Cloud Computing

Unit 5

Cloud Technologies and Advancements

Apache Hadoop is an open-source software framework for storage and large-scale processing of data-sets on clusters of commodity hardware. Hadoop is an Apache top-level project being built and used by a global community of contributors and users. It is licensed under the Apache License 2.0.

The Apache Hadoop framework is composed of the following modules:

- Hadoop Common– contains libraries and utilities needed by other Hadoop modules
- HadoopDistributed File System (HDFS) – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster.
- Hadoop YARN– a resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications.
- Hadoop MapReduce– a programming model for large scale data processing.

All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are common and thus should be automatically handled in software by the framework. Apache Hadoop's MapReduce and HDFS components originally derived respectively from Google's MapReduce and Google File System (GFS) papers. Beyond HDFS, YARN and MapReduce, the entire Apache Hadoop "platform" is now commonly considered to consist of a number of related projects as well – Apache Pig, Apache Hive, Apache HBase, Apache Spark, and others.

For the end-users, though MapReduce Java code is common, any programming language can be used with "Hadoop Streaming" to implement the "map" and "reduce" parts of the user's program. Apache Pig, Apache Hive, Apache Spark among other related projects expose higher level user interfaces like Pig latin and a SQL variant respectively. The Hadoop framework itself is mostly written in the Java programming language, with some native code in C and command line utilities written as shell-scripts.

Apache Hadoop is a registered trademark of the Apache Software Foundation.

Hadoop on the Cloud

Most of the major Apache Hadoop distributions now offer their version of the Hadoop platform in the cloud uniting two of the biggest topics in the tech world today: big data and cloud computing. cloud offers several advantages for businesses looking to use Hadoop, so all businesses – including small and medium-sized ones – can truly start to take advantage of big data.

1. Flexibility and Getting Started Quickly and Easily

Bringing Hadoop to the cloud offers businesses the flexibility to use Hadoop according their needs. One can scale up or scale down as often as one needs. Most cloud providers allow users to sign up and get started within a matter of minutes, making it ideal for a small business with limited resources and for large businesses that need to scale beyond their current solution without making a long-term investment.

2. Cost-Effective

Despite the fact that the open source version of Hadoop is free, the claim that Hadoop is cheap is a myth. While Hadoop is a much more cost-effective solution for storing large volumes of data, it can still require major investments in hardware, development, maintenance and expertise. This level of investment may be worth it for companies that plan on using Hadoop regularly, but for those who only need to run occasional analytics or simply don't have the budget for the upfront investment, the cost of Hadoop can be a real issue. The cloud offers a cost-effective solution for Hadoop. Most cloud providers charge on a pay-per-use basis so businesses can pay for the storage or analytics they need without making that upfront investment or paying for maintaining a system when it is not being used.

3. Real-Time Analytics

Big data analytics has a lot to offer. From a more complete view of the consumer to more efficient manufacturing and improved product innovation, valuable insights lie within the mass stores of multi-structured data being created. Unfortunately, the generic open source version of Hadoop cannot currently handle real time analysis, meaning businesses that want to analyze and adjust campaigns or products quickly can't currently do so. However, MapR's enterprise-ready distribution of Hadoop, which re-architected the data platform and applied other architectural innovations, can support real-time analytics. With this particular Hadoop distribution available on the cloud, businesses can have instant access to their data for real-time processing and analysis. These are just three advantages that Hadoop in the cloud has to offer. Depending on a business's needs the flexibility of the cloud allows it to adapt to various situations and solve problems that may not even be listed here. Before dismissing Hadoop as beyond your reach, look into what a cloud solution has to offer.

MapReduce

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

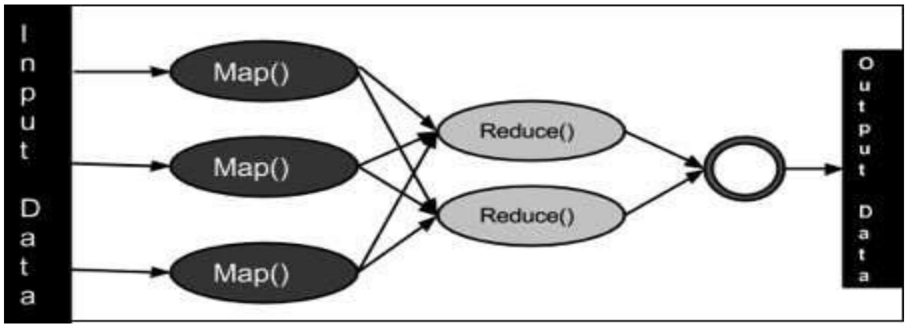
The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into *mappers* and *reducers* is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

The Algorithm

- Generally MapReduce paradigm is based on sending the computer to where the data resides!
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
  - **Map stage** − The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.

    - **Reduce stage** − This stage is the combination of the **Shuffle** stage and the **Reduce** stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.



Virtual Box

A **Virtual Box** or VB is a hypervisor for X86 computers from Oracle corporation. It was first developed by Innotek GmbH and released in 2007 as an open source software package. The company was later acquired by Sun Micro Systems in 2008. After that, Oracle has continued the development of Virtual Box since 2010 and the product name is titled as Oracle **VM Virtual Box**. Virtual Box comes in different flavours depending upon the operating systems for which it is being configured. **Virtual Box Ubuntu** is more preferred, even though **Virtual Box for windows** is equally popular. With the advent of android phones **VirtualBox for android** is becoming the new face of VM in smartphones.

Use of Virtual Box

In general, a Virtual Box is a software virtualization package that can be installed on any operating system as an application software. It allows additional operating systems to be installed on it, as a Guest OS. It can than **create and manage free guest virtual machines**, each with a guest operating system and its own virtual environment. **Virtual Box** is being supported by many operating systems like Windows XP, Windows Vista, **Windows** 7, Linux, Mac OS X, Solaris, and Open Solaries. Supported guest operating systems are versions and derivations of Windows, Linux, OS/2, BSD, Haiku, etc.

Which is better VMWare of Virtual Box?

Virtual Box gets a lot of support, primarily because it is free and open-source. It also allows unlimited snapshots – a feature only available in VMWare Pro. VMWare, on the other hand, is great for drag and drop functionality between host and the VM, but many features come only in paid version.

Google App Engine

Google App Engine is a web application hosting service. By “web application,” we mean an application or service accessed over the Web, usually with a web browser: storefronts with shopping carts, social networking sites, multiplayer games, mobile applications, survey applications, project management, collaboration, publishing, and all the other things we’re discovering are good uses for the Web. App Engine can serve traditional website content too, such as documents and images, but the environment is especially designed for real-time dynamic applications. Of course, a web browser is merely one kind of client: web application infrastructure is well suited to mobile applications, as well.

In particular, Google App Engine is designed to host applications with many simultaneous users. When an application can serve many simultaneous users without degrading performance, we say it *scales*. Applications written for App Engine scale automatically. As more people use the application, App Engine allocates more resources for the application and manages the use of those resources. The application itself does not need to know anything about the resources it is using.

Unlike traditional web hosting or self-managed servers, with Google App Engine, you only pay for the resources you use. Billed resources include CPU usage, storage per month, incoming and outgoing bandwidth, and several resources specific to App Engine services. To help you get started, every developer gets a certain amount of resources for free, enough for small applications with low traffic.

App Engine is part of Google Cloud Platform, a suite of services for running scalable applications, performing large amounts of computational work, and storing, using, and analyzing large amounts of data. The features of the platform work together to host applications efficiently and effectively, at minimal cost. App Engine's specific role on the platform is to host web applications and scale them automatically. App Engine apps use the other services of the platform as needed, especially for data storage.

An App Engine web application can be described as having three major parts: application instances, scalable data storage, and scalable services. In this chapter, we look at each of these parts at a high level. We also discuss features of App Engine for deploying and managing web applications, and for building websites integrated with other parts of Google Cloud Platform.

The Runtime Environment

An App Engine application responds to web requests. A web request begins when a client, typically a user's web browser, contacts the application with an HTTP request, such as to fetch a web page at a URL. When App Engine receives the request, it identifies the application from the domain name of the address, either a custom domain name you have registered and configured for use with the app, or an *.appspot.com* subdomain provided for free with every app. App Engine selects a server from many possible servers to handle the request, making its selection based on which server is most likely to provide a fast response. It then calls the application with the content of the HTTP request, receives the response data from the application, and returns the response to the client.

From the application's perspective, the runtime environment springs into existence when the request handler begins, and disappears when it ends. App Engine provides several methods for storing data that persists between requests, but these mechanisms live outside of the runtime environment. By not retaining state in the runtime environment between requests—or at least, by not expecting that state will be retained between requests—App Engine can distribute traffic among as many servers as it needs to give every request the same treatment, regardless of how much traffic it is handling at one time.

In the complete picture, App Engine allows runtime environments to outlive request handlers, and will reuse environments as much as possible to avoid unnecessary initialization. Each instance of your application has local memory for caching imported code and initialized data structures. App Engine creates and destroys instances as needed to accommodate your app's traffic. If you enable the multithreading feature, a single instance can handle multiple requests concurrently, further utilizing its resources.

Application code cannot access the server on which it is running in the traditional sense. An application can read its own files from the filesystem, but it cannot write to files, and it cannot read files that belong to other applications. An application can see environment variables set by App Engine, but manipulations of these variables do not necessarily persist between requests. An application cannot access the networking facilities of the server hardware, although it can perform networking operations by using services.

In short, each request lives in its own "sandbox." This allows App Engine to handle a request with the server that would, in its estimation, provide the fastest response. For web requests to the app, there is no way to guarantee that the same app instance will handle two requests, even if the requests come from the same client and arrive relatively quickly.

Sandboxing also allows App Engine to run multiple applications on the same server without the behavior of one application affecting another. In addition to limiting access to the operating system, the runtime environment also limits the amount of clock time and memory a single request can take. App Engine keeps these limits flexible, and applies stricter limits to applications that use up more resources to protect shared resources from "runaway" applications.

A request handler has up to 60 seconds to return a response to the client. While that may seem like a comfortably large amount for a web app, App Engine is optimized for applications that respond in less than a second. Also, if an application uses many CPU cycles, App Engine may slow it down so the app isn't hogging the processor on a machine serving multiple apps. A CPU-intensive request handler may take more clock time to complete than it would if it had exclusive use of the processor, and clock time may vary as App Engine detects patterns in CPU usage and allocates accordingly.

Google App Engine provides four possible runtime environments for applications, one for each of four programming languages: Java, Python, PHP, and Go. The environment you choose depends on the language and related technologies you want to use for developing the application.

The Python environment runs apps written in the Python 2.7 programming language, using a custom version of CPython, the official Python interpreter. App Engine invokes a Python app using WSGI, a widely supported application interface standard. An application can use most of Python's large and excellent standard library, as well as rich APIs and libraries for accessing services and modeling data. Many open source Python web application frameworks work with App Engine, such as Django, web2py, Pyramid, and Flask. App Engine even includes a lightweight framework of its own, called webapp.

Similarly, the Java, PHP, and Go runtime environments offer standard execution environments for those languages, with support for standard libraries and third-party frameworks.

All four runtime environments use the same application server model: a request is routed to an app server, an application instance is initialized (if necessary), application code is invoked to handle the request and produce a response, and the response is returned to the client. Each environment runs application code within sandbox restrictions, such that any attempt to use a feature of the language or a library that would require access outside of the sandbox returns an error.

(Another Source)

Google App Engine

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

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

Infrastructure

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

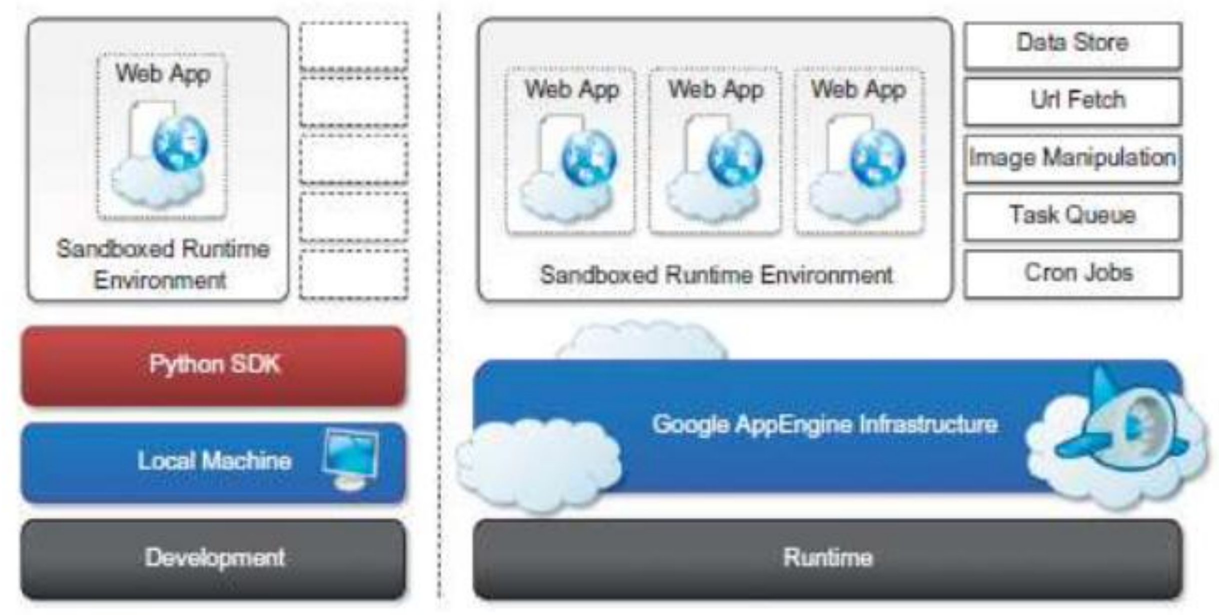


FIGURE 9.2

Google AppEngine platform architecture.

Microsoft Azure

AppEngine, a framework for developing scalable Web applications, leverages Google's infrastruc- ture. The core components of the service are a scalable and sandboxed runtime environment for executing applications and a collection of services that implement most of the common features required for Web development and that help developers build applications that are easy to scale. One of the characteristic elements of AppEngine is the use of simple interfaces that allow applica- tions to perform specific operations that are optimized and designed to scale. Building on top of these blocks, developers can build applications and let AppEngine scale them out when needed. The Windows Azure platform is made up of a foundation layer and a set of developer services that can be used to build scalable applications. These services cover compute, storage, networking, and identity management, which are tied together by middleware called AppFabric. This scalable computing environment is hosted within Microsoft datacenters and accessible through the Windows Azure Management Portal. Alternatively, developers can recreate a Windows Azure environment (with limited capabilities) on their own machines for development and testing purposes.

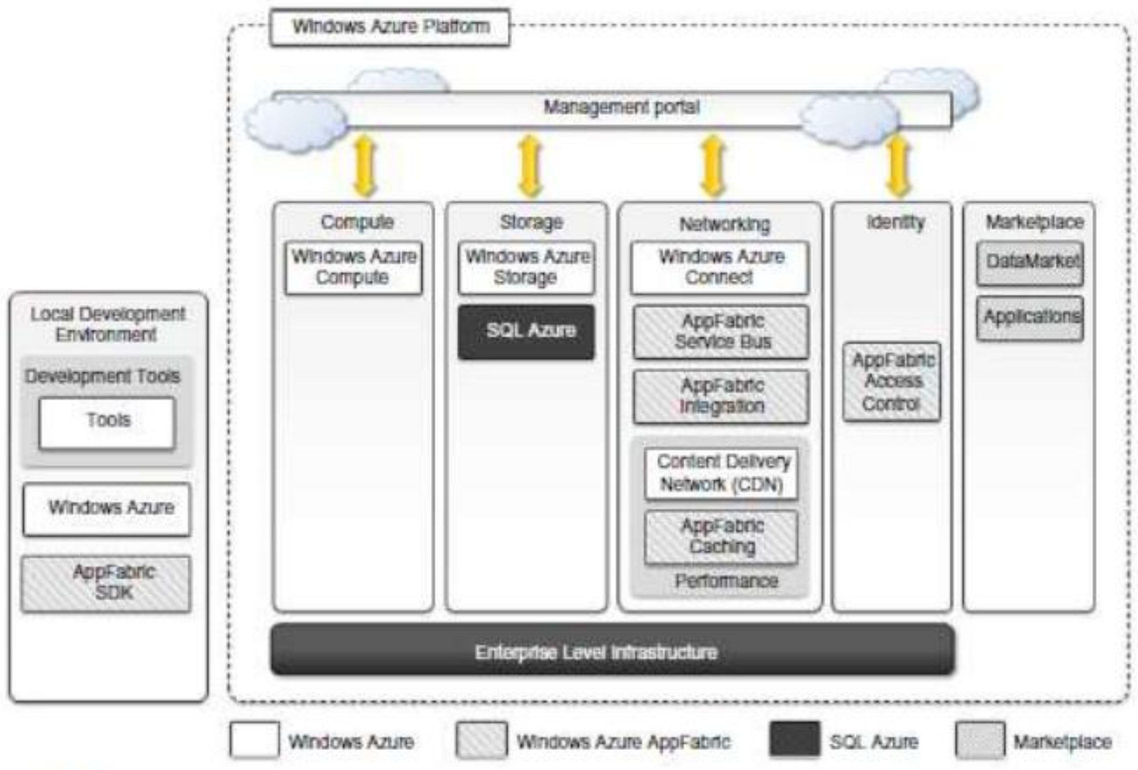


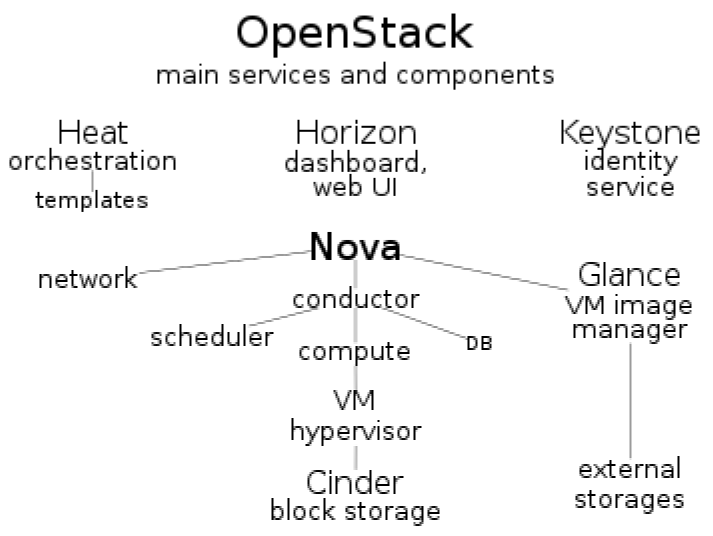
FIGURE 9.3

Microsoft Windows Azure Platform Architecture.

Open Stack

OpenStack is a free open standard cloud computing platform, mostly deployed as infrastructure-as-a-service (IaaS) in both public and private clouds where virtual servers and other resources are made available to users.The software platform consists of interrelated components that control diverse, multi-vendor hardware pools of processing, storage, and networking resources throughout a data center. Users either manage it through a web-based dashboard, through command-line tools, or through RESTful web services.

OpenStack has a modular architecture with various code names for its components.



Compute (Nova)

Nova is the OpenStack project that provides a way to provision compute instances (aka virtual servers). Nova supports creating virtual machines, baremetal servers (through the use of ironic), and has limited support for system containers. Nova runs as a set of daemons on top of existing Linux servers to provide that service. Nova is written in Python. It uses many external Python libraries such as Eventlet (concurrent networking library), Kombu (AMQP messaging framework), and SQLAlchemy (SQL toolkit and Object Relational Mapper). Nova is designed to be horizontally scalable. Rather than switching to larger servers, you procure more servers and simply install identically configured services. Due to its widespread integration into enterprise-level infrastructures, monitoring OpenStack performance in general, and Nova performance in particular, scaling has become an increasingly important issue. Monitoring end-to-end performance requires tracking metrics from Nova, Keystone, Neutron, Cinder, Swift and other services, in addition to monitoring RabbitMQ which is used by OpenStack services for message passing. All these services generate their own log files, which, especially in enterprise-level infrastructures, also should be monitored.

Networking (Neutron)

Neutron is an OpenStack project to provide “network connectivity as a service” between interface devices (e.g., vNICs) managed by other OpenStack services (e.g., nova). It implements the OpenStack Networking API. It manages all networking facets for the Virtual Networking Infrastructure (VNI) and the access layer aspects of the Physical Networking Infrastructure (PNI) in the OpenStack environment. OpenStack Networking enables projects to create advanced virtual network topologies which may include services such as a firewall, and a virtual private network (VPN). Neutron allows dedicated static IP addresses or DHCP. It also allows Floating IP addresses to let traffic be dynamically rerouted. Users can use software-defined networking (SDN) technologies like OpenFlow to support multi-tenancy and scale. OpenStack networking can deploy and manage additional network services—such as intrusion detection systems (IDS), load balancing, firewalls, and virtual private networks (VPN).

Block storage (Cinder)

Cinder is the OpenStack Block Storage service for providing volumes to Nova virtual machines, Ironic bare metal hosts, containers and more. Some of the goals of Cinder are to be/have:

Component based architecture: Quickly add new behaviors

Highly available: Scale to very serious workloads

Fault-Tolerant: Isolated processes avoid cascading failures

Recoverable: Failures should be easy to diagnose, debug, and rectify

Open Standards: Be a reference implementation for a community-driven api

Cinder volumes provide persistent storage to guest virtual machines - known as instances, that are managed by OpenStack Compute software. Cinder can also be used independent of other OpenStack services as stand-alone software-defined storage. The block storage system manages the creation,replication, snapshot management, attaching and detaching of the block devices to servers.

Identity (Keystone)

Keystone is an OpenStack service that provides API client authentication, service discovery, and distributed multi-tenant authorization by implementing OpenStack's Identity API. It is the common authentication system across the cloud operating system. Keystone can integrate with directory services like LDAP. It supports standard username and password credentials, token-based systems and AWS-style (i.e. Amazon Web Services) logins. The OpenStack keystone service catalog allows API clients to dynamically discover and navigate to cloud services.

Image (Glance)

The Image service (glance) project provides a service where users can upload and discover data assets that are meant to be used with other services. This currently includes images and metadata definitions.

Images

Glance image services include discovering, registering, and retrieving virtual machine (VM) images. Glance has a RESTful API that allows querying of VM image metadata as well as retrieval of the actual image. VM images made available through Glance can be stored in a variety of locations from simple filesystems to object-storage systems like the OpenStack Swift project.

Metadata Definitions

Glance hosts a metadefs catalog. This provides the OpenStack community with a way to programmatically determine various metadata key names and valid values that can be applied to OpenStack resources.

Object storage (Swift)

Swift is a distributed, eventually consistent object/blob store. The OpenStack Object Store project, known as Swift, offers cloud storage software so that you can store and retrieve lots of data with a simple API. It's built for scale and optimized for durability, availability, and concurrency across the entire data set. Swift is ideal for storing unstructured data that can grow without bound. In August 2009, Rackspace started the development of the precursor to

OpenStack Object Storage, as a complete replacement for the Cloud Files product. The initial development team consisted of nine developers. Swift-Stack, an object storage software company, is currently the leading developer for Swift with significant contributions from Intel, Red Hat, NTT, HP, IBM, and more.

Dashboard (Horizon)

Horizon is the canonical implementation of OpenStack's Dashboard, which provides a web based user interface to OpenStack services including Nova, Swift, Keystone, etc. Horizon ships with three central dashboards, a “User Dashboard”, a “System Dashboard”, and a “Settings” dashboard. Between these three they cover the core OpenStack applications and deliver on Core Support. The Horizon application also ships with a set of API abstractions for the core OpenStack projects in order to provide a consistent, stable set of reusable methods for developers. Using these abstractions, developers working on Horizon don't need to be intimately familiar with the APIs of each OpenStack project.

Orchestration (Heat)

Heat is a service to orchestrate multiple composite cloud applications using templates, through both an OpenStack-native REST API and a CloudFormation-compatible Query API.

Workflow (Mistral)

Mistral is a service that manages workflows. User typically writes a workflow using workflow language based on YAML and uploads the workflow definition to Mistral via its REST API. Then user can start this workflow manually via the same API or configure a trigger to start the workflow on some event.

Telemetry (Ceilometer)

OpenStack Telemetry (Ceilometer) provides a Single Point Of Contact for billing systems, providing all the counters they need to establish customer billing, across all current and future OpenStack components. The delivery of counters is traceable and auditable, the counters must be easily extensible to support new projects, and agents doing data collections should be independent of the overall system.

Database (Trove)

Trove is a database-as-a-service provisioning relational and a non-relational database engine.

Elastic map reduce (Sahara)

Sahara is a component to easily and rapidly provision Hadoop clusters. Users will specify several parameters like the Hadoop version number, the cluster topology type, node flavor details (defining disk space, CPU and RAM settings), and others. After a user provides all of the parameters, Sahara deploys the cluster in a few minutes. Sahara also provides means to scale a preexisting Hadoop cluster by adding and removing worker nodes on demand.

Bare metal (Ironic)

Ironic is an OpenStack project that provisions bare metal machines instead of virtual machines. It was initially forked from the Nova Baremetal driver and has evolved into a separate project. It is best thought of as a bare-metal hypervisor API and a set of plugins that interact with the bare-metal hypervisors. By default, it will use PXE and IPMI in concert to provision and turn on and off machines, but Ironic supports and can be extended with vendor-specific plugins to implement additional functionality.

Messaging (Zaqar)

Zaqar is a multi-tenant cloud messaging service for Web developers. The service features a fully RESTful API, which developers can use to send messages between various components of their SaaS and mobile applications by using a variety of communication patterns. Underlying this API is an efficient messaging engine designed with scalability and security in mind. Other OpenStack components can integrate with Zaqar to surface events to end users and to communicate with guest agents that run in the "over-cloud" layer.

Shared file system (Manila)

OpenStack Shared File System (Manila) provides an open API to manage shares in a vendor agnostic framework. Standard primitives include ability to create, delete, and give/deny access to a share and can be used standalone or in a variety of different network environments. Commercial storage appliances from EMC, NetApp, HP, IBM, Oracle, Quobyte, INFINIDAT and Hitachi Data Systems are supported as well as filesystem technologies such as Red Hat GlusterFS or Ceph.

DNS (Designate)

Designate is a multi-tenant REST API for managing DNS. This component provides DNS as a Service and is compatible with many backend technologies, including PowerDNS and BIND. It doesn't provide a DNS service as such as its purpose is to interface with existing DNS servers to manage DNS zones on a per tenant basis.

Search (Searchlight)

Searchlight provides advanced and consistent search capabilities across various OpenStack cloud services. It accomplishes this by offloading user search queries from other OpenStack API servers by indexing their data into ElasticSearch. Searchlight is being integrated into Horizon and also provides a Command-line interface.

Key manager (Barbican)

Barbican is a REST API designed for the secure storage, provisioning and management of secrets. It is aimed at being useful for all environments, including large ephemeral Clouds.

Container orchestration (Magnum)

Magnum is an OpenStack API service developed by the OpenStack Containers Team making container orchestration engines such as Docker Swarm, Kubernetes, and Apache Mesos available as first class resources in OpenStack. Magnum uses Heat to orchestrate an OS image which contains Docker and Kubernetes and runs that image in either virtual machines or bare metal in a cluster configuration.

Root Cause Analysis (Vitrage)

Vitrage is the OpenStack RCA (Root Cause Analysis) service for organizing, analyzing and expanding OpenStack alarms & events, yielding insights regarding the root cause of problems and deducing their existence before they are directly detected.

Rule-based alarm actions (Aodh)

This alarming service enables the ability to trigger actions based on defined rules against metric or event data collected by Ceilometer or Gnocchi.

Compatibility with other cloud APIs

OpenStack does not strive for compatibility with other clouds' APIs. However, there is some amount of compatibility driven by various members of the OpenStack community for whom such things are important.

The EC2 API project aims to provide compatibility with Amazon EC2

The GCE API project aims to provide compatibility with Google Compute Engine

Governance

OpenStack is governed by a non-profit foundation and its board of directors, a technical committee, and a user committee. The board of directors is made up of eight members from each of the eight platinum sponsors, eight members from the 24 defined maximum allowed Gold sponsors, and eight members elected by the Foundation individual members.

Appliances

An OpenStack Appliance is the name given to software that can support the OpenStack cloud computing platform on either physical devices such as servers or virtual machines or a combination of the two. Typically a software appliance is a set of software capabilities that can function without an operating system. Thus, they must contain enough of the essential underlying operating system components to work. Therefore, a strict definition might be: an application that is designed to offer OpenStack capability without the necessity of an underlying operating system. However, applying this strict definition may not be helpful, as there is not really a clear distinction between an appliance and a distribution. It could be argued that the term appliance is something of a misnomer because OpenStack itself is referred to as a cloud operating system so using the term OpenStack appliance could be a misnomer if one is being pedantic. If we look at the range of Appliances and Distributions one could make the distinction that distributions are those toolsets which attempt to provide a wide coverage of the OpenStack project scope, whereas an Appliance will have a more narrow focus, concentrating on fewer projects. Vendors have been heavily involved in OpenStack since its inception, and have since developed and are marketing a wide range of appliances, applications and distributions.

Federation in cloud

Cloud federation is the practice of interconnecting the cloud computing environments of two or more service providers for the purpose of load balancing traffic and accommodating spikes in demand.

Cloud federation requires one provider to wholesale or rent computing resources to another cloud provider. Those resources become a temporary or permanent extension of the buyer's cloud computing environment, depending on the specific federation agreement between providers.

Cloud federation is the practice of interconnecting the cloud computing environments of two or more service providers for the purpose of load balancing traffic and accommodating spikes in demand.

Cloud federation requires one provider to wholesale or rent computing resources to another cloud provider. Those resources become a temporary or permanent extension of the buyer's cloud computing environment, depending on the specific federation agreement between providers.

Cloud federation offers two substantial benefits to cloud providers. First, it allows providers to earn revenue from computing resources that would otherwise be idle or underutilized. Second, cloud federation enables cloud providers to expand their geographic footprints and accommodate sudden spikes in demand without having to build new points-of-presence (POPs).

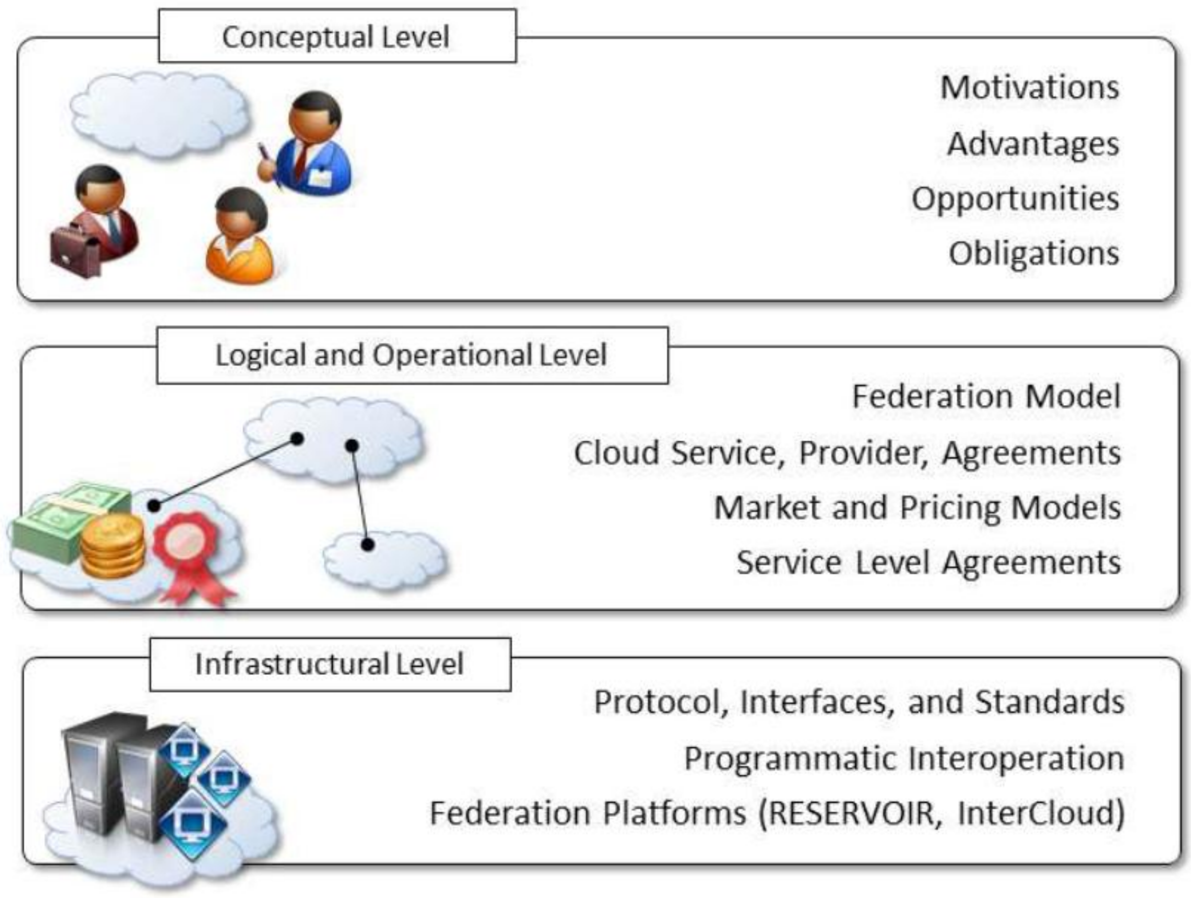
Service providers strive to make all aspects of cloud federation—from cloud provisioning to billing support systems (BSS) and customer support— transparent to customers. When federating cloud services with a partner, cloud providers will also establish extensions of their customer-facing service-level agreements (SLAs) into their partner provider's data centers.

Federated Clouds/Inter Cloud

The terms cloud federation and InterCloud, often used interchangeably, convey the general meaning of an aggregation of cloud computing providers that have separate administrative domains. It is important to clarify what these two terms mean and how they apply to cloud computing. The term federation implies the creation of an organization that supersedes the decisional and administrative power of the single entities and that acts as a whole. Within a cloud computing con-text, the word federation does not have such a strong connotation but implies that there are agree- ments between the various cloud providers, allowing them to leverage each other's services in a privileged manner. A definition of the term cloudfederation was given by Reuven Cohen,founder and CTO of EnomalyInc :

Cloud federation manages consistency and access controls when two or more independent geographically distinct Clouds share either authentication, files, computing resources, command and control or access to storage resources. InterCloud is a term that is often used interchangeably to express the concept of Cloud federation. It was introduced by Cisco for expressing a composition of clouds that are interconnected by means of open standards to provide a universal environment that leverages cloud computing services. By mimicking the Internet term, often referred as the "network of networks," InterCloud represents a "Cloud of Clouds" and therefore expresses the same concept of federating together clouds that belong to different administrative organizations. The term InterCloud refers mostly to a global vision in which interoperability among different cloud providers is governed by standards, thus creating an open platform where applications can shift workloads and freely compose services from different sources. On the other hand, the concept of a cloud federation is more general and includes ad hoc aggregations between cloud providers on the basis of private agreements and proprietary interfaces.

Cloud Federation Stack


Conceptual Level
Motivations
Advantages
Opportunities
Obligations
Logical and Operational Level
Federation Model
Cloud Service, Provider, Agreements
Market and Pricing Models
Service Level Agreements
Infrastructural Level
Protocol, Interfaces, and Standards
Programmatic Interoperation
Federation Platforms (RESERVOIR, InterCloud)

Creating a cloud federation involves research and development at different levels: conceptual, logical and operational, and infrastructural. Figure provides a comprehensive view of the challenges faced in designing and implementing an organizational structure that coordinates together cloud services that belong to different administrative domains and makes them operate within a context of a single unified service middleware. Each cloud federation level presents different challenges and operates at a different layer of the IT stack. It then requires the use of different approaches and technologies. Taken together, the solutions to the challenges faced at each of these levels constitute a reference model for a cloud federation.

The conceptual level addresses the challenges in presenting a cloud federation as a favorable solution with respect to the use of services leased by single cloud providers. In this level it is important to clearly identify the advantages for either service providers or service consumers in joining a federation and to delineate the new opportunities that a federated environment creates with respect to the single-provider solution.

The conceptual level addresses the challenges in presenting a cloud federation as a favorable soluion with respect to the use of services leased by single cloud providers. In this level it is important to clearly identify the advantages for either service providers or service consumers in joining a federation and to delineate the new opportunities that a federated environment creates with respect to the single-provider solution.

Elements of concern at this level are:

• Motivations for cloud providers to join a federation

Motivations for service consumers to leverage a federation

• Advantages for providers in leasing their services to other providers

• Obligations of providers once they have joined the federation

• Trust agreements between providers • Transparency versus consumers

The logical and operational level of a federated cloud identifies and addresses the challenges in devising a framework that enables the aggregation of providers that belong to different administrative domains within a context of a single overlay infrastructure, which is the cloud federation. At this level, policies and rules for interoperation are defined. Moreover, this is the layer at which decisions are made as to how and when to lease a service to—or to leverage a service from— another provider. The logical component defines a context in which agreements among providers are settled and services are negotiated, whereas the operational component characterizes and shapes the dynamic behavior of the federation as a result of the single providers' choices. This is the level where MOCC is implemented and realized.

The infrastructural level addresses the technical challenges involved in enabling heterogeneous cloud computing systems to interoperate seamlessly. It deals with the technology barriers that keep separate cloud computing systems belonging to different administrative domains. By having standardized protocols and interfaces, these barriers can be overcome. In other words, this level for the federation is what the TCP/IP stack is for the Internet: a model and a reference implementation of the technologies enabling the interoperation of systems. The infrastructural level lays its foundations in the IaaS and PaaS layers of the Cloud Computing Reference Model. Services for interoperation and interface may also find implementation at the SaaS level, especially for the realization of negotiations and of federated clouds.