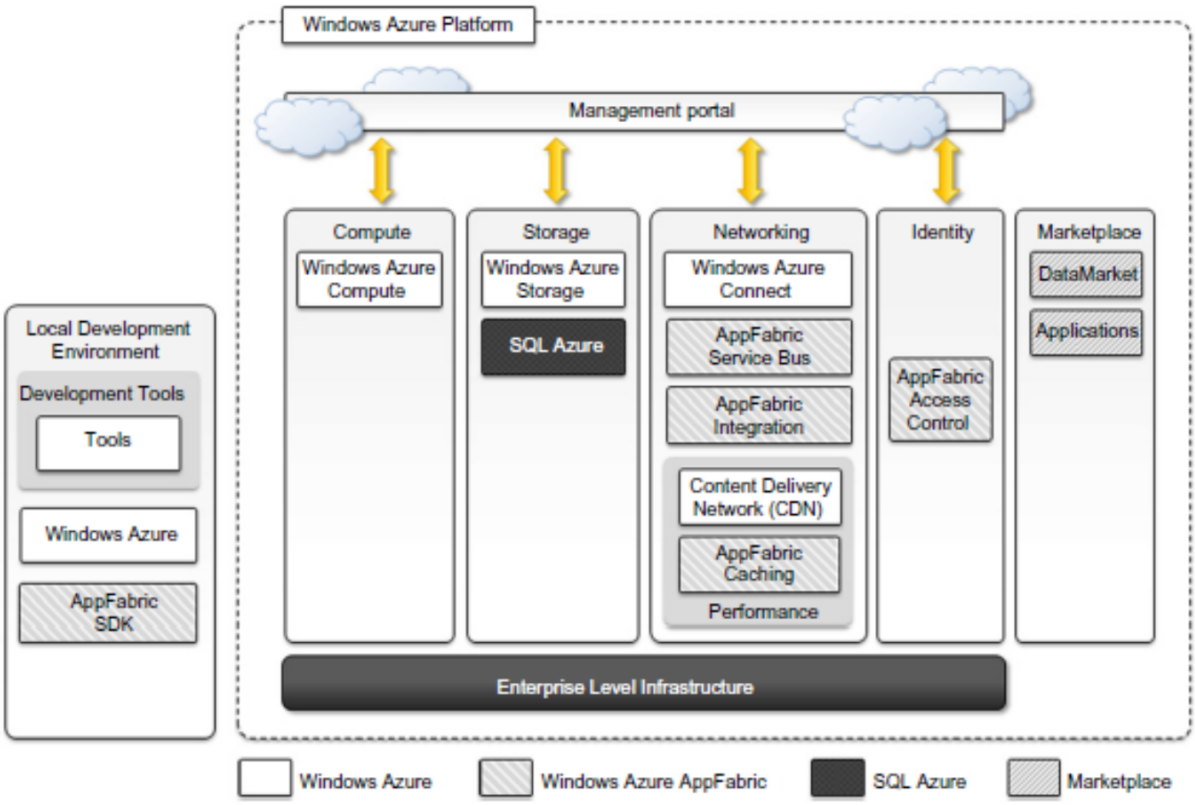
Additional Services

- Amazon CloudWatch is a service that provides a comprehensive set of statistics that help devel- opers understand and optimize the behavior of their application hosted on AWS. CloudWatch col- lects information from several other AWS services: EC2, S3, SimpleDB, CloudFront, and others. Using CloudWatch, developers can see a detailed breakdown of their usage of the service they are renting on AWS and can devise more efficient and cost-saving applications. Earlier services of CloudWatch were offered only through subscription, but now it is made available for free to all the AWS users.
- Amazon Flexible Payment Service (FPS) infrastructure allows AWS users to leverage Amazon‘s billing infrastructure to sell goods and services to other AWS users. Using Amazon FPS, developers do not have to set up alternative payment methods, and they can charge users via a billing service. The payment models available through FPS include one-time payments and delayed and periodic payments, required by subscriptions and usage-based services, transactions, and aggregate multiple payments.

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

2.9 Microsoft Azure

Microsoft Azure is a cloud operating system and a platform for developing applications in the cloud. It provides a scalable runtime environment for Web applications and distributed applications in general. Applications in Azure are organized around the concept of roles, which identify a distribution unit for applications and embody the application‘s logic. Currently, there are three types of role: Web role, worker role, and virtual machine role. The Web role is designed to host a Web application, the worker role is a more generic container of applications and can be used to perform workload processing, and the virtual machine role provides a virtual environment in which the computing stack can be fully customized, including the operating systems. Besides roles, Azure provides a set of additional services that complement application execution, such as support for storage (relational data and blobs), networking, caching, content delivery, and others.



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

Web role

The Web role is designed to implement scalable Web applications. Web roles represent the units of deployment of Web applications within the Azure infrastructure. They are hosted on the IIS 7 Web Server, which is a component of the infrastructure that supports Azure. When Azure detects peak loads in the request made to a given application, it instantiates multiple Web roles for that application and distributes the load among them by means of a load balancer. Since version 3.5, the .NET technology natively supports Web roles; developers can directly develop their applications in Visual Studio, test them locally, and upload to Azure. It is possible to develop ASP.NET (ASP.NET Web Role and ASP.NET MVC 2 Web Role) and WCF (WCF Service Web Role) applications. Since IIS 7 also supports the PHP runtime environment by means of the Fast CGI module, Web roles can be used to run and scale PHP Web applications on Azure (CGI Web Role). Other Web technologies that are not integrated with IIS can still be hosted on Azure (i.e., Java Server Pages on Apache Tomcat), but there is no advantage to using a Web role over a Worker role.

Worker role

Worker roles are designed to host general compute services on Azure. They can be used to quickly provide compute power or to host services that do not communicate with the external world through HTTP. A common practice for Worker roles is to use them to provide background processing for Web applications developed with Web roles. Developing a worker role is like a developing a service. Compared to a Web role whose computation is triggered by the interaction with an HTTP client (i.e., a browser), a Worker role runs continuously from the creation of its instance until it is shut down. The Azure SDK provides developers with convenient APIs and libraries that allow connecting the role with the service provided by the runtime and easily controlling its startup as well as being notified of changes in the hosting environment. As with

Web roles, the .NET technology provides complete support for Worker roles, but any technology that runs on a Windows Server stack can be used to implement its core logic. For example, Worker roles can be used to host Tomcat and serve JSP-based applications.

Virtual machine role

The Virtual Machine role allows developers to fully control the computing stack of their compute service by defining a custom image of the Windows Server 2008 R2 operating system and all the service stack required by their applications. The Virtual Machine role is based on the Windows Hyper-V virtualization technology, which is natively integrated in the Windows server technology at the base of Azure. Developers can image a Windows server installation complete with all the required applications and components, save it into a Virtual Hard Disk (VHD) file, and upload it to Windows Azure to create compute instances on demand.

Storage solutions

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

Blobs

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

- **Block blobs-** Block blobs are composed of blocks and are optimized for sequential access; therefore they are appropriate for media streaming. Currently, blocks are of 4 MB, and a single block blob can reach 200 GB in dimension.

- **Page blobs**- Page blobs are made of pages that are identified by an offset from the beginning of the blob. A page blob can be split into multiple pages or constituted of a single page. This type of blob is optimized for random access and can be used to host data different from streaming. Currently, the maximum dimension of a page blob can be 1 TB. Blobs storage

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

Azure drive

Page blobs can be used to store an entire file system in the form of a single Virtual Hard Drive (VHD) file. This can then be mounted as a part of the NTFS file system by Azure compute resources, thus providing persistent and durable storage. A page blob mounted as part of an NTFS tree is called an Azure Drive.

Tables

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

Queues

Queue storage allows applications to communicate by exchanging messages through durable queues, thus avoiding lost or unprocessed messages. Applications enter messages into a queue, and other applications can read them in a first-in, first-out (FIFO) style. To ensure that messages get processed, when an application reads a message it is marked as invisible; hence it will not be available to other clients. Once the application has completed processing the message, it needs to explicitly delete the message from the queue. This two-phase process ensures that messages get processed before they are removed from the queue, and the client failures do not prevent messages from being processed. At the same time, this is also a reason that the queue does not enforce a strict FIFO model: Messages that are read by applications that crash during processing are made available again after a timeout, during which other messages can be read by

other clients. An alternative to reading a message is peeking, which allows retrieving the message but letting it stay visible in the queue. Messages that are peeked are not considered processed.

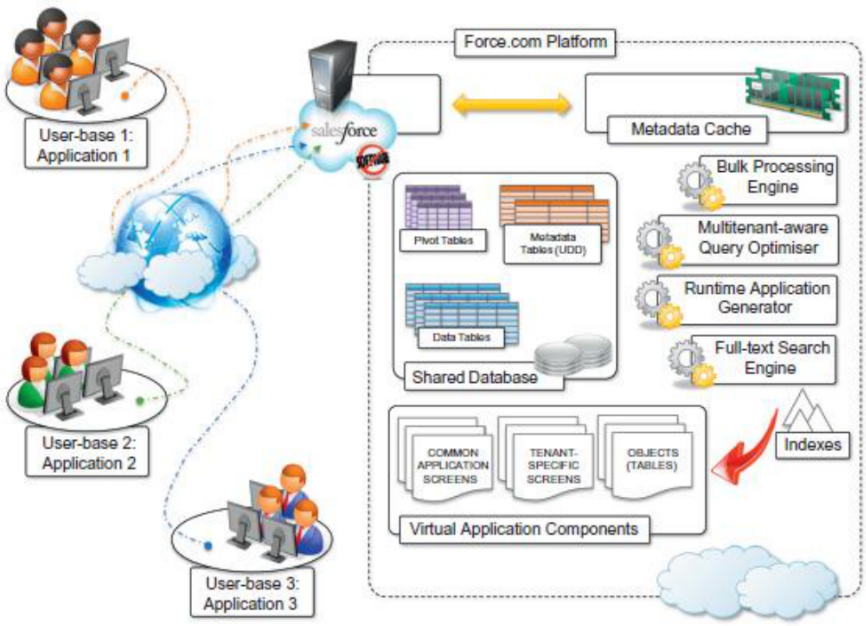
2.10 Salesforce.com

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

As of today more than 100,000 customers have chosen Safesforce.com to implement their CRM solutions. The application provides customizable CRM solutions that can be integrated with additional features developed by third parties. Salesforce.com is based on the Force.com cloud development platform. This represents scalable and high-performance middleware executing all the operations of all Salesforce.com applications. Initially designed to support scalable CRM applications, the platform has evolved to support the entire life cycle of a wider range of cloud applications by implementing a flexible and scalable infrastructure.

At the core of the platform resides its metadata architecture, which provides the system with flexibility and scalability. Rather than being built on top of specific components and tables,

application core logic and business rules are saved as metadata into the Force.com store. Both application structure and application data are stored in the store.



A runtime engine executes application logic by retrieving its metadata and then performing the operations on the data. Although running in isolated containers, different applications logically share the same database structure, and the runtime engine executes all of them uniformly. A full-text search engine supports the runtime engine. This allows application users to have an effective user experience despite the large amounts of data that need to be crawled.

The search engine maintains its indexing data in a separate store and is constantly updated by background processes triggered by user interaction. Users can customize their application by leveraging the ―native‖ Force.com application framework or by using programmatic APIs in the most popular programming languages.

The application framework allows users to visually define either the data or the core structure of a Force.com application, while the programmatic APIs provide them with a more conventional way for developing applications that relies on Web services to interact with the platform. Customization of application processes and logic can also be implemented by developing scripts in APEX. This is a Java-like language that provides object-oriented and procedural capabilities for defining either scripts executed on demand or triggers. APEX also

offers the capability of expressing searches and queries to have complete access to the data managed by the Force.com platform.

2.11 Introduction to MapReduce

MapReduce is a programming platform Google introduced for processing large quantities of data. It expresses the computational logic of an application in two simple functions: *map* and *reduce*. Data transfer and management are completely handled by the distributed storage infrastructure (i.e., the Google File System), which is in charge of providing access to data, replicating files, and eventually moving them where needed. Therefore, developers no longer have to handle these issues and are provided with an interface that presents data at a higher level: as a collection of key- value pairs.

The computation of MapReduce applications is then organized into a workflow of map and reduce operations that is entirely controlled by the runtime system; developers need only specify how the map and reduce functions operate on the key-value pairs. More precisely, the MapReduce model is expressed in the form of the two functions, which are defined as follows:

Map (k1, v1) ⟶ list (k2, v2)
Reduce (k2, list (v2)) ⟶ list (v2)

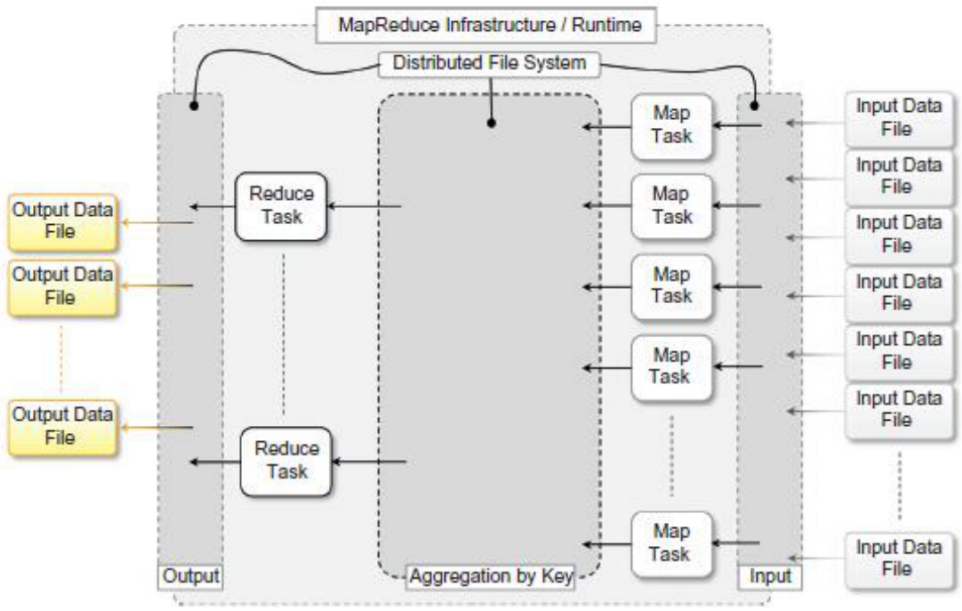
The map function reads a key-value pair and produces a list of key-value pairs of different types. The reduce function reads a pair composed of a key and a list of values and produces a list of values of the same type. The types (k1,v1,k2,kv2) used in the expression of the two functions provide hints as to how these two functions are connected and are
executed to carry out the computation of a MapReduce job: The output of map tasks is aggregated together by grouping the values according to their corresponding keys and constitutes the input of reduce tasks that, for each of the keys found, reduces the list of attached values to a single value. Therefore, the input of a MapReduce computation is expressed as a collection of key-value pairs <k1,v1> and the final output is represented by a list of values: list(v2).

The user submits a collection of files that are expressed in the form of a list of <k1,v1> pairs and specifies the map and reduce functions. These files are entered into the distributed file system that supports MapReduce and, if necessary, partitioned in order to be the input of map tasks. Map tasks generate intermediate files that store collections of < k2, list (v2) > pairs, and

these files are saved into the distributed file system. The MapReduce runtime might eventually aggregate the values corresponding to the same keys. These files constitute the input of reduce tasks, which finally produce output files in the form of list (v2).

The operation performed by reduce tasks is generally expressed as an aggregation of all the values that are mapped by a specific key. The number of map and reduce tasks to create, the way files are partitioned with respect to these tasks, and the number of map tasks connected to a single reduce task are the responsibilities of the MapReduce runtime. In addition, the way files are stored and moved is the responsibility of the distributed file system that supports MapReduce.

The computation model expressed by MapReduce is very straightforward and allows greater productivity for people who have to code the algorithms for processing huge quantities of data. This model has proven successful in the case of Google, where the majority of the information that needs to be processed is stored in textual form and is represented by Web pages or log files. Some of the examples that show the flexibility of MapReduce are



- **Distributed grep**: The grep operation, which performs the recognition of patterns within text streams, is performed across a wide set of files. MapReduce is leveraged to provide a parallel and faster execution of this operation. In this case, the input file is a plain text file,

and the map function emits a line into the output each time it recognizes the given pattern. The reduce task aggregates all the lines emitted by the map tasks into a single file.

- **Count of URL-access frequency:** MapReduce is used to distribute the execution of Web server log parsing. In this case, the map function takes as input the log of a Web server and emits into output file a key-value pair <URL,1> for each page access recorded in the log. The reduce function aggregates all these lines by the corresponding URL, thus summing the single accesses, and outputs a <URL, total-count> pair.
- **Reverse Web-link graph**: The Reverse Web-link graph keeps track of all the possible Web pages that might lead to a given link. In this case input files are simple HTML pages that are scanned by map tasks emitting <target, source> pairs for each of the links found in the Web page source. The reduce task will collate all the pairs that have the same target into a <target, list (source)> pair. The final result is given one or more files containing these mappings.

- **Term vector per host**: A term vector recaps the most important words occurring in a set of documents in the form of list <word, frequency> where the number of occurrences of a word is taken as a measure of its importance. MapReduce is used to provide a mapping between the origin of a set of document, obtained as the host component of the URL of a document, and the corresponding term vector. In this case, the map task creates a pair <host, term-vector> for each text document retrieved, and the reduce task aggregates the term vectors corresponding to documents retrieved from the same host.

- **Inverted index**: The inverted index contains information about the presence of words in documents. This information is useful to allow fast full-text searches compared to direct document scans. In this case, the map task takes as input a document, and for each document it emits a collection of <word, document-id>.The reduce function aggregates the occurrences of the same word, producing a pair <word, list (document-id)>.
- **Distributed sort:** In this case, MapReduce is used to parallelize the execution of a sort operation over a large number of records. This application mostly relies on the properties of the MapReduce runtime, which sorts and creates partitions of the intermediate files, rather than in the operations, performed in the map and reduce tasks. Indeed, these are very simple: The map task extracts the key from a record and emits a <key, record> pair for each record; the reduce task will simply copy through all the pairs. The actual sorting process is

performed by the MapReduce runtime, which will emit and partition the key-value pair by ordering them according to the key.

In general, any computation that can be expressed in the form of two major stages can be represented in the terms of MapReduce computation. These stages are:

- **Analysis:** This phase operates directly on the data input file and corresponds to the operation performed by the map task. Moreover, the computation at this stage is expected to be embarrassingly parallel, since map tasks are executed without any sequencing or ordering.
- **Aggregation:** This phase operates on the intermediate results and is characterized by operations that are aimed at aggregating, summing, and/or elaborating the data obtained at the previous stage to present the data in their final form. This is the task performed by the reduce function.

Adaptations to this model are mostly concerned with identifying the appropriate keys, creating reasonable keys when the original problem does not have such a model, and finding ways to partition the computation between map and reduce functions. Moreover, more complex algorithms can be decomposed into multiple MapReduce programs, where the output of one program constitutes the input of the following program.

The user submits the execution of MapReduce jobs by using the client libraries that are in charge of submitting the input data files, registering the map and reduce functions, and returning control to the user once the job is completed.

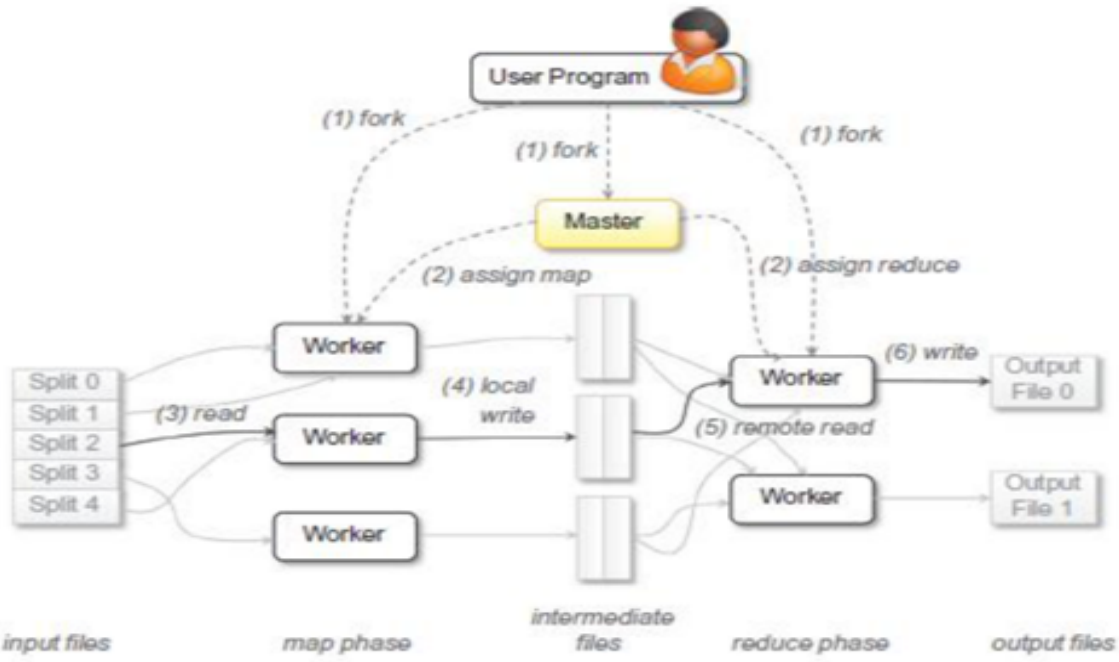
A generic distributed infrastructure (i.e., a cluster) equipped with job-scheduling capabilities and distributed storage can be used to run MapReduce applications. Two different kinds of processes are run on the distributed infrastructure: a master process and a worker process.

The master process is in charge of controlling the execution of map and reduce tasks, partitioning, and reorganizing the intermediate output produced by the map task in order to feed the reduce tasks. The worker processes are used to host the execution of map and reduce tasks and provide basic I/O facilities that are used to interface the map and reduce tasks with input and output files. In a MapReduce computation, input files are initially divided into splits (generally 16 to 64 MB) and stored in the distributed file system. The master process generates the map tasks and assigns input splits to each of them by balancing the load.

Worker processes have input and output buffers that are used to optimize the performance of map and reduce tasks. In particular, output buffers for map tasks are periodically dumped to disk to create intermediate files. Intermediate files are partitioned using a user-defined function to evenly split the output of map tasks. The locations of these pairs are then notified to the master process, which forwards this information to the reduce tasks, which are able to collect the required input via a remote procedure call in order to read from the map tasks‘ local storage.

The key range is then sorted and all the same keys are grouped together. Finally, the reduce task is executed to produce the final output, which is stored in the global file system. This process is completely automatic; users may control it through configuration parameters that allow specifying (besides the map and reduce functions) the number of map tasks, the number of partitions into which to separate the final output, and the partition function for the intermediate key range.

Besides orchestrating the execution of map and reduce tasks as previously described, the MapReduce runtime ensures a reliable execution of applications by providing a fault-tolerant infrastructure. Failures of both master and worker processes are handled, as are machine failures that make intermediate outputs inaccessible. Worker failures are handled by rescheduling map tasks somewhere else. This is also the technique that is used to address machine failures since the valid intermediate output of map tasks has become inaccessible. Master process failure is instead addressed using check pointing, which allows restarting the MapReduce job with a minimum loss of data and computation.



Variations and extensions of MapReduce

MapReduce constitutes a simplified model for processing large quantities of data and imposes constraints on the way distributed algorithms should be organized to run over a MapReduce infrastructure. Although the model can be applied to several different problem scenarios, it still exhibits limitations, mostly due to the fact that the abstractions provided to process data are very simple, and complex problems might require considerable effort to be represented in terms of map and reduce functions only. Therefore, a series of extensions to and variations of the original MapReduce model have been proposed. They aim at extending the MapReduce application space and providing developers with an easier interface for designing distributed algorithms.

- Hadoop- Apache Hadoop is a collection of software projects for reliable and scalable distributed computing. Taken together, the entire collection is an open-source implementation of the MapReduce framework supported by a GFS-like distributed file system. The initiative consists of mostly two projects: Hadoop Distributed File System (HDFS) and Hadoop MapReduce. The former is an implementation of the Google File System; the latter provides the same features and abstractions as Google MapReduce. Initially developed and supported by Yahoo!, Hadoop now constitutes the most mature and large data cloud application and has a very robust community of developers and users supporting it. Yahoo! now runs the world‘s largest Hadoop cluster, composed of 40,000

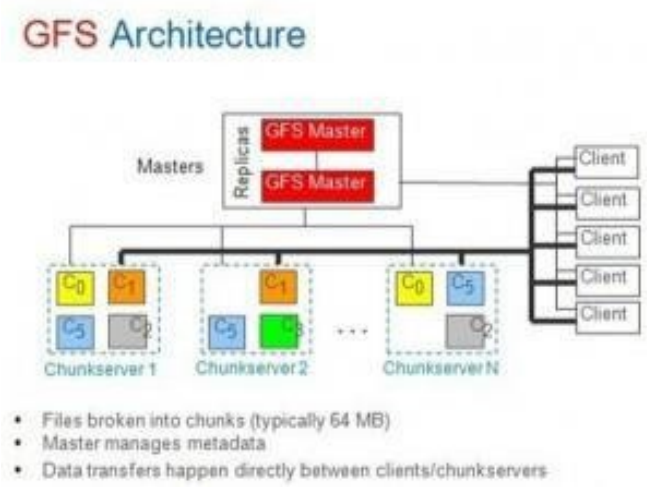
machines and more than 300,000 cores, made available to academic institutions all over the world. Besides the core projects of Hadoop, a collection of other projects related to it provides services for distributed computing.

- Pig- Pig is a platform that allows the analysis of large datasets. Developed as an Apache project, Pig consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The Pig infrastructure‘s layer consists of a compiler for a high-level language that produces a sequence of MapReduce jobs that can be run on top of distributed infrastructures such as Hadoop. Developers can express their data analysis programs in a textual language called Pig Latin, which exposes a SQL-like interface and is characterized by major expressiveness, reduced programming effort, and a familiar interface with respect to MapReduce.
- Hive- Hive is another Apache initiative that provides a data warehouse infrastructure on top of Hadoop MapReduce. It provides tools for easy data summarization, ad hoc queries, and analysis of large datasets stored in Hadoop MapReduce files. Whereas the framework provides the same capabilities as a classical data warehouse, it does not exhibit the same performance, especially in terms of query latency, and for this reason does not constitute a valid solution for online transaction processing. Hive‘s major advantages reside in the ability to scale out, since it is based on the Hadoop framework, and in the ability to provide a data warehouse infrastructure in environments where there is already a Hadoop system running.
- Map-Reduce-Merge- Map-Reduce-Merge is an extension of the MapReduce model, introducing a third phase to the standard MapReduce pipeline—the Merge phase—that allows efficiently merging data already partitioned and sorted (or hashed) by map and reduce modules. The Map- Reduce-Merge framework simplifies the management of heterogeneous related datasets and pro- vides an abstraction able to express the common relational algebra operators as well as several join algorithms.
- Twister- Twister is an extension of the MapReduce model that allows the creation of iterative executions of MapReduce jobs. Besides the iterative MapReduce computation, Twister provides additional features such as the ability for map and reduce tasks to refer to static and in-memory data; the introduction of an additional phase called combine, run at the end of the MapReduce job, that aggregates the output together; and other tools for management of data.

2.12 GFS - Google File System

GFS is the storage infrastructure that supports the execution of distributed applications in Google‘s computing cloud. The system has been designed to be a fault-tolerant, highly available, distributed file system built on commodity hardware and standard Linux operating systems. Rather than a generic implementation of a distributed file system, GFS specifically addresses Google‘s needs in terms of distributed storage for applications, and it has been designed with the following assumptions:

- The system is built on top of commodity hardware that often fails.
- The system stores a modest number of large files; multi-GB files are common and should be treated efficiently, and small files must be supported, but there is no need to optimize for that.
- The workloads primarily consist of two kinds of reads: large streaming reads and small random reads.
- The workloads also have many large, sequential writes that append data to files.
- High-sustained bandwidth is more important than low latency.



The architecture of the file system is organized into a single master, which contains the metadata of the entire file system, and a collection of chunk servers, which provide storage space. From a logical point of view the system is composed of a collection of software daemons, which implement either the master server or the chunk server. A file is a collection of chunks for which the size can be configured at file system level. Chunks are

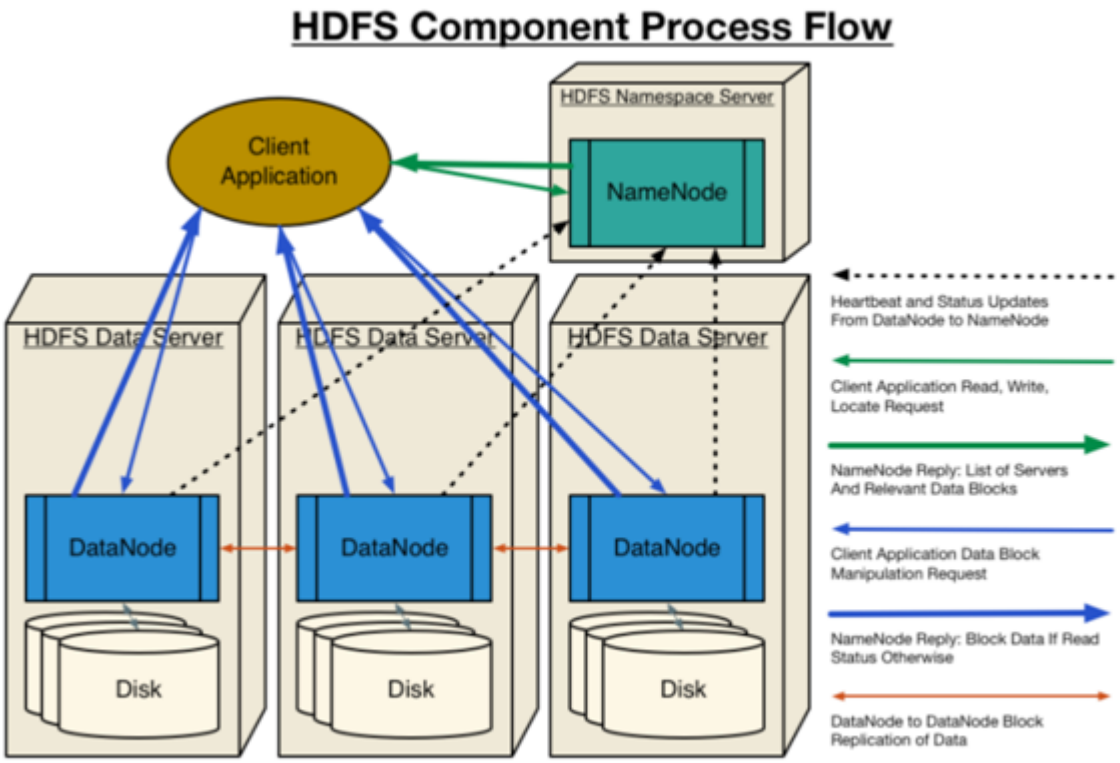
replicated on multiple nodes in order to tolerate failures. Clients look up the master server and identify the specific chunk of a file they want to access. Once the chunk is identified, the interaction happens between the client and the chunk server. Applications interact through the file system with a specific interface supporting the usual operations for file creation, deletion, read, and write. The interface also supports snapshots and records append operations that are frequently performed by applications. GFS has been conceived by considering that failures in a large distributed infrastructure are common rather than a rarity; therefore, specific attention has been given to implementing a highly available, lightweight, and fault-tolerant infrastructure. The potential single point of failure of the single-master architecture has been addressed by giving the possibility of replicating the master node on any other node belonging to the infrastructure. Moreover, a stateless daemon and extensive logging capabilities facilitate the system's recovery from failures.

2.13 HDFS (Hadoop Distributed File System)

HDFS is a distributed file system that provides high-performance access to data across Hadoop clusters. Like other Hadoop-related technologies, HDFS has become a key tool for managing pools of big data and supporting big data analytics applications.

Because HDFS typically is deployed on low-cost commodity hardware, server failures are common. The file system is designed to be highly fault-tolerant, however, by facilitating the rapid transfer of data between compute nodes and enabling Hadoop systems to continue running if a node fails. That decreases the risk of catastrophic failure, even in the event that numerous nodes fail.

When HDFS takes in data, it breaks the information down into separate pieces and distributes them to different nodes in a cluster, allowing for parallel processing. The file system also copies each piece of data multiple times and distributes the copies to individual nodes, placing at least one copy on a different server rack than the others. As a result, the data on nodes that crash can be found elsewhere within a cluster, which allows processing to continue while the failure is resolved.



HDFS is built to support applications with large data sets, including individual files that reach into the terabytes. It uses master/slave architecture, with each cluster consisting of a single Name Node that manages file system operations and supporting Data Nodes that manage data storage on individual compute nodes. The HDFS cluster consists of a single Name Node and a master server manages the file system namespace and regulates access to files. The primary objective of HDFS is to store data reliably even in the presence of failures including Name Node failures, Data Node failures and network partitions. The Name Node is a single point of failure for the HDFS cluster and a Data Node stores data in the Hadoop file management system

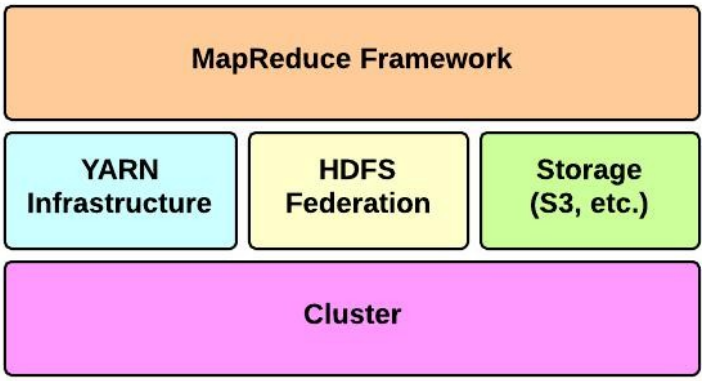
2.14 Hadoop Framework

Apache Hadoop is an open-source software framework for storage and large-scale processing of data-sets on clusters of commodity hardware. The basic functionalities of Hadoop Distributed File System (HDFS)

- Large files are split into blocks of equal size
- These blocks are distributed across the cluster for storage

- Because node failure is a reality to be considered in a larger cluster, each block is stored multiple times (typically three times) on different computers

There are mainly five building blocks inside this runtime environment (from bottom to top):

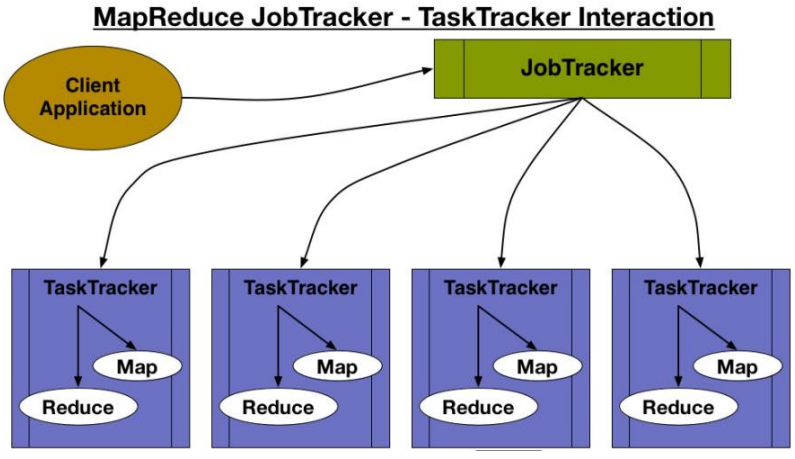


- The **cluster** is the set of host machines (**nodes**). Nodes may be partitioned in **racks**. This is the hardware part of the infrastructure.
- The **YARN Infrastructure** (Yet Another Resource Negotiator) is the framework responsible for providing the computational resources (e.g., CPUs, memory, etc.) needed for application executions. Two important elements are:
  - The **Resource Manager** (one per cluster) is the master. It knows where the slaves are located (Rack Awareness) and how many resources they have. It runs several services; the most important is the **Resource Scheduler** which decides how to assign the resources.
  - The **Node Manager** (many per cluster) is the slave of the infrastructure. When it starts, it announces himself to the Resource Manager. Periodically, it sends an heartbeat to the Resource Manager. Each Node Manager offers some resources to the cluster. Its resource capacity is the amount of memory and the number of vcores. At run-time, the Resource Scheduler will decide how to use this capacity: a **Container** is a fraction of the NM capacity and it is used by the client for running a program.

- The **HDFS Federation** is the framework responsible for providing permanent, reliable and distributed storage. This is typically used for storing inputs and output (but not intermediate ones).
- Other alternative storage solutions- Amazon uses the **Simple Storage Service (S3).**
- **MapReduce Framework** is the software layer implementing the **MapReduce paradigm**.

The other critical components are your **MapReduce** computational layers. This is a complex set of rules that Hadoop workloads depend on where massive volumes of data are mapped and then reduced for efficient lookups, reads and writes – across all the nodes. There are other processes like combiner, sorting and shuffling that effects the reduced output. **Reducer** communicates with all nodes and facilitates the output. Since, the reducer sort of plays an aggregate role; the number of reducers to the number of nodes (mappers/etc.) effects the performance of the applications. It's **TaskTrackers** responsibility to track at a local node, while the **JobTracker** oversees all the nodes in the cluster.

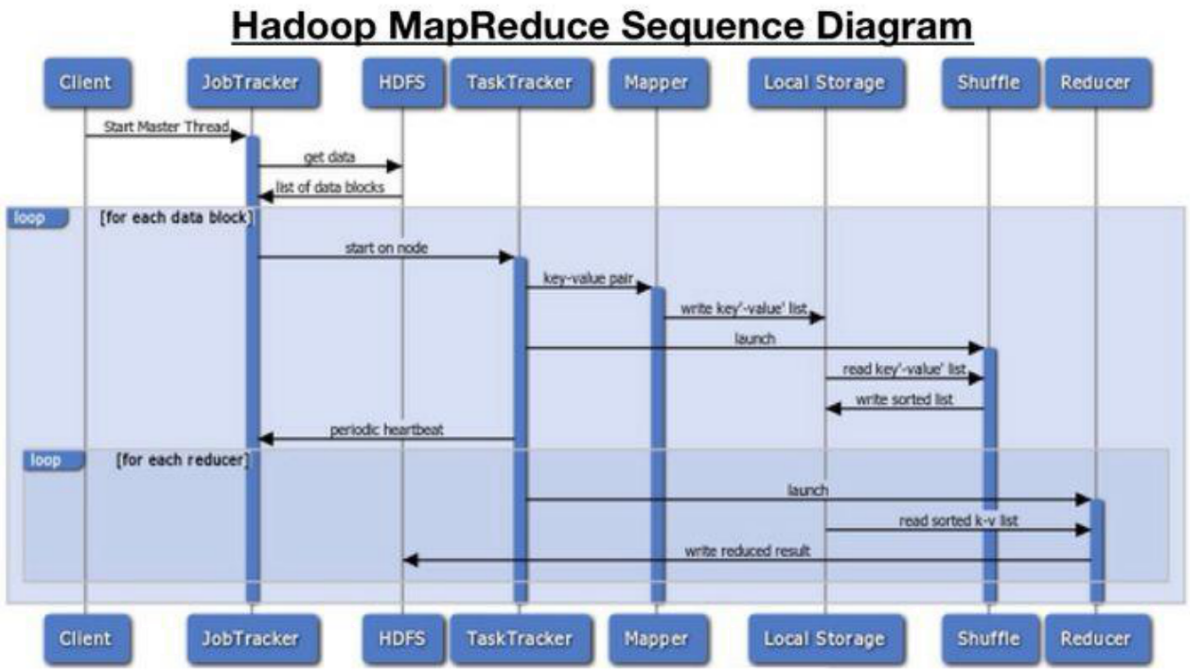
The YARN infrastructure and the HDFS federation are completely decoupled and independent: the first one provides resources for running an application while the second one provides storage. The MapReduce framework is only one of many possible framework which runs on top of YARN (although currently is the only one implemented).



YARN: Application Startup

In YARN, there are at least three actors:

- the **Job Submitter** (the client)
- the **Resource Manager** (the master)
- the **Node Manager** (the slave)



The application startup process is the following:

1. a client submits an application to the Resource Manager
2. the Resource Manager allocates a container
3. the Resource Manager contacts the related Node Manager
4. the Node Manager launches the container
5. the Container executes the **Application Master**

The Application Master is responsible for the execution of a single application. It asks for containers to the Resource Scheduler (Resource Manager) and executes specific programs (e.g., the main of a Java class) on the obtained containers. The Application Master knows the application logic and thus it is framework-specific. The MapReduce framework provides its own implementation of an Application Master. The Resource Manager is a single point of failure in YARN. Using Application Masters, YARN is spreading over the cluster the metadata related to running applications. This reduces the load of the Resource Manager and makes it fast recoverable.