2. The NIST Definition of Cloud Computing

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics**,** three service models, and four deployment models.

Essential Characteristics:

On-demand self-service. A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.

Broad network access. Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).

Resource pooling. The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth.

Rapid elasticity. Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

Measured service. Cloud systems automatically control and optimize resource use by leveraging a metering capability[1] at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

Service Models:

Software as a Service (SaaS). The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure[2]. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

Platform as a Service (PaaS). The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming

[1] Typically this is done on a pay-per-use or charge-per-use basis.

[2] A cloud infrastructure is the collection of hardware and software that enables the five essential characteristics of cloud computing. The cloud infrastructure can be viewed as containing both a physical layer and an abstraction layer. The physical layer consists of the hardware resources that are necessary to support the cloud services being provided, and typically includes server, storage and network components. The abstraction layer consists of the software deployed across the physical layer, which manifests the essential cloud characteristics. Conceptually the abstraction layer sits above the physical layer.

languages, libraries, services, and tools supported by the provider.[3] The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

Infrastructure as a Service (IaaS). The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).

Deployment Models:

Private cloud. The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.

Community cloud. The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises.

Public cloud. The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.

Hybrid cloud. The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

[3] This capability does not necessarily preclude the use of compatible programming languages, libraries, services, and tools from other sources.

Chapter 1

Introduction

After decades of development since the early 2000s, cloud computing today provides a wide variety of services to applications from many different domains. At the same time, there are also many confusions and misconceptions about cloud computing. The "cloudy" name, pun intended, that cloud computing uses to call itself also certainly does not help. Understanding the definition and characteristics of cloud computing will demystify the basic concepts of cloud computing and later also help us learn the fundamental principles of cloud computing. Understanding the models of cloud computing will help us appreciate the differences and connections among various types of cloud services and later help us learn how to properly choose and develop the different models of cloud services.

1.1 The Essential Characteristics of Cloud Computing

Let's start with the "official" definition of cloud computing given by the National Institute of Standard and Technology [Mell et al., 2011].

"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

Cloud computing is a model for providing users access to computing resources across the network. The resources are shared by all the users, and they can be configured based on the users' needs. They include both hardware resources such as servers, storage, and networks and software resources such as various applications and frameworks. The access to the resources is ubiquitous, meaning that it is accessible from everywhere, is convenient, meaning that it does not require much effort from the cloud users, and is on-demand, meaning

that it is available anytime when the users need the resources. Finally, the resources are provided rapidly to the users when they are needed, and released rapidly when they are not needed any more; both of which are done without requiring the users spending much effort on managing the resources or interacting with the cloud provider.

This definition captures five essential characteristics of cloud computing, which we will discuss in depth next. These characteristics are essential: on one hand, missing any one of them, cloud computing would not have achieved its success today; on the other hand, if any service that claims to be "cloud" misses any of these characteristics, it is not truly a cloud service.

Note that in our discussions, "capability" and "resource" are often used interchangeably. They both refer to what a cloud service can provide to its users for meeting their computing and data needs, for example, to run a particular application or to store and access a particular dataset. "Consumer" and "user" are also two terms often used interchangeably; they both refer to the entity that uses a cloud service. Sometimes users specifically refer to human users, but much more often the consumer of a cloud service is another program running either inside or outside the cloud. In the latter context, the term "client" is often used, which refers to the program that consumes the service provided by the server in the typical client-server model of a distributed system. Cloud computing is a form of distributed computing, which we will discuss more later in the architecture of cloud computing.

On-demand Self-service

The first essential characteristic is **on-demand self-service**.

"A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider."

Self-service means that a consumer can request and acquire resources from the cloud on her own, without having to interact in person with the service provider. The service provider offers user interface for users to interact with the service, and more importantly a program interface for user programs to access the service. The user interface is typically web based, which allows users to configure, use, and monitor the cloud service. The programming interface is provided via software development kit (SDK) for commonly used programming languages such as C++, Java, .Net, and Python.

On-demand means that the consumer can provision the resources immediately whenever she needs it. The opposite of on-demand is that the consumer gets the service only when the provider is ready to deliver it, which can take a long, unpredictable amount of time. On-demand service also implies that the consumer can dynamically request and return cloud resources because the consumer's demand may grow and shrink over time. For example, the consumer's application may have different phases where each phase requires a different amount of computing, storage, and network resources. In another example,

the consumer may be using the cloud resources to provide her application as a service to her users, and the demand of her application may vary depending on its popularity, which in turn requires different amounts of resources to run.

Broad Network Access

The second essential characteristic is **broad network access**.

"Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations)."

Cloud computing builds upon the pervasiveness of the Internet, but this essential characteristic is beyond just broad network connectivity. It is more about the support of heterogeneous clients that consumers may use to access the cloud. The clients can have wildly different software environments, including the language and libraries used to develop a client program and the operating system and runtime that support the execution of the client. The clients also have wildly different hardware configurations, including the processors, memory, and I/O devices such as storage and networking devices.

In particular, in this definition of broad network access, it makes a distinction between "thick" and "thin" clients. Thick clients are conventional computers such as laptops and workstations, which have significant local capabilities for computations and data storage. They are considered "thick" because they have all the typical hardware and software components, and thus in terms of architecture, they have a thick hierarchy of hardware and software layers. In contrast to thick clients, and more interestingly, thin clients have much reduced hardware and software components and thus much reduced processing and storage capabilities. In the simplest form, a thin client has only the interfaces for accepting computing requests and for sending the requests to the service provider for processing. They are considered "thin" because in terms of architecture they have a thin hierarchy of hardware and software layers.

In order to support such wide heterogeneity among the clients, cloud provides standard mechanisms for all the clients, no matter how different they are, to access its services. An important example of these standard mechanisms is web services, which we will discuss later in the enabling technologies of cloud.

Resource Pooling

The third essential characteristic is **resource pooling**.

"The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter)."

This characteristic captures how cloud resources are managed and allocated to cloud consumers. It has two important, complementary aspects. The first

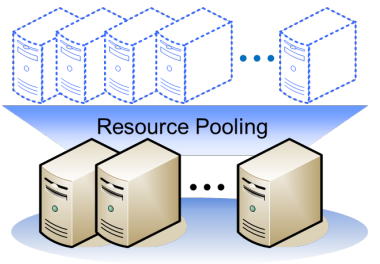


Figure 1.1: Resource Pooling

aspect is about aggregating the distributed and disparate resources in a cloud system into a single unified resource pool. A cloud system is a distributed system with many servers connected by networks. Pooling allows resources contributed by different servers to be aggregated to satisfy user requests and be allocated to users in a distinguishable manner. This is fundamental to achieve good scalability and resource utilization, because an application can use the resources from multiple servers to scale out, and the available resources on a server can be used to service any application. Resource pooling requires, first, the distributed resources from independent servers and devices to be aggregated into a single pool of resources; and second, the potentially heterogeneous resources in the pool are unified so they can be allocated and used in the same way. Virtualization is the key technology to enable both aspects of resource pooling, by hiding the distribution and heterogeneity of resources in the system. This is why it is common for cloud to provide virtual resources, instead of physical ones, to its users. We will discuss virtualization in depth in the later chapters.

The second aspect of resource pooling is that the resources in a pool are allocated to the cloud users using a multi-tenant model. **Multi-tenancy** is an important principle of cloud computing. Much like how multiple tenants share an apartment, cloud consumers share the resources available in a cloud system. A tenant can be a single user or a group of users who share the same computing requirements in terms of applications, data, security/privacy, billing, etc. For example, in a single organization, different departments can have distinct computing requirements and can be considered as different tenants of the cloud. But unlike the apartment where the rooms are dedicated to different tenants and the assignment is static, i.e., it does not change over time , in the cloud, resources are dynamically assigned and reassigned to the consumers according to their demands.

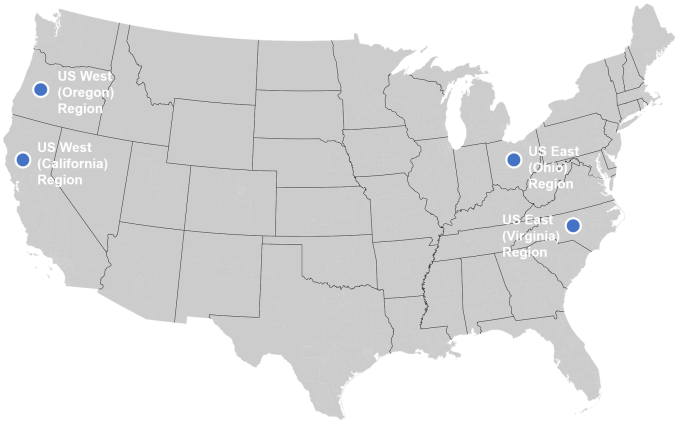
There is one more aspect to resource pooling. Generally speaking, consumers are not aware and cannot control where their assigned resources are located, because as mentioned earlier the cloud resources become indistinguishable when they are pooled together via virtualization. More specifically, consumers do not know and cannot control which specific servers are providing the resources

that they are using. Ideally, consumers also should not be concerned with the location of their assigned resources. But in current practice, consumers are often allowed to specify the location of their resources at a level of abstraction that is higher than the abstraction of individual servers or individual devices. Specifically, customers may be able to decide which country, state, or even datacenter to host their applications or store their data. For example, as we will learn later, with Amazon Web Services (AWS), customers can specify which particular Availability Zone to acquire the resources and such a zone is provided by one or multiple datacenters in a specific geographical region.

There are important reasons for consumers to have knowledge and even control of the location of their assigned cloud resources. The first reason is performance. As a distributed system with geographically distributed resources, the network latency between a consumer and her assigned resources can be significant, and it varies depending on the geographical distance between them. So a consumer may want to use resources that is the closest to her and provides the lowest latency when she interacts with the cloud service. Geographical distance also matters, and probably even more, if the consumer is using the cloud resources to run her own service and offer it to her own users. It is critical for these users to have a good experience when using her service, and this experience can be largely affected by the network latency. Therefore, it also makes sense for this consumer to choose resources according to where the users of her service are located.

The second reason is reliability. Cloud resources provide different levels of reliability. For example, Amazon S3, an object data storage service, redundantly stores objects on on multiple devices across a minimum of three Availability Zones in an AWS Region, to provide highly reliable data storage (e.g., 99.999999999% durability and 99.99% availability of objects over a given year). However, if there is a catastrophic event such as a natural disaster that affects an entire region, then maybe even the high region-level reliability boasted by S3 is not sufficient. Ironically, there have been several serious S3 outages in the past, but they were not caused by natural disasters—instead, they were caused by errors made by human operators [s3-,]. In any case, a consumer may want to replicate her data across multiple regions or even multiple cloud providers in order to tolerate such region-level failures.

The third important reason is security and privacy (to follow policies and regulations that vary by areas). This matters greatly when cloud is used to store and process sensitive data from its consumers. Different countries and even different states can have different regulations in terms of consumer data protection. For example, the California Consumer Privacy Act (CCPA) [CCP,] is a state statute intended to enhance privacy rights and consumer protection for residents of the state of California in the United States; the General Data Protection Regulation (GDPR) [GDP,] is an European Union regulation on information privacy in the European Union and the European Economic Area. Therefore, a consumer may want to choose where her data can be stored and processed according to the data protection policies that fit her needs.



AWS has a global distributed infrastructure to provide services to users from all of the world. This infrastructure is organized by Regions and Availability Zones (AZs). The figure above illustrates some of the AWS Regions in the United States. Each Region has a minimum of three isolated AZs in the same geographical area. Each AZ is based on one or multiple data centers with independent power, cooling, and physical security, and is not affected by the operations of the other AZs in the same region. The AZs in the same region are physically separated by a meaningful distance, and yet are connected with high-bandwidth, low-latency networking. Consumers can replicate their applications and data across multiple AZs to tolerate data center level failures caused by power outages and natural disasters etc.

Rapid Elasticity

The fourth essential characteristic is **rapid elasticity**.

"Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time."

Elasticity means that the resources provided to a consumer can dynamically grow and shrink according to the consumer's demand, and this scaling can be done rapidly. Elasticity is the ability to change and adapt. Here, what is changing is the cloud resources provided to a consumer, and it is adapted according to the consumer's demand. Now we understand why one of AWS' first and best-known services is called Elastic Cloud (EC2), which is probably the cloud service that has made "elasticity" the term to describe cloud computing. Noticed that the way that cloud adapts to a consumer's demand is by scaling the provisioned resources outward and inward, which refer to adding and removing, respectively, the instances of resources such as machines provided to the customer. This is different from the type of scaling that is achieved by providing

more powerful instances of resources such as machines with faster CPUs and larger memory. We will discuss scaling in detail later in the autoscaling chapter.

The second aspect of this characteristic is about how quickly the resources can be provisioned and released to consumers. Without cloud, the only way for a consumer to achieve "elasticity" is to buy physical resources and operate them on her own. This is time-consuming process involving many steps and many parties and often taking weeks or even months. For example, the consumer needs to purchase servers and wait for them to be delivered, prepare space, power, and cooling for running the servers, setting up the servers including their hardware and software as well as networking, and operate the servers to provide the needed computing and storage. When the consumer's demand falls, she cannot return the resources already purchased; the best she can do is to power them off and leave the surplus resources wasted. In comparison, provisioning resources from cloud is "rapid", as it does not require any of the steps or parties mentioned in the above process. However, the exact definition of rapidness is not clearly specified. It, in fact, evolves over time as cloud computing technology continues to improve. For example, currently, it takes minutes to spin up a new EC2 instance to handle the growing demand of a cloud application; in the near future, it is likely to be much faster. Still, the amount of time that it takes for a consumer to rent cloud resources and return them is indeed rapid compared to the alternative solution.

Rapid elasticity implies that cloud is able to meet all its tenants' demands at all times. So from a consumer's point of view, it appears that cloud has unlimited amount of resources and can always fulfil the consumer's needs. Obviously no cloud provider really has unlimited resources. So how does cloud provide this illusion of having unlimited amount of resources? A naive way for the cloud provider to meet every consumer's demand is to prepare enough resources so it has more than every customer's peak demand combined. But this would lead to severe under-utilization of resources, since each consumer's peak demand only happens occasionally, not all the time, and everyone's peak demand generally does not happen at the same time. This peak-demand-based resource provisioning scheme would severely undermine the cost-effectiveness of cloud computing.

The solution to meet all the consumers' demands cost-effectively is through **statistical multiplexing**. Statistical multiplexing is a technology originated from the communication field to allow multiple data streams to share the bandwidth of a communication link where each data stream gets a share of the bandwidth according to its current demand. In the context of cloud computing, statistical multiplexing allows multiple consumers to share the cloud resources and each consumer gets a share of the resources according to its current demand. Therefore, the cloud needs to provision only enough resources to meet the total current demands from all the consumers in order to provide the illusion of having unlimited resources. Statistical multiplexing also ensures the cost-effectiveness of cloud computing, because the sum of everyone's current demand is generally much lower than the sum of everyone's peak demand.

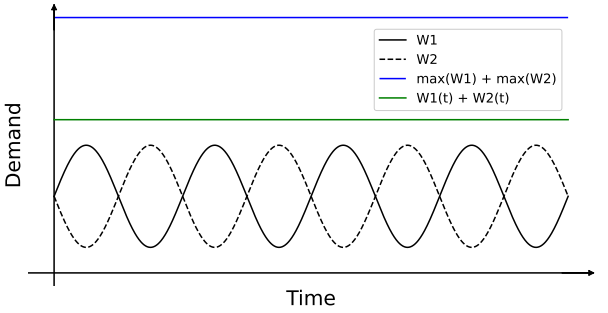


Figure 1.2: Service Models

Statistically speaking, when the cloud services a large number of independent workloads, the aggregated workload is much more stable than the individual ones, and thus making the total current demand from all the workloads more predicable. But no prediction techniques are perfect, so statistical multiplexing can only ensure that, with a good probability, cloud can satisfy every consumer's current demand at all times. Measured service

Measured Service

The last essential characteristic is **measured service**.

"Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service."

In order to support on-demand resource management, which is required by on-demand self-service and resource pooling, there must be a way to measure the amount of service, how much a consumer needs and how much is provided to the consumer, using an abstraction that is appropriate for the type of service. In current practice, such abstractions are typically specified in terms of certain units of resources, e.g., number of instances for a compute service like Amazon EC2, number of bytes for a storage service like Amazon S3, and number of messages for a communication service like Amazon SQS.

This metering capability is important for cloud computing to achieve cost effectiveness. Cloud is a field that has been largely driven by the economic successes in industry, although a great deal of academic research has laid the foundation for the enabling technologies. With this metering capability, on one hand, the cloud provider can optimize the resource allocations so that each consumer gets only what she needs and no resources are wasted; on the other hand, each consumer pays for only the resources that she uses, which is

commonly known as the **pay-as-you-go** model. Therefore, good economics can be achieved for both cloud providers and consumers. The pay-as-you-go model is another major differentiator to the conventional way of resource provisioning discussed above where the consumer has to purchase physical resources and operate them on her own. In the conventional way, the consumer has to make a upfront commitment to the equipment and its supporting infrastructure, which is often expensive and requires a long time for the investment to break even. In contrast, with pay-as-you-go, there is no upfront commitment required at all, and the consumer pay for only what she needs.

1.2 Cloud Computing Models

With the understanding of what cloud computing is, now let's examine the different models of cloud computing, and understand their differences and connections. There are many diverse cloud services being offered today, assuming they are all truly cloud, that is they all have the five essential characteristics discussed earlier. We can classify cloud services based on 1) which level of services that it provides and 2) how it is deployed, which lead us to three service models and three deployment models.

1.2.1 Service Models

Based on the level of service that cloud systems offer, we can classify them into Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).

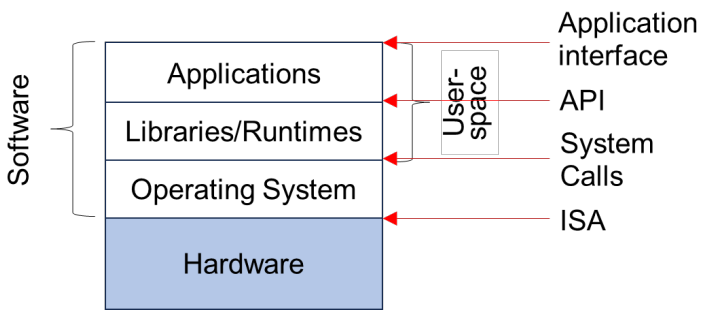


Figure 1.3: Computer System Hierarchy

To understand these different service models, it helps to review the hierarchical view of computer organization, shown in Figure 1.3 , where applications, libraries/runtimes, and operating system (OS) are the main software layers on top of the hardware. Each layer is built upon the functionalities of the layers below through the interface provided by the underlying layer, and in turn offers an interface to the layer above to utilize the functionalities that it provides.

Starting from the bottom-most layer of this hierarchy, hardware provides the **instruction set architecture** (ISA) interface for software to run utilizing the CPUs, memory, and I/O devices. OS provides the **system calls** interface for user-space software to request the use of shared resources, including CPUs, memory and I/O devices. Programming language specific libraries provide the **application programming interface (API)** for applications to utilize the libraries (and the corresponding runtime systems) to perform computations and data accesses. Finally, applications provide their application interface to users to interact with the applications and fulfill their computing needs.

If we consider the whole cloud system as a single computer (which is indeed the goal of any distributed computing system), the interfaces that the different layers provide in this software/hardware stack are the same interfaces that the different cloud service models provide. The different service models differ exactly in the interface that each model offers to cloud customers, and the differences in the interface lead to different capabilities provided to the customers and what are not.

Infrastructure as a Service (IaaS)

"Infrastructure as a Service (IaaS). The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications."

In IaaS, the cloud provides the hardware interface, i.e., ISA, to cloud consumers so they can deploy and run arbitrary software, including OSes, libraries/runtimes, and applications, on cloud resources. Infrastructure refers to the hardware, including processors, memories, storage devices, and networks. But more accurately, the hardware provided to cloud customers is the virtualized versions of the physical hardware, such as virtual machines (VMs), virtual storage, and virtual networks. As we will learn in later chapters, virtualization is critical to allow the consumers share the hardware.

With access to the infrastructure, a cloud consumer has control of the entire software stack. The consumer can decide what OSes, libraries and runtimes, and applications to use and how to use them. For example, with a VM provided by IaaS, the consumer can install the OS and library and runtime that she wants to use for her applications on the VM in the exact same way as how she does it on her own computer. At the same time, it is also the consumer's responsibility to use her chosen software properly, which requires substantial expertise and effort. For example, the consumer needs to know how to configure and manage the OS and library/runtime properly so her applications can run well on the given VM. But, the consumer does not have control over the underlying hardware, because what she is given is virtualized hardware, in the form of a VM, and the virtualization layer ensures that the consumer cannot control the underlying

hardware which is shared with other consumers.

Examples of IaaS include AWS Elastic Computing (EC2) (one of the first ever IaaS offering) and Google Computing Engine (GCE).

Platform as a Service (PaaS)

"Platform as a Service (PaaS). The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment."

In PaaS, the cloud provides the library/runtime interface, i.e., API, to cloud consumers so they can deploy and run applications created using the libraries and runtimes supported by the provider. Platform here refers to the libraries and runtimes that consumers can use to deploy and run their applications in specific languages such as Java, Python, and .Net. For example, a consumer can use the Java platform provided by PaaS to develop and run her Java applications in nearly the same way as she does on her own computer. But PaaS consumers do not have access to the the OSes and libraries/runtimes that their applications run on, which are in fact hidden from the consumers. Instead, PaaS provides user interface and tools for consumers to develop, execute, and manage their applications.

PaaS consumers cannot control the OSes and libraries/runtimes that their applications run on, which are shared by multiple consumers. At the same time, the consumers also do not need to worry about how to install and manage the underlying OS and Java platform which are entirely taken care of by the PaaS provider. By removing this burden from the PaaS consumers' shoulders, it allows the consumers to focus on their applications.

Examples of PaaS include Google App Engine (GAE), one of the first PaaS services, and AWS Elastic Beanstalk.

Software as a Service (SaaS)

Software as a Service (SaaS). The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

In SaaS, the cloud provides the application interface to cloud consumers so they can use the provided applications. For example, a consumer can use a cloud-based email service to send, receive, and manage emails similarly to how

she uses an email client on her own computer. Everything under this application interface, including the platform, OS, and hardware that the application runs on, are hidden from the consumers.

With SaaS, consumers have the least amount of control; they cannot control the applications or any other layer in the software/hardware stack. The only control that consumers may have over SaaS services are the application configurations that allow user-specific customization. For example, with a cloud-based email service, consumers may be able to configure labels, filters, and forwarding rules for emails based on their personal preferences. If a SaaS service meets what a consumer's needs, it can be very convenient for consumers, because the consumer does not need to do anything to set up the application or the platform, OS, and hardware that the application runs on.

Many applications that we use today are SaaS, e.g., consumer applications such as Google Apps and enterprise applications such as Workday.

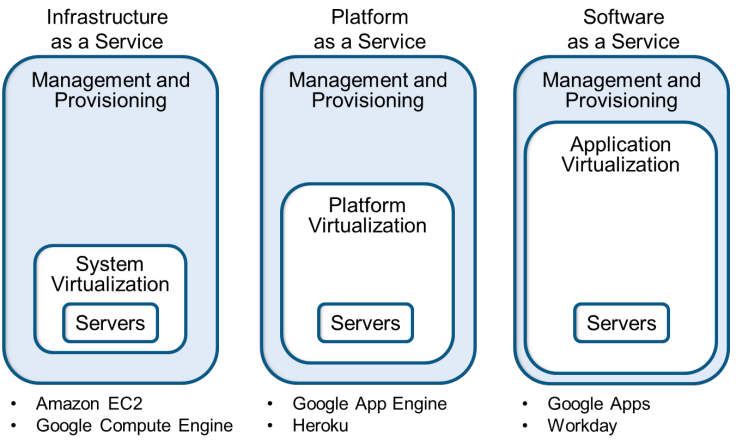


Figure 1.4: Service Models

The Evolution of Service Models

Recall the hierarchical computer organization diagram where each layer in the hierarchy builds upon the layer below, e.g., applications build upon libraries/runtimes (i.e., platforms), libraries/runtimes build upon OSes, and OSes build upon hardware (i.e., infrastructure). The three service models discussed above can also form a hierarchy, where a SaaS service builds upon a PaaS service or an IaaS service and a PaaS service builds upon an IaaS service. In fact, it is very common for a SaaS provider to utilize PaaS or IaaS resources, instead of acquiring and operating its own resources, to deliver its applications to the SaaS consumers so it can focus on the application development and delivery while enjoying all the benefits of PaaS or IaaS. At the same time, a PaaS provider

can also utilize IaaS resources to deliver its platforms to PaaS customers. For example, Reddit, a discussion forum SaaS can be hosted on Heroku, a PaaS provider, which in turn uses AWS' IaaS resources to provide PaaS services.

Cloud computing is all about offering computing as a service. In addition to the above three basic service models, there are many other types of services. One might even say, anything can be offered as a service in the cloud, as in "X as a Service."

Not all the cloud services can fit in the IaaS, PaaS, and SaaS classification. A noteworthy one is Storage as a Service. One type of storage services such as Amazon S3 provide cloud storage to applications. They are as fundamental as IaaS services, and are in fact commonly utilized by cloud applications, but users cannot deploy their software on them, so they do not really match the definition of IaaS. Another type of storage services such as Dropbox allow users to backup their data from their personal computer and devices to the cloud. If we consider data backup as an application, then one can say this kind of storage services belongs to the SaaS category.

Databases and big data can also be offered as services, There are a wide range of database services offered by the cloud, from NoSQL database services such as Amazon SimpleDB to relational database services such as Azure SQL. These services provide the database software as a service, so we can consider them as SaaS. At the same time, they allow users to run queries implemented with the SQL language on the data so they also act as a PaaS.

Another interesting example is Big Data as a Service. For example, Amazon Elastic MapReduce allows users to deploy and run MapReduce programs for processing and analyzing big data. These services provide the capabilities to not only use the MapReduce software but also run MapReduce programs. Hence, they can be considered as SaaS and PaaS.

1.2.2 Deployment Models

In the above discussions, we differentiate cloud services based on the level of service they are providing. Another important way to classify the cloud computing models is based on how the services are deployed. There are two basic deployment models, public cloud and private cloud. The main differences lie in three key factors to the model deployment: 1) the users, 2) the stakeholder who owns, manages, and operates the cloud, and 3) where is the cloud located.

Public Cloud

"Public cloud. The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider."

A public cloud is provided for open use by users from the general public. It is like the Internet which can be used by anyone from the general public. Public

clouds can be owned, managed, and operated by different types of organizations that have an interest in serving the general public, although today the best known public cloud providers are mainly industry companies. A public cloud is located on the premises of the cloud provider, specifically the cloud provider's datacenters.

Private Cloud

"Private cloud. The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises."

A private cloud is provided for exclusive use by consumers from a single organization such as a company, a university, or a government agency. It is like the private network which is exclusively used by users from a single organization. For example, a company's private cloud is provided for the business activities from its employees, and a university's private cloud is provided for the research and education activities of its faculty and students.

Even though all the users are from the same organization, there is still multi-tenancy, as the different users or the different organization units (e.g., departments, business units) may have different computing demands and need to share the private cloud resources.

For a private cloud, the party that owns, manages, and operates the cloud can be either the organization that the cloud is provided for or a third party. Hence, there are different combinations of options. The most straightforward option is that the private cloud is owned, managed, and operated by the user organization. In this case, the private cloud is physically located in the organization's premises, which is called a on-premises private cloud.

However, managing and operating a cloud infrastructure requires substantial expertise and investment, and it is not always the best option for organizations that need a private cloud. Instead, an organization can delegate the infrastructure operations, including managing and maintaining of all the hardware, software, and networking components, to a third party such as a commercial cloud provider who has the expertise and can operate the infrastructure cost effectively. The organization can also choose to delegate the management of the private cloud in terms of policies for user management, resource allocation, and billing etc. to a third party, so it can focus on its own business development needs. The organization can even rent the infrastructure for its private cloud from a third party, instead of owning the infrastructure, so it can avoid the large upfront investment for resource ownership. With these other options, an organization's private cloud does not have to exist on the premises of the organization; they can run off-premises on the third-party vendor's datacenters, which we call off-premises private clouds.

Hybrid Cloud

Public and private clouds provide users different benefits on several important aspects that affect the quality of cloud services delivered to the consumers. Here we provide a high-level, general comparison between these two deployment models.

In terms of performance, generally speaking, public cloud can provide much larger-scale resources to meet the consumers' performance need than private cloud, simply because public cloud providers have more resources and wider geographical distribution of the resources. When there is a sudden surge of demand from a consumer, a private cloud may not be prepared and become quickly oversubscribed, but a public cloud can better handle the surge by utilizing its much larger pool of resources. New consumers demand can also show up unexpectedly from a certain geographical area, and the wider distribution of a public cloud's datacenters can better handle it by providing the consumers with resources closer to them.

Private cloud, generally speaking, can provide more predictable performance as it has better knowledge and control on what applications run on the cloud and how the resources are used. Performance predictability is becoming an increasingly important requirement for modern cloud applications. Delivering a consistent performance, e.g., a good response time to all requests, is often more important than simply being fast on average but with a considerable number of slow outliers, which is called a "long tail'". Knowing the characteristics of the applications such as their CPU, memory, storage, and network demands, allows the cloud to find the best resources for these applications. In particular, lack of performance predictability for one application is often caused by performance interference from the other applications. Even though cloud makes its best effort to provide isolate the applications that share its resources, it cannot guarantee perfect performance isolation. In this regard, provide cloud can provide stronger performance isolation and therefore better performance predictability to applications based on its knowledge about the application characteristics. For example, it can schedule applications with complementary resource demands such as CPU and memory to the same host to avoid the competition for the same type of resources and thereby provide better performance predictability.

In terms of reliability, public cloud again can utilize its size and geographical distribution to help its consumers tolerate large-scale failures such as datacenter-level and region-level failures, e.g., by replicating applications and data across datacenters and regions, which are difficult to handle for private cloud which is often smaller and more centralized. At lower level of the location abstractions such as server level and device level, private cloud may be able to provide higher reliability because it has a better knowledge about the applications characteristics and can find better resources and better fault-tolerant mechanisms to support their reliability requirements.

In terms of security and privacy, private cloud is generally better as it has

better knowledge and control of its users and applications. A private cloud can limit its use to trustworthy users and applications, and restrict access from external networks. In comparison, a public cloud is open to the general public, and a consumer has no knowledge or control of who else is sharing the resources including processors, memory, storage, and networks with her. Cloud provider, again, cannot guarantee perfect isolation, and the imperfections in isolation can be discovered and exploited by attackers to compromise a consumer's applications and data or steal the consumer's secrets. For example, side-channel attacks (e.g., Meltdown and Spectre [Metz and Perlroth,]) extract information leaked through shared resources (e.g., shared CPUs) to learn about private data. Moreover, the scale and cost-effectiveness of cloud can also be utilized by malicious users to launch large attacks previously infeasible [Galante et al.,].

In terms of cost effectiveness, there is a tradeoff between public cloud and private cloud. On one hand, public cloud requires zero upfront cost from a consumer whereas private cloud needs a large upfront investment from the consumer to acquiring the infrastructure and getting it up and running. Hence, public cloud is great for users such as startup companies which want to start small and do not have a good projection of their future demands. On the other hand, private cloud can be more cost effective in the long term because the unit resource cost is typically cheaper and the upfront cost of acquisition becomes insignificant compared to the long-term total cost of ownership.

Public cloud and private cloud complement each other in the aforementioned important aspects. Therefore, it is desirable to utilize both deployment models and leverage their complementary strengths for cloud consumers. This leads to the hybrid cloud deployment model.

"Hybrid cloud. The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds)."

A hybrid cloud has both a public cloud component and a private cloud component, per the aforementioned definitions of these two deployment models. However, the key to a hybrid cloud is the technology that allows these distinct public and private cloud components to be combined into a seamless infrastructure where the users' applications and data can be flexibly placed in either component and even migrate between them as needed.

For example, a consumer can start with a private cloud for her applications, and scale out to the public cloud when the demand surges beyond the private cloud component's capacity. With this so called cloud bursting method, the consumer can keep the total cost low, but it requires the technology to dynamically deploy applications and data onto public cloud and distribute demand between the public and private cloud components. In another example, a consumer can use public cloud to provide backups of applications and data on the private cloud, in order to tolerate large-scale failures that impact the private cloud component. It requires the technology for efficient application and data

backup from private cloud to public cloud and quick failover when failures take place. In a final example, a consumer can place her applications and data that are not sensitive in public cloud and those that are sensitive in private cloud so that she can optimize the tradeoff between cost and security/privacy. This example requires the technology to distinguish and distribute applications and data based on their security/privacy needs while still allowing them to be collectively used to meet the customer's computing needs.

Chapter 2

Historical Perspective

Even though cloud computing is still a relatively new paradigm, the fundamental idea behind cloud computing is nothing new. In fact, it is a dream that has been held by computing pioneers since the 1960s, at the dawn of the modern computing era. In 1961, John McCarthy, who is one of the founding fathers of artificial intelligence (AI), made the following prediction during his lecture at the MIT centennial,

"If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility ... The computer utility could become the basis of a new and important industry."

This "computing utility" that John McCarthy was envisioning, back in 1961, is exactly what cloud computing aims to deliver. Cloud computing is in fact one of the many efforts that have been made to deliver computing as a utility, much like the other utilities such as electricity and water that we use everyday. To use electricity, you do not need to run a generator in your house; instead, you simply plug your appliance into an electrical outlet and electricity flows from the power plant to your appliance through the power grid. To use water, you also do not need to dig a well in your backyard; instead, you simply turn on your water faucet and water flows from the water processing plant to your faucet through the pipes. With computing as a utility, you will not need a powerful computer on your desk to run your applications; instead, you will be able to run your applications on the resources provided by remote data centers through the computer network.

Indeed, there have been several major efforts in the past to deliver computing as a utility. In this chapter, we will discuss these past efforts of computing as utility. The goal of providing this historical perspective is to understand cloud computing in light of its earliest phases and subsequent evolution, which will sharpen our vision of the present as well as the future of cloud computing. By comparing cloud computing in its current form to the previous related paradigms, we will gain a deeper understanding of the common principles of

computing as utility and at the same time appreciate the important differences in cloud computing that have made it so successful. In particular, we will learn how virtualization has always been a key enabler to the different efforts of computing as utility.

2.1 Time-shared Mainframes

In the 60s and 70s, mainframe computers were considered the "supercomputer" of the time. They were powerful and expensive at the time, used by many large organizations such as banks and government agencies. The need to share the mainframes by multiple customers motivated the invention of not only the first time-sharing operating systems [Lett and Konigsford, 1968] but also the first virtual machines [Creasy, 1981] (which today we refer to as the classical virtual machines). Time-sharing operating systems allow many applications to share a single mainframe and remain interactable with their users. VMs further allow many OSes and the applications that they host to share a single mainframe.

Terminals, another major invention at the time, provide customers basic input (keyboard) and output (screen) for interacting with the remotely located, shared mainframes. By today's standards, these terminals are "dumb", because they do not have much local processing capabilities. But they were sufficient to provide customers efficient access to the mainframes. In fact, a 70s-era mainframe was able to support tens of thousands of terminals simultaneously.

Although mainframes were eventually largely replaced by personal computers, these inventions that came out of the mainframe era are still hugely influential to today's computing systems, including the clouds.

2.2 Thin Clients

Fast forward to the 90s. The technologies behind the success of personal computers, x86 hardware and software, became increasingly powerful. That is when people started to think about computing as a utility again. This time, the resources to be shared were x86-based servers. In the 90s, the Internet also became pervasive, which made sharing resources across the network a possibility. These technologies motivated the Thin Clients movement. The idea of thin clients is to place shared x86 servers across the Internet; users can perform computing on these servers through Internet from their thin clients, instead of owning and managing their own servers. Thin clients are considered "thin", because they do not require much software or hardware as they do not need much local processing capability. Since computing is done on remote servers across the network, thin clients, which are also called network computers, need to provide only the user interface and network interface in their simplest form.

As part of this movement, x86 virtual machines (notably Xen [Reed et al., 1999]) were created to allow different applications to share the servers. Like today's

public cloud, thin clients based network computing enable arbitrary applications from different users to share the resources. A key challenge is how to safely run these untrusted applications on shared servers and ensure that no application can compromise the others or even the system. VM technology was developed to provide the necessary isolation among the resource-sharing applications and between the applications and the system. Different from the classical VMs developed for mainframes, these VMs were created for the x86 architecture which was unfortunately much more difficult to virtualize than the mainframe architecture.

At the same time, virtual (remote) desktops were created to allow users to access the servers from different client platforms. A virtual desktop provides its user an interface that is identical to a local desktop for the user to conveniently interact with the remote server for computing and data accesses. The virtual desktop software can run on physical thin or thick clients with heterogeneous software and hardware environments, thereby providing users broad access to the shared servers. Some noteworthy examples of these virtual desktops include Citrix' virtual desktop for sharing Windows servers and the platform-independent virtual desktop software, Virtual Network Computing (VNC), both of which are still influential today.

2.3 Utility Computing

In the late 90s, another form of computing as a utility achieved some great successes, and it was appropriately named "utility computing", true to its actual purpose. A distinct characteristic of this utility computing paradigm is that it went beyond the sharing of individual servers and offered shared data centers as a service. It became possible due to growing importance of data centers for providing computing capabilities with fast Internet access. A data center has racks of compute and storage servers and the supporting networking, power supplies, and cooling etc. Technologies were developed to enable customers to share this expansive and expensive data center infrastructure. HP was one of the leaders in this area with its Utility Data Center, which allows a customer to construct a virtual data center, including servers, storage, networking, and firewalls out of the shared data center resources. By offering such virtual data centers for the exclusive use by specific customers, HP essentially became one of the first private cloud providers.

Utility computing also saw successes in supporting some specific types of applications, such as movie film rendering. At that time, computing started to play an important role in film making, especially for animated movies which require a lot of compute-intensive rendering. The most noteworthy success example is HP's Utility Rendering Service (URS) for the making of the Shrek 2 movie. URS delivered about 10 percent of Shrek 2's 10 million rendering hours via a data center consisting of 500 dual-processor HP Linux servers, each configured with 4GB of memory and 72GB of local disk, over 4 Terabytes of

NFS-based storage, and connected to the DreamWorks studio via a 1G bps network link. In this example, HP in essence created a hybrid cloud combing DreamWorks' own infrastructure with HP's utility data center for rendering the Shrek 2 movie.

2.4 Grid Computing

In the early 2000s, grid computing emerged in the scientific community as a solution to meet the increasing computing demand from scientific applications by sharing high-performance computing (HPC) resources across different organizations such as universities, labs, and supercomputing centers. We have many geographically distributed HPC resources owned, managed, and operated by different organizations. However, the demand for computing for sciences, including physics, chemistry, biology, astrophysics, earth sciences, life sciences, environmental science, and medicine, has grown so much—some people consider computer simulations as the third pillar of science, after theory and experiment—that no single HPC system is able to meet the resource demand of the scientific applications, such as high-energy physics, molecule dynamics, seismic modeling, weather forecasting, and protein folding. This ever-growing need of computing resources motivated the development of grid computing systems which can aggregate the geographically distributed resources from different organizations to collectively meet the resource demands of scientific applications. It is called grid computing because of the analogy to a power grid, where Internet is the power transmission system, network jacket is the power outlet, and users are charged based on the computer resources (processor cycles, storage bandwidth, application time) that they use just like how they pay for the electricity that they use.

A unique challenge to grid computing that does not exist in the other paradigms and not even in today's cloud computing is the need to share resources across administrative domains. The users of a grid computing system and the resources contributed to the system are under the administration of different organizations where the users and resources come from. Each administrative domain has its own user and resource management mechanisms and policies, which creates difficulties for cross-domain job executions and resource accesses. As a simple example, the same user of a grid computing system will have different identities (such as user ID and group ID) in two different organizations of the system and be managed under different policies (such as access control, resource limit). Consequently, the job that the user launches on one organization's resource cannot access the data that the user has on the other organization's resource.

Grid computing addresses the problem of cross-administrative-domain resource access by creating a virtual organization over the physical organizations of the system so that users and resources can be managed consistently under the same policies across the entire grid computing system. Following the previ-

ous simple example, the same user of the grid computing system will have the same virtual user identity, implemented by specific physical identities of the underlying physical organization, which the user can use to access resources across the system under the same policies, implemented by specific physical resource allocation and access control mechanisms of the underlying physical organizations.

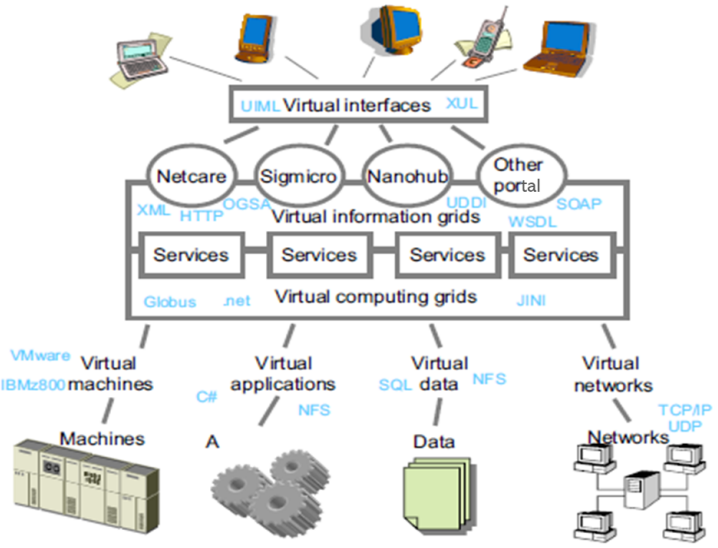


Figure 2.1: Architecture of the virtualized grid computing system, In-VIGO

One particular grid computing system that brought us much closer to cloud computing is the In-VIGO system [Adabala et al., 2005]. Different from the other grid computing systems at the time, In-VIGO is unique in its extensive use of virtualization, including virtual machines, virtual networks, virtual data, and virtual applications, as well as automatic management of the resources to deliver computing as a utility to scientific applications. It employed VMs to deliver applications with their required execution environments (including OS and libraries/runtimes) so they can run on the diverse resources available in a grid computing system and share the resources with good isolation. It developed virtual networks to provide seamless connectivity to the VMs distributed over the heterogeneous and sometimes unreliable network environments in the grid computing system. It created grid-wide virtual file systems to allow applications to access their remotely stored data across the grid computing system in the same fashion as how they access their local data. It also virtualized applications by enabling different instances of the same application to be created with versatile configurations according to the user specifications. Finally, it employed autonomic management so the entire system can automatically manage and optimize itself according to the users' requirements in performance,

reliability, and security. As we will discuss next, these are truly the enabling technologies of cloud computing.

2.5 Cloud Computing

Finally, we arrived at cloud computing in the mid 2000s. In 2006, Amazon launched Amazon Simple Storage Service (S3) and the Amazon Elastic Compute Cloud (EC2). In the same year, Google launched Google Docs. In 2008, Google released a preview of Google App Engine. In 2010, Microsoft launched Microsoft Azure.

Having reviewed the previous efforts of computing as a utility, now we understand that cloud is really the latest milestone in the long journey. If we compare cloud computing to all the other past efforts, we can see they share the fundamental principles of computing as a utility, such as multi-tenancy, resource pooling, and statistical multiplexing. Virtualization has always been a key enabling technology of computing as a utility, from virtual machines, virtual desktops, to virtual datacenters, and virtual organizations. At the same time, cloud computing is unique in its five essential characteristics; the others either do not have all these characteristics or cannot adequately deliver them. This explains why cloud computing is so much more successful than our past efforts of computing as a utility.

Behind the success of cloud computing is the amazing progress that we have made in both hardware and software. On one hand, we have increasingly powerful hardware which provides increasingly more capacity and capabilities that can be shared by many customers. In terms of capacity, processors are providing faster performance by integrating more cores and delivering higher parallelism; memory and storage are getting faster and bigger; and network bandwidth also continues to improve. At the same time, hardware resources are gaining new capabilities. CPUs are getting new extensions to support new tasks, such as the Virtualization Technology extension for VMs and the Advanced Matrix Extensions for deep learning. New specialized processors, i.e., accelerators such as GPUs, FPGAs, and TPUs are are being developed and employed to speed up specific tasks such as data analytics and machine learning. New memory and storage devices such as persistent memory and smart SSDs are being developed and incorporated into the memory/storage hierarchy, making it deeper and more capable. Software-defined networking is enabling dynamic, programmatically managed network configuration to improve network performance.

On the other hand, we also have increasingly mature software technologies that can take advantage of the power of hardware to provide computing as a utility. In particular, virtualization, including virtual machines, virtual storage, and virtual networks are extensively used by clouds to support resource pooling and rapid elasticity, which we will discuss in the following virtualization chapters.

Service-oriented architecture (SOA) enables cloud applications to be developed with loosely coupled components which are offered as services, instead of as one monolithic system. Each service provides a discrete functionality that can be accessed remotely via a well-defined interface. The interface that a service provides abstracts away the underlying complexity of service implementation and allows it to be updated and maintained without affecting the other services. Different services can be put together to form applications or compose new services. The services and their corresponding consumers communicate with each other through standardized protocols over a network. The *de facto* way to implement SOA is with web services. As the name suggests, web services enable the services (and their consumers) in an SOA system to communicate and collaborate over the web. This is why AWS is called Amazon Web Services; in fact, almost all cloud services today are implemented as web services. We will learn more about web services when in our IaaS chapter.

Finally, autonomic computing enables self-management of cloud systems to support on-demand self-service and rapid elasticity by eliminating the need to involve human administrators and to support the automatic resource control and management in measured service. An autonomic computing system has the ability to automatically manage itself, i.e., self-management, without human intervention, according to the high-level objectives such as performance and reliability specified by users. Self-management has several key aspects: 1) Self-configuration: the ability to automatically configure the system such as the resource configurations; 2) Self-optimization: the ability to automatically optimize the system according to the given objectives such as performance, availability, and resource utilization; 3) Self-healing: the ability to automatically discover and recover from failures including both hardware and software failures; and 4) Self-protection: the ability to automatically identify and defend against arbitrary attacks.

On-demand self-service requires the cloud system to automatically configure the resource allocation to consumers whenever they are needed, without having to involve human operators. Rapid elasticity further requires the cloud system to automatically scale in and out the resource allocations according to the users' demand; any human intervention would slow down the process tremendously and not be able to achieve the desired rapidness of elasticity. Metered service mentions that cloud needs to automatically optimize its resource use, i.e., self-optimization, which means on one hand resources need to provisioned sufficiently to meet consumers' demands and on the other hand no resources should be allocated unnecessary and wasted. As a concrete case of self-management, in our autoscaling chapter, we will learn how to automatically scale in and out the resources for a cloud application on demand.

Chapter 3

Introduction to Virtualization

As we have already seen in the previous chapters, virtualization is a fundamental enabling technology to cloud computing as well as the other paradigms for computing as a utility. In this chapter, we will discuss at a high-level what virtualization is, what are the different forms of virtualization, and what are the common principles and benefits.

3.1 Definition of Virtualization

Let's first try to define virtualization. In computing, "virtual" refers to artificial objects that have the essence or effect of the real objects, but are in fact not real. What this definition really means is that a virtual object presents the same interface as the physical object—after all, the only way to interact with an object is through its interface—but its implementation is not the same as the physical object. These virtual objects are often created by a computer system to help the system control access to shared resources, i.e., the virtual objects can be given to different consumers while sharing the underlying resources provided by the physical object, and the computer system controls the resource sharing by controlling the allocation of resources to the virtual objects.

Moreover, in computing, "virtual" and "logical" these two terms are often used interchangeably. For example, logical memory is the virtual memory presented by an OS to a process upon the physical memory. A logical volume is a virtual volume presented by the logical volume manager upon to physical volumes. A logical file system is a virtual file system presented by an OS upon the physical file systems.

Virtualization is a layer of indirection, inserted into the existing system hierarchy, in order to solve problems that the current implementation of the layers cannot address. We can envision any computer system as a hierarchy of

layers, where each layer provides its functionality to the layer above through its interface and consumes the functionality provided by the layer below through the underlying interface. A layer of indirection can be inserted in between any two layers in this hierarchy to achieve virtualization. The layers below the virtualization layer provide the resources that the system wants to virtualize. The layers above the virtualization layer are consumers of the virtual resources that the system provides. The virtualization layer implements the solution to the problem that the existing system stack cannot address.

A key advantage of virtualization-based solutions is transparency. The layer of indirection that implements virtualization uses the original interface provided by the underlying layer and provides the original interface that the layer above expects to use. Therefore, it does not require any change to the existing layers in the system hierarchy. It merely provides an indirection of the requests from the layer above to the layer below with an implementation that addresses the problem that the existing layers cannot solve.

3.2 Forms of Virtualization

There are three major forms of virtualization: virtual machines, virtual storage, and virtual networks. They are orthogonal to each other. Figure 3.1 illustrates how they are all utilized to support computing in a system such as the cloud: VMs provide applications their desired execution environments and allow them share the resources with good isolation; virtual storage provides VMs and their hosted applications with efficient data storage and accesses; virtual networks provide cloud services and their consumers strong isolation while sharing the underlying network resources.

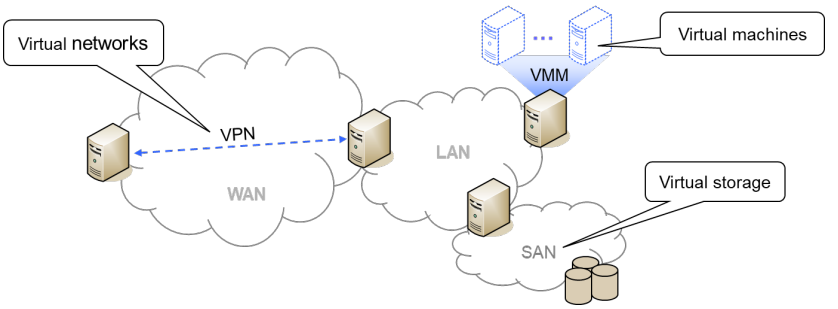


Figure 3.1: Illustration of virtual machines, virtual storage, and virtual networks used together in a cloud computing system

Here we briefly look at some examples of these different forms of virtualization, and discuss the common principles and benefits of virtualization. In the following lectures, we will learn virtual machines, virtual storage, and virtual networks in detail.

Virtual Machines

Consider the computer system hierarchy that we discussed before as shown in Figure 3.2. While the existing OS layer in the hierarchy often provides good support for the applications to share the underlying hardware, there are important scenarios where it becomes insufficient. For example, the OS may not be able to provide enough isolation between the applications sharing the hardware; the users may want to run applications developed for different OSes on the same hardware. To address this insufficiency, a layer of indirection can be inserted between the OS layer and the hardware layer to virtualize the hardware and create virtual machines to the upper layers. Virtual machines (VMs) present virtual hardware (CPUs, memory, I/O devices) to the software stack (applications, libraries/runtimes, OSes) running inside each VM, and they share the underlying hardware that the physical machine provides. The VMs can each run a different OS and allow them to share the underlying hardware; the virtualization layer that implements the VMs can also be carefully designed to provide strong isolation between the applications running on different VMs.

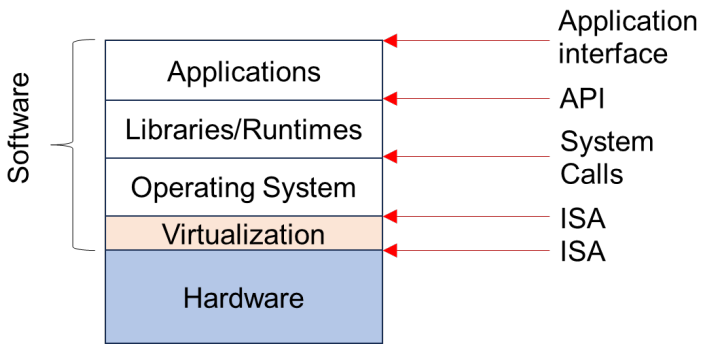


Figure 3.2: System virtualization creating virtual machines

Virtual Storage

Storage virtualization presents virtual storage resources (virtual memory, virtual disks, virtual file systems, etc.) upon the shared physical storage resources. It is implemented as an indirection layer inserted somewhere in the storage software/hardware stack. The layers below are the storage resources that are virtualized, and the layers above are the consumers of the virtual storage resources.

Take virtual memory shown in Figure 3.3 as an example. It is a layer of indirection, implemented by the memory management unit (MMU) hardware and the OS, between the processor and the memory. It presents virtual memory address spaces to the processes running on the processor, and allows them to share the address space of the physical memory. Each process sees an indepen-

dent, continuous virtual address space for storing its code and data, while in reality it is sharing the physical memory with other processes and its allocated physical memory is not continuous. To implement this memory virtualization, OS maintains a page table for each process to map its virtual addresses to the physical addresses. The MMU uses the page table to translates the virtual memory addresses used by the process while running on the processor to the physical addresses that can be used to access the physical memory. Memory virtualization allows many processes to transparently share the physical memory—how the memory is shared is entirely hidden from the processes—while providing strong isolation to the memory-sharing processes—no process can even reference an address that belongs to any other process. We will learn more about virtual storage in the later storage virtualization chapter.

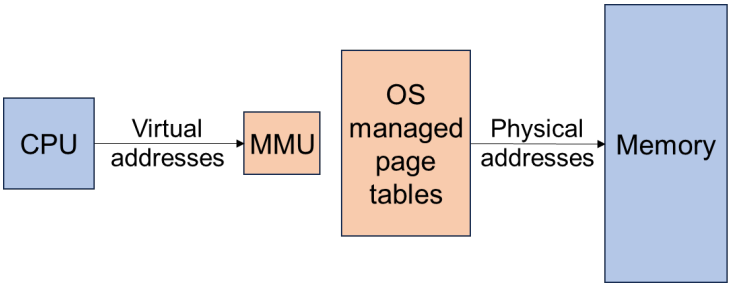


Figure 3.3: Memory virtualization with memory management unit (MMU) and OS

Virtual Networks

Network virtualization presents virtual networks upon the shared physical network infrastructure. Compared to other types of systems, networks have well defined layers in its software/hardware stack; the Internet has five layers: application layer, transport layer, network layer, data link layer, and physical layer. But, at the same time, a network system always involves multiple stacks from the hosts and network devices connected by the network. Network virtualization is implemented as an indirection layer somewhere in this 5-layer network network hierarchy across the software/hardware network stacks in the system.

For example, virtual private network (VPN) virtualizes the data link layer by providing a virtual private link for the client to access a remote private network over the public network. As illustrated in Figure 3.4, the client is connected to the server through the public network which is insecure and cannot allow the client to access the resources located in the private network. The VPN software on the client takes the private IP datagram used by the client-side application, encrypt it, and encapsulate it in a public IP datagram so it can be securely transmitted across the public network to the server. The server then extracts the private IP datagram from the received public IP datagram by decapsulating

and decrypting it, and then forwards it to the private network so it can reach the private resource that it intends to reach. We will learn more about virtual networks in the later network virtualization chapter.

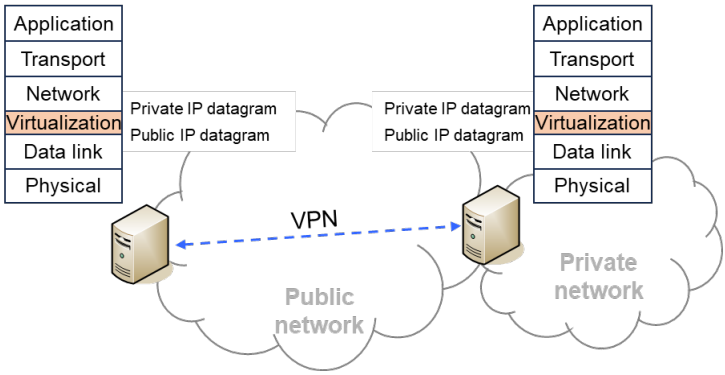


Figure 3.4: Virtual private network (VPN) allows a client to securely access a remote private network over a public network

3.3 Benefits of Virtualization

Overall, we can summarize the benefits of virtualization in the following aspects.

First, virtualization enables resource consolidation and sharing, i.e., the resource pooling characteristic of cloud computing. Resource consolidation is the process of converting many, disparate resources, which are typically dedicated to different users, into aggregated, unified resources, which are shared by the users. Consolidation can substantially improve application scalability and resource utilization, as resources can now be pooled to meet an application's demand and unused resources can be reallocated to others.

Second, virtualization allows great customizability and mobility of the resources. Virtual resources are typically implemented in software and represented as data, which are much easier to manipulate and move around than physical resources implemented in hardware. For example, customizing a physical machine is time-consuming, but customizing a VM is just a matter of editing the configuration of the software process that implement the VM; moving a physical machine is laborious, but move a VM is just a matter of copying the data that represents the VM.

Third, virtualization provides isolation to the computing tasks consuming the virtual resources, compared to running them directly on the physical resource. The indirection layer can implement a variety of isolation mechanisms (e.g., address space isolation, control of privileged operations) to enforce the isolation among the resource-sharing tasks in terms of performance, mitigating the performance impact of one computing task to the others, reliability, pre-

venting the failure of one task from affecting the others, and security, protecting one task from being compromised by the others.

Finally, virtualization can reduce the management complexity of the system. Manageability is a serious problem faced by resource providers which have to manage a large amount of different types of resources and support the diverse demands from many different consumers. The management overhead can significantly impact the scalability of the system. The resource consolidation enabled by virtualization can substantially reduce the complexity of resource management, as there are now fewer, disparate physical resources to manage and the virtual resources can be managed in the same way as part of the same pool. Moreover, the management of virtual resources (often implemented in software) is much simpler and faster compared to the management of physical resources (which often requires manipulation of the hardware).

Infrastructure as a Service (IaaS)

In this lecture, we use Amazon Web Services (AWS) to study IaaS and learn how to develop a cloud application (app) using IaaS resources. We choose AWS because it provides representative IaaS services that can be found in other cloud providers; it is also the first IaaS provider and today the largest cloud provider in the world.

There are three fundamental types of IaaS services that we typically need to develop a cloud app: compute (AWS EC2), which provides computing resources for executing the applications, storage (AWS EBS, S3), which provides storage resources for storing the data that applications needs to access, and messaging (AWS SQS) for the distributed applications and their distributed components to communicate across the network.

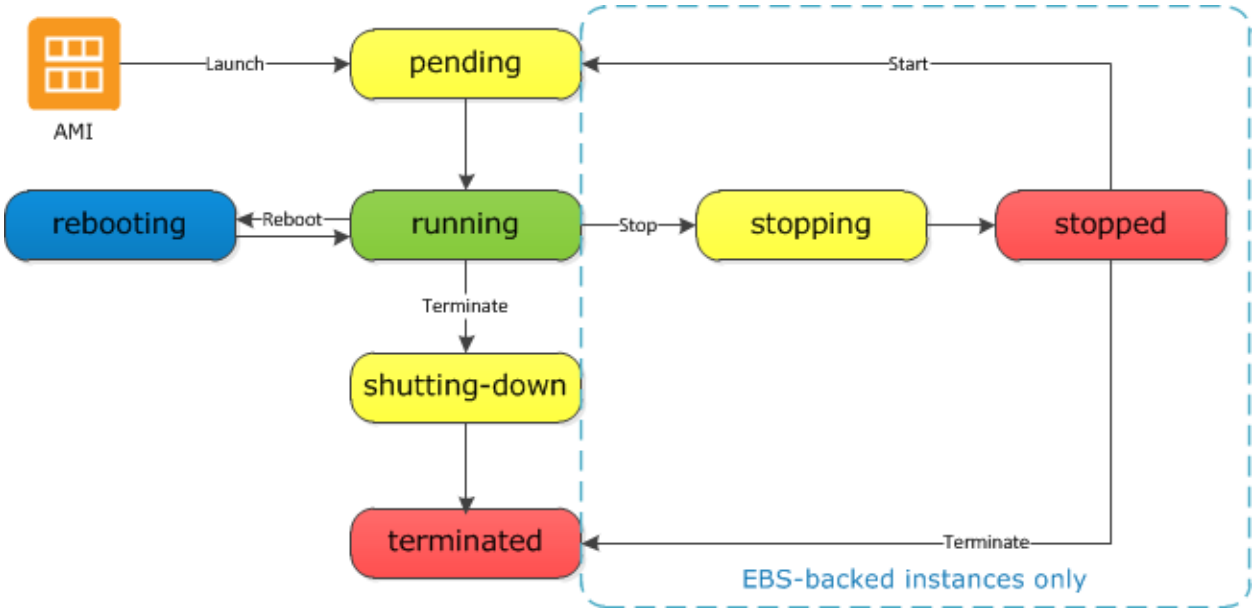
Before we discuss these IaaS services in detail, I want to introduce a success story of AWS. Animoto.com is an SaaS providers on April 18th 2008, it launched a viral ad campaign on Facebook, which led to a huge spike in the demand of its service: its number of users grew from 5,000 to 750,000 in just three days. Fortunately, Animoto.com was hosted on AWS, and it was able to handle this sudden surge of demand by quickly growing its number of EC2 instances from 40 to 4000. Even though it is a really outdated story, it still serves as a good example of the power of cloud computing. Without the cloud, a company like Animoto.com would not be able to deal with the sudden change of demand quickly and cost-effectively.

In the following, we will learn how to use the basic IaaS resources. Although the discussions here are based on AWS, we can also find similar IaaS resources from the other providers.

EC2 provides virtual environments for computing using virtual machine instances. Instances come in different types, each configured with a certain amount of CPUs, memory, storage, and network capacity. As discussed in Lecture 1, users can specify the location of their instances at some higher level of abstraction, regions and availability zones. Each region is a separate geographic area. A region has multiple availability zones. Each zone can fail independently from the other zones.

Instances are instantiated from images. The relationship between instances and images is analogous to the relationship between processes and programs. Multiple instances of different types can be started from the same image. AWS provides templates for the images, called AMI. Users can also create their own custom AMIs.

During the lifecycle of an instance, it goes through various state as shown in the state diagram.



Instances that run a distributed application need to communicate with one another. Some of the instances also need to communicate with programs outside of the cloud, e.g., servicing requests received from the clients of the application. An instance always gets a private IP and an internal hostname. These addresses are visible within the cloud, and are useful for communications among the instances. An instance also gets a public IP and an external hostname. These are visible outside of the cloud, but they are not persistent---they are automatically released when the instance is stopped or terminated.

Having a persistent external address is important to an instance that needs to communicate with the outside world. AWS provides these persistent IP addresses as a service: EIP. A user can rent EIP addresses and allocate them to the instances that need persistent external addresses.

There are three basic types of storage services in AWS. Both the instance store and EBS can provide storage for instance volumes. S3 can store the snapshots of instance volumes.

An instance can have multiple volumes, including the volume that stores the file system and volumes that store user data. Volumes are typically stored on EBS, which provides reliable network storage by replicating the volumes within the availability zone. Volumes can be attached to any instance in the same zone. Volumes can persist independently

from the instances---they can be created and deleted independently from the lifecycles of the instances.

The instance store provides temporary storage for volumes. It is local to the instances, therefore providing better performance than EBS-backed volumes. But instance store backed volumes persist only when their associated instances are running; they are automatically deleted when their instances are stopped or terminated.

A user can back up an EBS volume by taking a snapshot of the volume, which is a point-in-time copy of the data on the volume. Snapshots are stored on S3 and replicated across availability zones for high reliability. Later the user can recover EBS volumes or create new volumes from the snapshots.

Unlike EBS and instance store which provide only volume storage, S3 is a general-purpose storage service which can store all kinds of data. It provides an object store interface, which organizes objects into buckets. Objects can be of any formats and any sizes, and they are all accessed through the same Get and Put interface. S3 is also highly reliable and cost-effective. But its performance is inferior to EBS and instance store, and cannot be used to provide volume storage to running instances.

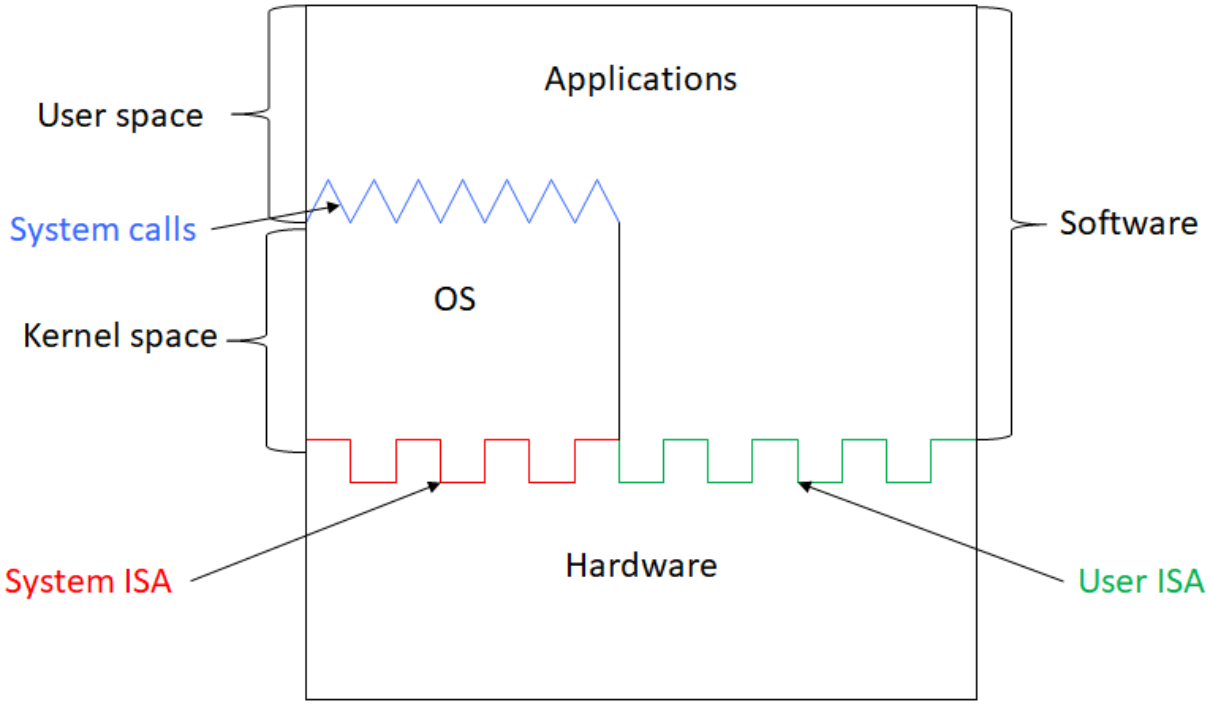
SQS provides reliable and scalable message queuing service. Queues are a fundamental data structure for messaging. Senders deposit messages in a queue, and receivers retrieve the message from the queue. SQS is highly reliable: it employs replication to ensure that every message is always delivered; but occasionally, duplicates of the same message may be delivered. SQS is also highly scalable: it delivers the same performance to message delivery regardless of the number of queued messages and the number of concurrent senders and receivers working on the same queue.

To use SQS service, the user first creates an SQS queue; then the senders can send messages to the queue and the receivers can receive messages from the queue. To ensure the delivery of a message, the message stays in the queue until the receiver explicitly deletes the message when it determines it is safe to do so. To avoid multiple receivers getting the same message, SQS provides a visibility timeout clock on the queue. When a receiver retrieves a message from the queue, the timeout period for this message starts. During this period, this message is not visible to the other receivers, and therefore cannot be retrieved. The receiver that has retrieved the message needs to delete the message after it has successfully processed it.

□

Virtual Machines

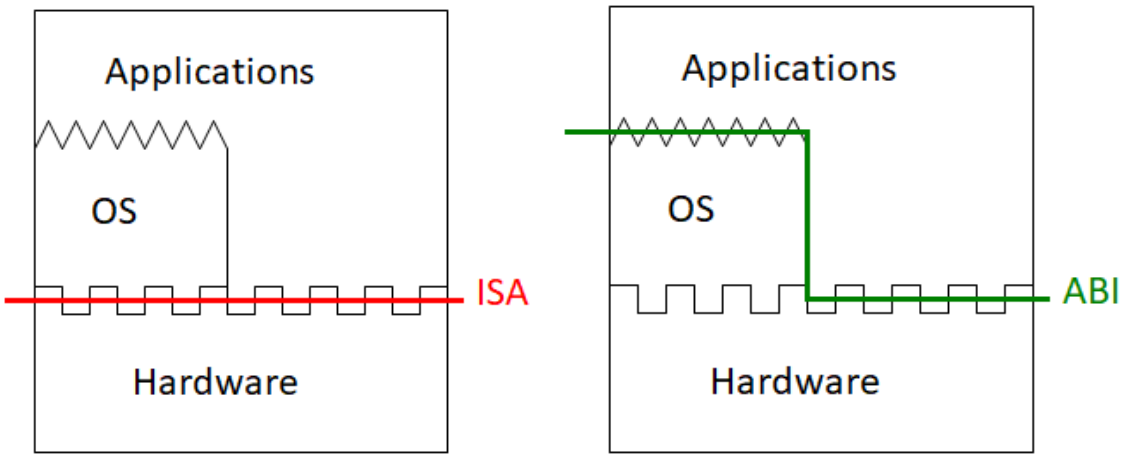
Before we discuss virtual machines, let's review the key system interfaces.



Instruction set architecture (ISA) is the interface that hardware, including the CPUs, memory, and I/O devices, provides to the entire software stack, including the applications, libraries/runtimes, and OS. It is what we see from the viewpoint of the whole software environment. It is the interface that the entire software stack uses.

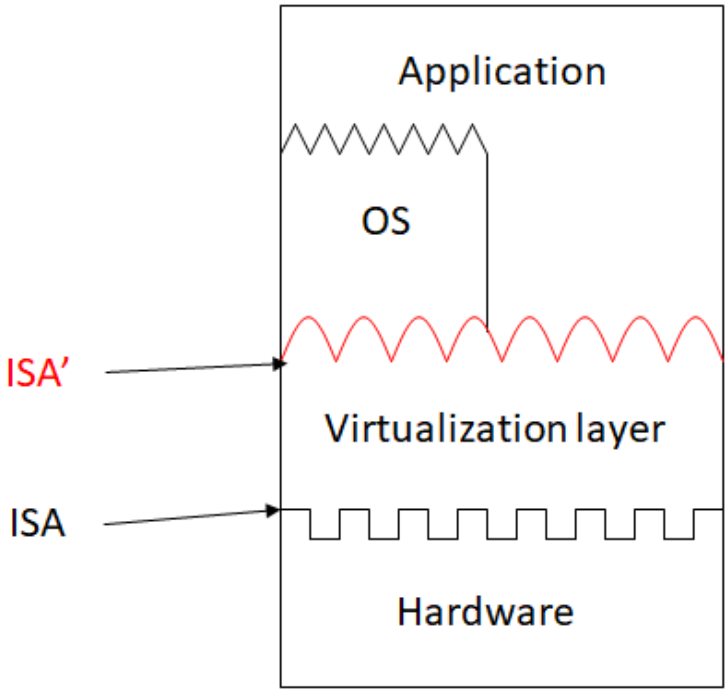
The ISA consists of two parts, with different levels of privileges. System ISA includes privileged instructions such as I/O instructions (in and out). It is available only to the kernel-space software (OS) which runs in the kernel model of the hardware. User ISA includes unprivileged instructions such as arithmetic instructions (e.g., add, sub), branch instructions (e.g., bz, bnz), and memory access instructions (load and store). It is available to both kernel-space software which runs in the kernel model of the hardware and user-space software (applications) which runs in the user mode of the hardware.

Application binary interface (ABI) is the interface that the hardware and kernel-space software (OS) provide to the user-space software (applications). It includes the user ISA, which allows applications to directly perform unprivileged operations on the hardware, and the system call interface, which allows applications to request privileged operations through OS. ABI is the interface that the applications use.

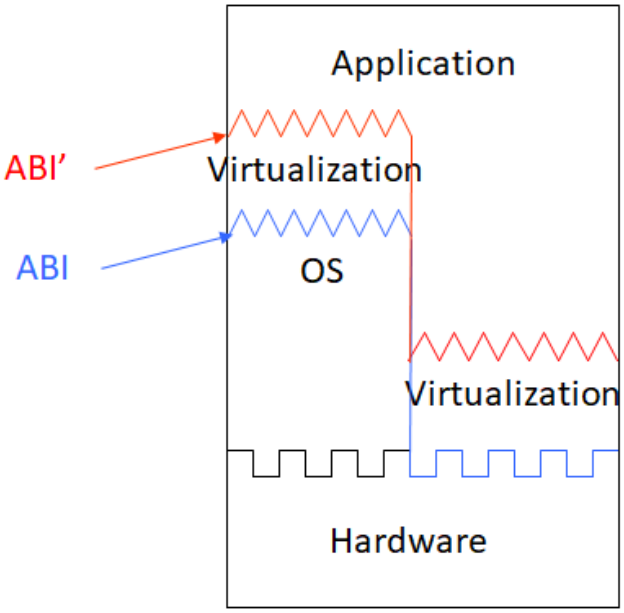


Based on ISA and ABI, we have two major types of VMs.

ISA virtualization is a layer of indirection inserted in between the software stack and the hardware. It virtualizes the underlying physical ISA and presents a virtual ISA to the software stack in the VMs. ISA VMs are the kind of VMs that we are familiar with, including desktop VMs such as VMware Workstation and VirtualBox, and server VMs such as VMware ESX and Xen.



ABI virtualization is a layer of indirection inserted in between the applications and the OS and hardware. It virtualizes the underlying physical ABI and presents a virtual ABI to the applications in the VMs. ABI VMs are also widely used such as Java VMs (JVM). With JVM, the bytecode virtualizes the underlying user ISA, and Java core classes virtualize the underlying system calls. Together, the virtual ABI provided by JVMs allow Java applications to run everywhere despite the differences in the underlying OS and hardware.

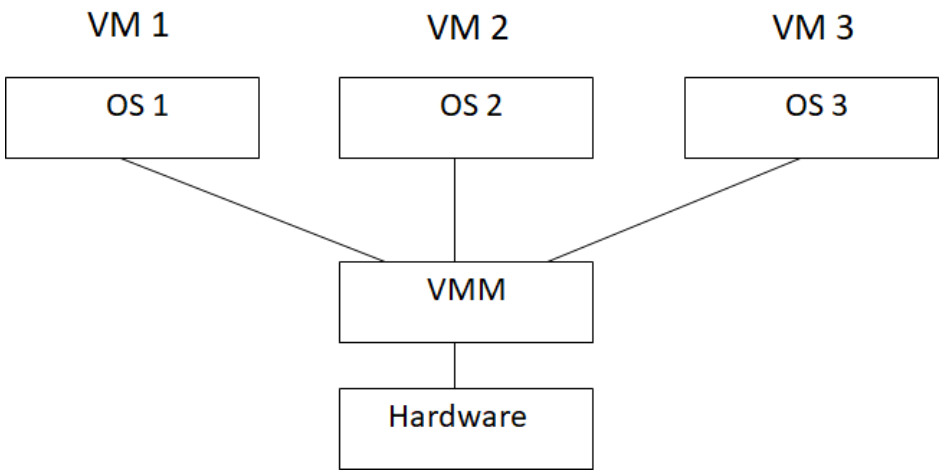


The virtual ISA (ISA') can be the same as or different from the physical ISA. Emulators such as Android emulators and game emulators provide an ISA' for a hardware architecture (ARM, game consoles) that is entirely different from the underlying hardware, and allow applications (Android apps, games) developed for ISA' to run on the hardware that provides a different ISA. However, because every instruction that the software issues using ISA' has to be emulated by the emulator using instructions from ISA, the speed of emulated applications is typically poor.

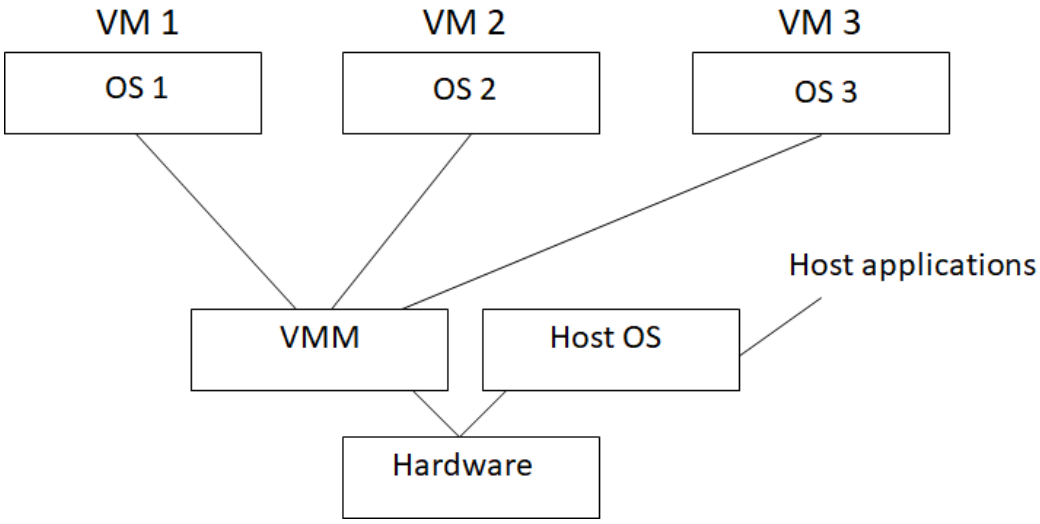
In contrast, system VMs such as VMware, Xen, and VirtualBox VMs provide a virtual ISA (ISA') that is identical to the underlying ISA. This allows the majority of the instructions, i.e., the instructions that belong to the user ISA, from the VMs to run directly on the underlying hardware without requiring any emulation. The privileged instructions from the VMs still need to be emulated, but they typically account for only a small portion of all the instructions executed by the software in the VMs. System VMs are the enabling technology of cloud computing, and by providing an ISA that is identical to the

underlying hardware, they ensure that cloud applications can achieve good performance when running on the VMs. In the rest of this lecture, we will focus on system VMs.

Depending on whether there is a host OS on the system, there are two types of system VMs. With classic VMs, the virtualization software, which is called VM monitor (VMM) (a.k.a. hypervisor), is the only kernel-space software running in privileged mode and managing hardware resources at all times. These are called “classic” VMs because they follow the same architecture of the first VMs invented by IBM back in the 70s which we discussed in our history of cloud computing lecture. Modern examples of classic VMs include VMware ESX and Xen.



With hosted VMs, there is a host OS that runs alongside the VMM. The host OS and the VMM both run in kernel mode, and they share the management of the hardware resources. To differentiate from the OS that runs in a VM, we call the OS that runs directly on the VM host the host OS and the OS that runs inside a VM guest the guest OS. In this environment, the host OS supports the execution of the host applications, i..e, the applications that do not run in VMs, on the hardware, and the VMM supports the VMs on the hardware. Examples of hosted VMs include the desktop VMs that we are familiar with, such as VMware Workstation and Virtualbox.



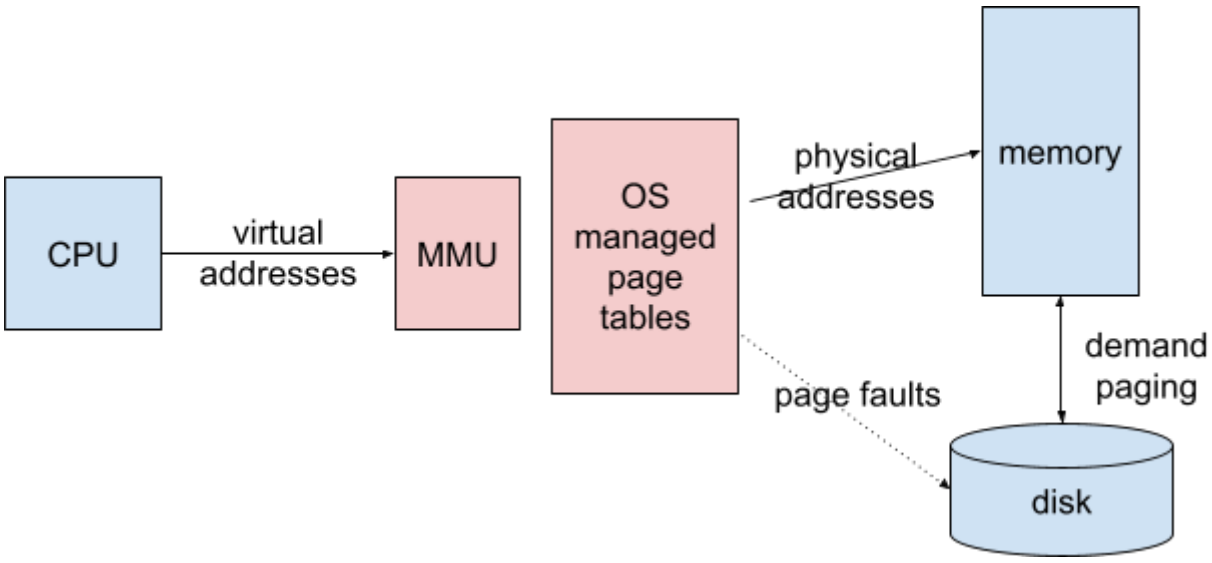
Now look at the details of system virtualization. We will study how VMM virtualizes the three major hardware subsystems, processors, memory, and I/O devices. On a non-virtualized system, the OS manages these three subsystems and in essence also virtualizes them for the processes sharing these resources. We will use what we know about OS to help us understand how VMM works.

Processor virtualization. In a non-virtualized system managed by the OS, most application instructions (the part that belongs to the user ISA) execute directly on the processor, without involving the OS, which ensures good performance of the applications. Applications' accesses to shared resources are handled indirectly (because they cannot use privileged instructions) by calling the OS via system calls, and the OS performs these accesses using instructions from the system ISA.

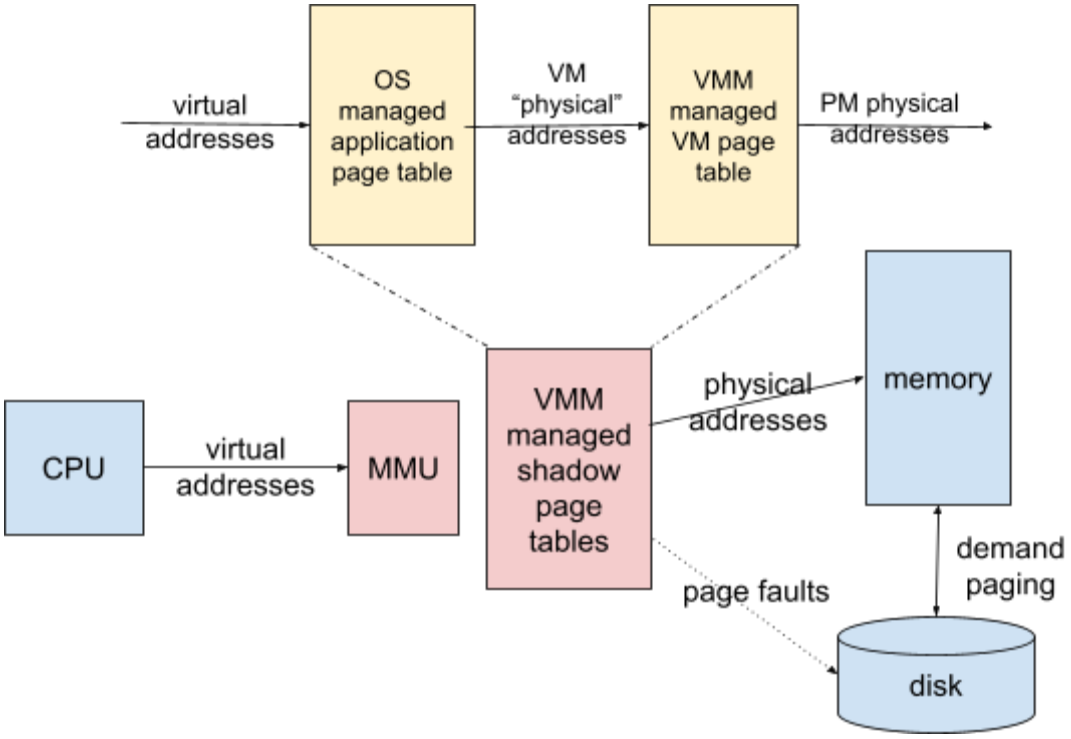
In a virtualized system managed by the VMM, most application instructions (the part that belongs to the user ISA) still execute directly on the processor, without involving the VMM, which ensures good performance of the applications. When applications need access to shared resources, they will still make system calls to the guest OS, but the OS cannot run privileged instructions either since it also runs in user mode (and only the VMM runs in kernel mode). When the virtualization is entirely transparent to the guests, the guest OS is unaware that it is running in a VM, and it cannot and will not call the VMM for help. Instead, it will do what it is supposed to do, using privileged instructions to perform access to shared resources. When privileged instructions are executed in user mode, they will be trapped by the hardware and handled by the system software which in this case is the VMM. This gives the VMM the opportunity to intercept the

guest's attempt to access shared resources and emulate it using privileged instructions that the VMM can use.

Memory virtualization. In a non-virtualized system managed by the OS, an application running on the CP loads and stores virtual addresses; the MMU maps the virtual addresses to physical addresses of the physical memory, using page tables managed by the OS. To support many concurrent applications on the limited physical memory, the OS uses the secondary storage (disks) to provide additional space: it swaps out pages that are currently not needed by the applications from the memory and temporarily stores them on disk. When the application makes reference to a swapped out page, the MMU cannot find its mapping from the page table (since it is currently not in the physical memory), which triggers a page fault. The OS handles the page fault by bringing the faulting page back to physical memory on demand, which is called demand paging.



In a virtualized system managed by the VMM, there is one more level of memory virtualization. An application still loads and stores virtual addresses; the guest OS maps virtual addresses to the "physical" addresses of the VM using the application's page tables; and the VMM maps the "physical" addresses of the VM to the physical addresses of the physical machine using the VM's page tables. However, traditional MMU can work with only one level of page table for address translation; it cannot use the two levels of page tables needed for translating a virtual address to a physical address. To solve this problem, the VMM maintains a shadow page table on the side for each application, which maps directly from the application's virtual addresses to the physical memory's addresses. The MMU then uses only this single-level page table for address translation.

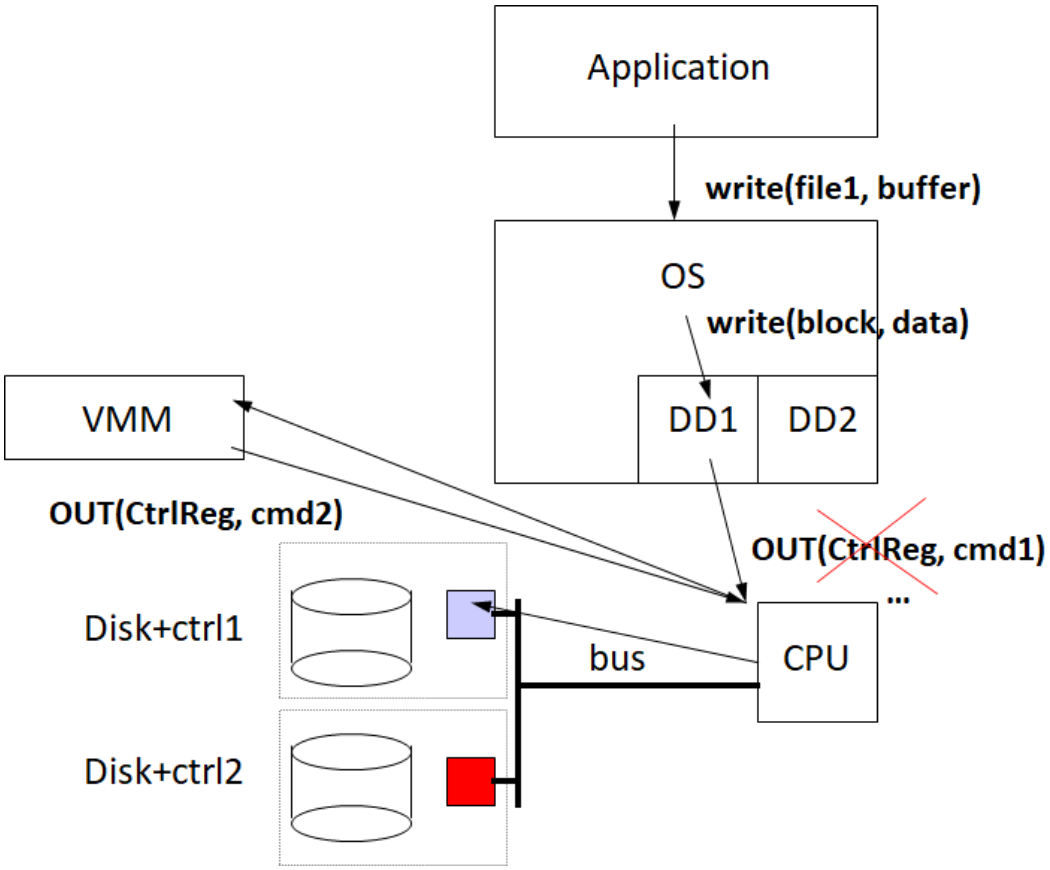


In a virtualized system, there are two levels of demand paging. A guest OS still does demand paging for its applications: it swaps out pages from the VM's "physical" memory to the VM's "disk" (which is often stored as a file on the host's file system). The VMM does the second demand paging for its VMs: it swaps out pages from the physical machine's physical memory to the physical disk.

When a page fault happens in a virtualized system, the VMM is the kernel-space software to handle it. The faulting page might have been swapped out by either guest OS or the VMM. If it was swapped out by the guest OS, the guest OS knows how to handle it and is expecting a page fault when the page was referenced. In this case, the VMM injects a virtual page fault into the VM, and the guest OS runs its page fault handler by bringing the page back from its VM's disk to its "physical" memory. If the faulting page was swapped out by the VMM, the VMM knows how to handle it, and the guest OS shouldn't be made aware since it expects the page to be present in its "physical" memory. The VMM brings the page back from the physical disk to the physical memory, masking the page fault and its handling both from the guest OS.

I/O virtualization. When it comes to the virtualization of I/O devices, there is a major difference between classic VMs and hosted VMs. With classic VMs, the VMM needs to implement all the device drivers to support the diverse I/O devices that the physical machine may have. With hosted VMs, the host OS already has all the device drivers, to

allow its applications to use the devices, and the VMM can exploit these drivers instead of implementing them on its own. Moreover, in both cases, the virtual I/O devices provided by the VMM to the VMs can be very much different from the underlying physical I/O devices, since the applications typically don't really care about what specific devices they are using.

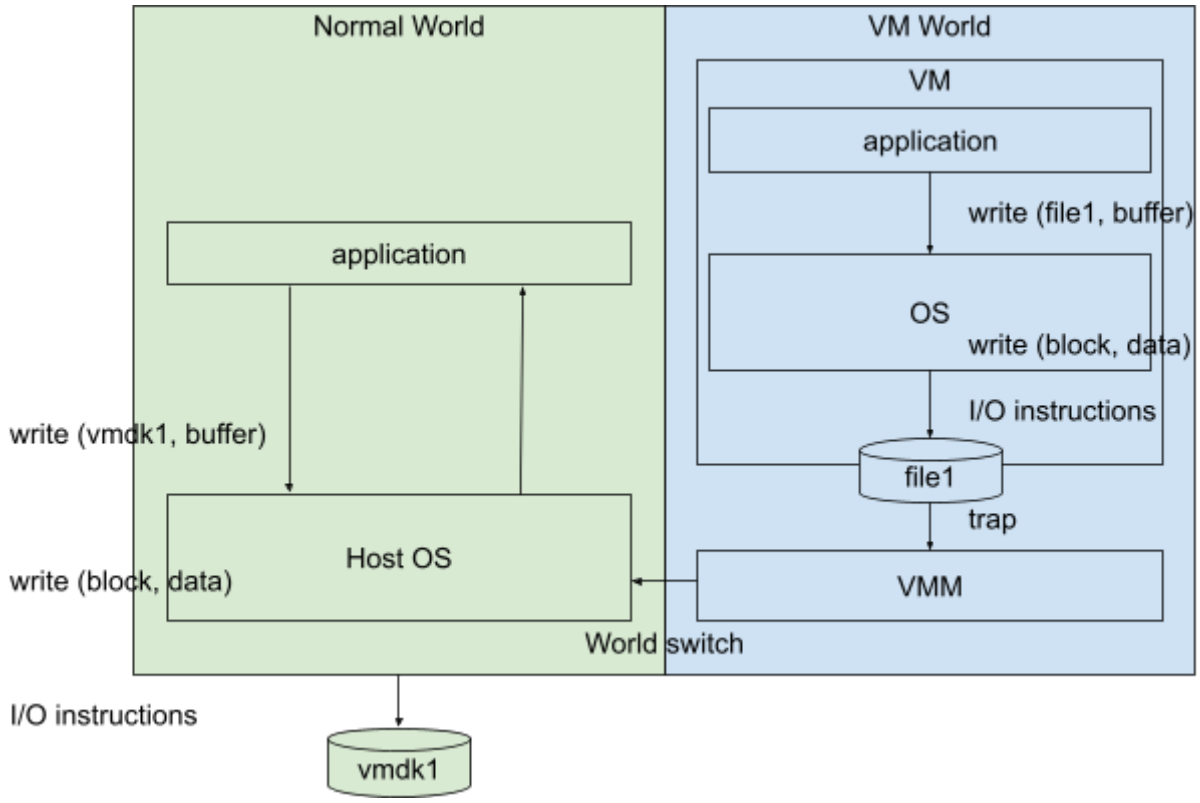


Let's first walk through the steps involved in I/O. The application's I/O request, such as writing to a file (stored on VM's disk), it is handled by the I/O stack in the VM, including API call to the libraries/runtimes, the system call to the guest OS, the file system request to the file system, the block request to the block device layer, and finally I/O instructions to the device. As we discussed in processor virtualization, when the guest OS uses these I/O instructions, which are privileged, they will be trapped into the VMM, which allows them to be intercepted and emulated.

On a classic VM system, the VMM will use its I/O stack to emulate the VM's I/O request. Assuming the VM's disk is stored as a file on the file system of the physical disk, the application's write request should be emulated by a write request to the VM disk file (and other relevant I/Os). The VMM will use its own file system implementation and

block device driver to perform this request, which eventually will generate a series of I/O instructions to the physical disk (more precisely, the disk controller). When the write completes, the disk controller generates an interrupt, which is handled by the VMM's interrupt handler. This result traverses back across all the VMM and guest layers we mentioned above, and eventually returns to the application which is waiting for its write to complete. Note that in order for guest OS to be notified of the write completion, a virtual interrupt needs to be injected into the VM, much like how a virtual page fault is injected as discussed earlier in memory virtualization.

On a hosted VM system, the VMM does not have the necessary I/O stack for handling the VM's I/O request. Instead, it passes it on to the host OS and leverages the host OS's I/O stack to emulate the I/O. This involves a switch from the VM world, managed by the VMM, to the normal world, managed by the host OS. Again, to simplify the implementation, the request is further passed on to a user-space application in the normal world, which implements the emulation of the request using the APIs and system calls available to the user-space applications. The host OS handles the system calls and performs the I/Os using its own I/O stack, including its device driver provided for the physical disk.



□

Cloud Computing

A Bottom-up Approach

First Edition

Ming Zhao, PhD

Table of Contents

Chapter 3

Infrastructure as a Service

Infrastructure as a Service (IaaS) is the most fundamental service model as it provides users access to the virtualized hardware, including processors, storage, and networking, and allows them to build an entire software stack that they desire, including operating systems, libraries and runtimes, and applications.

In this chapter, we will learn the key services that an IaaS provide offers and how to use these services to develop a cloud application. We will often use the IaaS services from Amazon Web Services (AWS) as examples. We choose AWS because it provides representative IaaS services that can be found in other cloud providers; it is also the first IaaS provider and today the largest cloud provider in the world.

There are three key types of IaaS services that we typically need to develop a cloud app: compute (AWS EC2), which provides computing resources for executing the applications, storage (AWS EBS, S3), which provides storage resources for storing the data that applications needs to access, and messaging (AWS SQS) for the distributed applications and their distributed components to communicate across the network.

3.1 Motivating Example

Before we discuss the IaaS services in detail, let us look at an interesting success story of IaaS. Animoto.com is a SaaS provider that provides video making as a service. Its customers can upload materials that they want to use, such as photos and video clips, and use the provided templates to customize and generate their videos. On April 18th 2008, Animoto.com launched a viral ad campaign on Facebook. As illustrated in Figure 3.1, this led to a huge spike in the demand of its service, resulting in its number of users growing from 5,000 to 750,000 in just three days. Fortunately, Animoto.com was hosted on AWS, and it was able to handle this sudden surge of demand by quickly growing its number of EC2 instances from 40 to 4000. Even though it is a really outdated story, it still serves as a good example of the power of cloud computing. Without the

cloud, a company like Animoto.com would not be able to deal with the sudden change of demand quickly and cost-effectively.

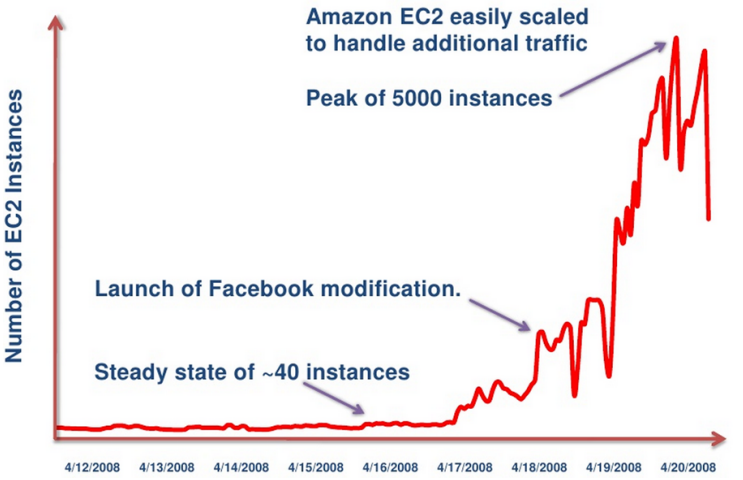


Figure 3.1: The success story of Animoto.com on EC2

In the following, we will learn how to use the basic IaaS resources. Although the discussions here are based on AWS, we can also find similar IaaS resources from the other providers.

3.2 Compute

Compute is the most basic service that a cloud needs to provide to allow its customers to run their applications and perform computations. With IaaS, compute resources are provided in the form of virtual machines (VMs). VMs serve two important purposes for an IaaS cloud. First, VMs are resource containers, each providing the necessary hardware resources, including CPUs, memory, and storage and network devices, for a customer to run her applications. At the same time, the amount of resources that an application can consume is also defined by the VM where it runs. For the sake of manageability, IaaS providers commonly offer a limited category of different types of VMs, called instance types in EC2. The various VM types differ in the amount of CPU, memory, and I/O resources. For example, t2.micro, the instance type that the AWS Free Tier provides, has 1 vCPU and 1 GB of memory; t2.large has 2 vCPUs and 8 GB of memory; Some VM types also differ in capabilities. For example, m5d.large has a local SSD to provide faster storage to the VM; p2.xlarge has a GPU for accelerating the VM's workload. To achieve vertical scaling, a user can change the type of VMs that her application runs on; to achieve horizontal scaling, she can change the number of instances of the same VM type that her

application uses.

The second important purpose of VMs in an IaaS cloud is to provide the execution environments, including OSes and libraries/runtimes, that the customers' applications need to run. A user can install and customize the entire software stack in a VM, which is then encapsulated by the VM image. With this image, the user can start VM instances of any type of her choice on any IaaS resource in the cloud. As discussed in Chapter 1, users can specify the location of their instances at some higher level of abstraction, e.g., Availability Zones (AZs) in AWS, but they have no control or even knowledge of the location at lower level of abstraction such as individual servers that are used to host the VMs.

VM instances are instantiated from VM images. The relationship between instances and images is analogous to the relationship between processes and programs. A program contains the code and data, and is often stored as a file on a file system. When the program is brought into memory and executed on CPU, it becomes a process which is an instance of the program in execution. Many processes can be instantiated from the same program, and each process can be given a different amount of resources to execute. A VM image contains an entire file system that stores the entire software stack, including the OS, system programs, libraries, and application programs needed for starting a machine. When these programs from the image are loaded into memory and executed on CPU of a host, they become an instance of the VM in execution. becomes an instance upon execution. Many VM instances can be launched from the same image, and each VM instance can be launched as a different instance type and therefore executed with a different amount of resources specified by the instance type.

Specifically in AWS, VM images for launching EC2 VM instances are managed as Amazon Machine Images (AMIs). AWS provides a catalog of different AMIs with preinstalled OSes such as Linux, Windows, and MacOS and even preinstalled software frameworks such as PyTorch for machine learning and Hadoop for data analytics. Users can also customize these template images or upload their own images as AMIs for launching EC2 instances that have their desired software environment.

During the lifecycle of an instance, it goes through various states as shown in the state diagram Figure 3.3. An instance is launched from an AMI and starts to run. A running instance can be rebooted, stopped, or terminated. The difference between the latter two cases is that a stopped instance's state still exists in the system and can be started again later, whereas a terminated instance is removed from the system. Both cases do not incur EC2 charges since the instance is not running on EC2 any more, but a stopped instance still incurs charges for storing the instance's state, which will become clearer in the following IaaS storage section.

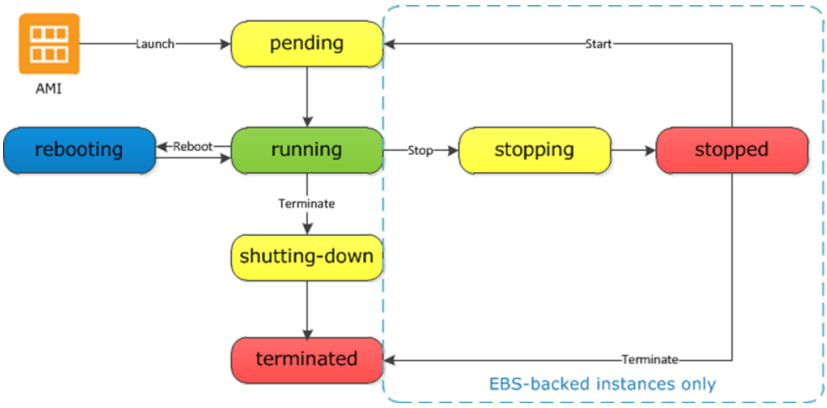


Figure 3.2: The lifecycle of EC2 instances

3.3 Storage

There are generally three types of storage resources in an IaaS cloud to support VMs, including local volume storage, remote volume storage, and general-purpose storage. Each of them has different characteristics and can be used in different ways to support IaaS VMs and applications. We need to understand these storage resources and use them properly.

An IaaS user manages both the VMs and the applications running on the VMs, and the user needs storage for two types of data: VM data and application data. The former is what the VMs needs to run, and the latter is the applications need to run, including input and output.

Let us first understand the concept of disk volumes which is the basic abstraction for VM storage. Just like a physical machine can have multiple disks, a VM has multiple virtual disks. Each disk is typically installed with a file system to allow the applications to store and organize data using folders and files. Such disks are often called volumes. Among all the volumes that are attached to a machine, one of them has to provide the OS, which is called the root volume, so the machine can load the OS and run it. The other volumes can store applications. The VM images discussed in the previous section are copies of such VM volumes that can be saved and used later to launch VM instances. For example, when a user creates an EC2 instance from an AMI image, the chosen image is copied to a volume and the volume is attached to the instance so the instance can start and run. In the rest of this section, we will discuss the three different types of storage resources for an IaaS application in detail.

Volume Store

On a personal computer, VM volumes are stored on local disks, often as files on a local file system. In a cloud, storage resources are consolidated into storage

servers and shared by all the VM hosts via storage virtualization. In this setup, a VM instance running on a VM host accesses its VM volumes *remotely* stored on the storage servers over the network. These shared volume storage resources are offered as a cloud service to users to store their volumes and provide them to their VM instances. In AWS, this service is the Elastic Block Store service. It is called block store because data is stored and accessed as fixed-size blocks.

A volume store provides VM volumes accessible by VM instances anywhere in the same zone, because the storage resources are shared by all the VM hosts. Volumes can be attached to any instance in the same zone. An instance can simultaneously have multiple volumes attached to it. But the volumes stored in the volume store of one zone are not available to the instances in a different zone. The volume store also provides high reliability by replicating the volumes across the storage servers in the same zone. Volumes can persist independently from the instances—a volume can be created without being attached to any instance, and a volume can be saved after the instance that it was previously attached to is terminated.

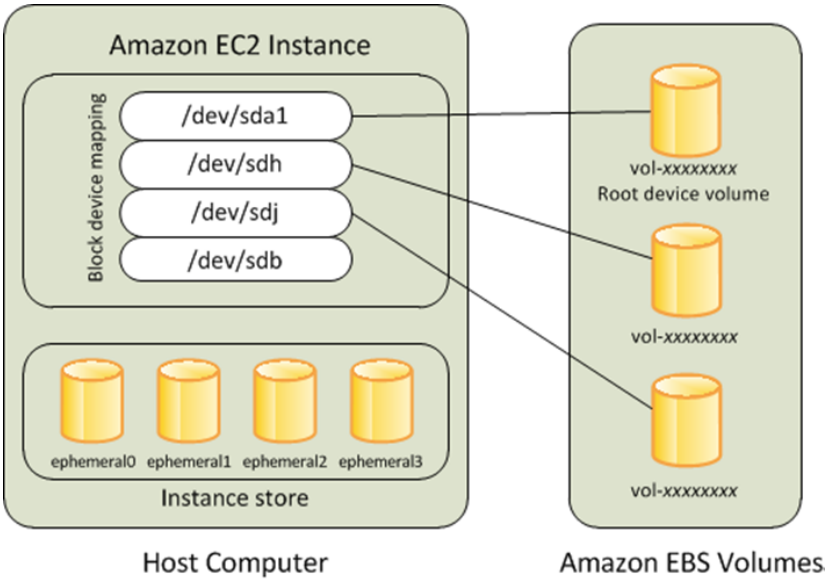


Figure 3.3: EBS and Instance Store provide volume storage to EC2 VM instances

Local Volume Store

The VM hosts in an IaaS cloud can also provide *local* volume storage to the instances that they are hosting. Volumes stored on such a local volume store typical provides better performance to the instances than the volumes accessed

over the network from a *remote* volume store discussed above. But local volume store cannot be offered as a standalone service as the *remote* volume store because it is tied to the specific host and it is available to only the instances that are running on the same host. Therefore, local volume store is typically provided as a feature to high-performance instance types.

When such an instance starts, it will be given local volumes to provide high-performance storage. The lifecycle of local volumes are tied to the instance that they are attached to—they are deleted when the instance is stopped or terminated. This is because no VM host can be dedicated to any VM; every time an instance is launched, it is run on a host that is likely to be different from the host where a previous instance of the same VM ran, and therefore any local volumes on that previous host will not be local any more to the new host and have no reason to persist any more when the previous instance was stopped or terminated.

General-purpose Object Store

A general-purpose object store can store any kind of data, including both structured and unstructured data, data of any format, and data of any size. In AWS, this service is the Simple Storage Service (S3). Such a general-purpose storage service is often offered as an object store because of the generalizability and scalability of object storage. Object storage stores data as self-contained objects, where each object stores both data and metadata and has a unique identifier for addressing the object. The self-contained nature allows objects to store any kind of data with any type of attributes. Object storage also eliminates the need of any complex structure such as a directory tree in a file system for naming the objects. Instead, a general-purpose cloud object store allows users to organize data into buckets, where each bucket serves as a container of arbitrary objects and each bucket has a globally unique address. The object store also provides a simple interface, mainly just "get" and "put" operations, for accessing the objects, which is much simpler than file systems. These designs all help make the object store scalable.

Compared to the other storage services that an IaaS cloud offers, this general-purpose object store has several advantages. First, it is highly reliable. Volume store is also reliable, but it replicates data only within an AZ, and thus cannot tolerate failures occurred at the AZ level. In comparison, the object store replicates data across the AZs in the same region, and can thus tolerate failures occurred at a larger scale. Second, the object store is typically cheaper per unit of data storage, and is more suited for long-term storage of large datasets. Finally, because the object store data is replicated across a cloud region, it is available to all the AZs in the region; in comparison, local volume store is available to the host that it resides, and remote volume store is available to only the AZ that it resides.

This general-purpose object store provides important storage services to IaaS VMs. Specifically, it is commonly used to store the snapshots of VM

volumes. A volume snapshot is a point-in-time copy of the data on the volume. Just like a photo snapshot captures the state of a scenery at the time when the snapshot was taken, a volume snapshot captures the state of a volume at the time when the snapshot was taken. Volume snapshots have several important uses. First, they provide backups of volumes, and can be used to recover the volumes in case of failures. Second, a volume can have a series of snapshots taken at different times, which allow the volume to be rolled back to any previous state captured by the snapshots. For example, if an update made to the OS on a volume becomes undesirable and there is no clean way to undo the update, the user can simply roll back the volume to the snapshot taken right before the update and continue to use the volume from there. Finally, volume snapshots provide an efficient way to clone volumes and use them to launch multiple instances each with a dedicated volume clone.

However, different from the local and remote volume stores, VM instances cannot be directly launched from the volume snapshots. The volume snapshots have to be loaded into a volume store first before they can be used to launch VM instances. Performance is the main reason for this restriction. On one hand, the object store is much slower and farther away from the VM hosts than the volume stores; on the other hand, the object store's interface does not allow the objects to be partially modified, which is inefficient for instance executions.

Bibliography