Chapter 2
Cloud Basics

One of the appealing aspects of cloud computing is that it hides the complexity of IT technology from users and developers. No need to know details of how a service is generated – it is the service provider's job to provide a corresponding abstraction layer. This chapter contains an overview of some technologies on which cloud computing depends: virtualization, service-oriented architectures (SOA), and Web services.

2.1 Virtualization

Resource virtualization is at the heart of most cloud architectures. The concept of virtualization allows an abstract, logical view on the physical resources and includes servers, data stores, networks, and software. The basic idea is to pool physical resources and manage them as a whole. Individual requests can then be served as required from these resource pools. For example, it is possible to dynamically generate a certain platform for a specific application at the very moment when it is needed. Instead of a real machine, a *virtual machine* is used.

2.1.1 Benefits and Drawbacks of Virtualization

For a provider of IT services, the use of virtualization techniques has a number of advantages [4]:

- **Resource usage:** Physical servers rarely work to capacity because their operators usually allow for sufficient computing resources to cover peak usage. If virtual machines are used, any load requirement can be satisfied from the resource pool. In case the demand increases, it is possible to delay or even avoid the purchase of new capacities.

- **Management:** It is possible to automate resource pool management. Virtual machines can be created and configured automatically as required.
- **Consolidation:** Different application classes can be consolidated to run on a smaller number of physical components. Besides server or storage consolidation, it is also possible to include entire system landscapes, data and databases, networks, and desktops. Consolidation leads to increased efficiency and thus to cost reduction.
- **Energy consumption:** Supplying large data centers with electric power has become increasingly difficult, and, seen over its lifetime, the cost of energy required to operate a server is higher than its purchase price. Consolidation reduces the number of physical components. This, in turn, reduces the expenses for energy supply.
- **Less space required:** Each and every square yard of data center space is scarce and expensive. With consolidation, the same performance can be obtained on a smaller footprint and the costly expansion of an existing data center might possibly be avoided.
- **Emergency planning:** It is possible to move virtual machines from one resource pool to another. This ensures better availability of the services and makes it easier to comply with service level agreements. Hardware maintenance windows are inherently no longer required.

Since the providers of cloud services tend to build very large resource centers (*IT factories*), virtualization not only leads to a size advantage, but also to a more favorable cost situation. This results in the following benefits for the customer:

- **Dynamic behavior:** Any request can be satisfied just in time and without any delays. In case of bottlenecks, a virtual machine can draw on additional resources (such as storage space, I/O capabilities).
- **Availability:** Services are highly available and can be used day and night without stop. In the event of technology upgrades, it is possible to hot-migrate applications because virtual machines can easily be moved to an up-to-date system.
- **Access:** The virtualization layer isolates each virtual machine from the others and from the physical infrastructure. This way, virtual systems feature multi-tenant capabilities and, using a roles concept, it is possible to safely delegate management functionality to the customer. Customers can purchase IT capabilities from a self-service portal *(customer emancipation)*.

A drawback of virtualization is the fact that the operation of the abstraction layer itself requires resources. Modern virtualization techniques, however, are so sophisticated that this overhead is not too significant: Due to the particularly effective interaction of current multicore systems with virtualization technology, this performance loss plays only a minor role in today's systems. In view of possible savings and the quality benefits perceived by the customers, the use of virtualization pays off in nearly all cases.

Another problem is that after the consolidation, more systems need to be operated and managed: Besides the virtual machines, there is the physical infrastructure. By managing the resources using sophisticated management tools, the total balance is nevertheless positive because much less staff is required in practice.

2.1.2 Virtualization Concepts

Virtualization stands for a variety of different concepts and technologies, which differ by their implementation, relevance to practical use, and frequency of use.

2.1.2.1 Operating System Virtualization

The use of operating system virtualization or partitioning (such as IBM LPARs) in cloud environments may help to solve security and confidentiality problems, which would otherwise impair the acceptance of the cloud approach.

For this type of virtualization, which is also called 'container' or 'jails', the host operating system plays a major role. This is a concept where multiple identical system environments or runtime environments, which are completely isolated from each other, run under one operating system kernel (see Fig. 2.1). Seen from the outside, virtual environments appear as autonomous systems. All running applications use the same kernel, but they can only see the processes belonging to the same virtual environment.

Mainly Internet service providers (ISPs), who offer (virtual) root servers, prefer this kind of virtualization because it is associated with a minor performance loss and a high degree of security. The drawback of operating system virtualization is its reduced flexibility: While multiple independent instances of the same operating system can be used simultaneously, it is not possible to run different operating systems at the same time. Popular examples of operating system virtualization are the container technology from Sun Solaris, OpenVZ for Linux, Linux-VServer, FreeBSD Jails, and Virtuozzo.
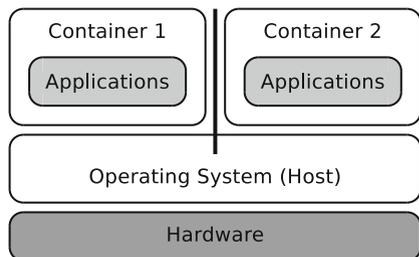

Fig. 2.1 The concept of operating system virtualization (container)

2.1.2.2   Platform Virtualization

Platform virtualization allows to run any desired operating systems and applications in virtual environments. There are two different models: Full virtualization and paravirtualization. Both solutions are implemented on the basis of a virtual machine monitor or hypervisor.

The hypervisor is a minimalistic meta-operating system used for distributing the hardware resources among the guest systems and for access coordination. A type-1 hypervisor is built directly on top of the hardware, a type-2 hypervisor runs under a traditional basic operating system (see Fig. 2.2).

Full virtualization is based on the simulation of an entire virtual computer with virtual resources, such as CPU, RAM, drives, network adapters, etc. including its own BIOS. Since the access to the most important resources, such as the processor and the RAM, is passed through, the processing speed of the guest operating systems nearly equals the speed to be expected if there was no virtualization. Other components, e.g. drives or network adapters, are emulated. While this decreases the performance, it allows to run unmodified guest operating systems.

Paravirtualization does not provide an emulated hardware layer to the guest operating systems, but only an application interface. For this purpose, the guest operating systems need to be modified because any direct access to hardware must be replaced by the corresponding hypervisor interface call. This is also referred to as hyper calls (just like system calls), which are used by the applications to call functions in the operating system kernel. Since this approach allows the guest system to participate actively in the virtualization (at least to some extent), a higher throughput than with full virtualization can be obtained, especially for I/O-intensive applications. Examples of full virtualization are the VMware products [135] or, specifically for Linux, the *Kernel-based Virtual Machine* (KVM). Under Linux, mostly Xen-based solutions are used for paravirtualization [14]. They play an important role, particularly in the realization of the Amazon Web Services [50].

2.1.2.3   Storage Virtualization

Cloud systems should also offer dynamically scalable storage space as a service. In this context, storage virtualization boasts a number of advantages. The fundamental
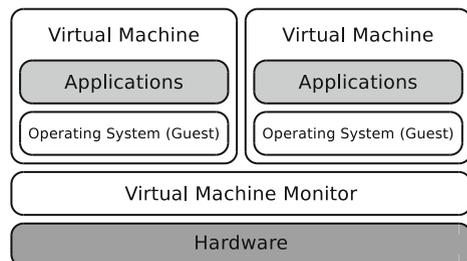

Fig. 2.2 Type-1 hypervisor (virtual machine monitor)

idea of storage virtualization is to separate the data store from the classical file servers and to pool the physical storage systems. Applications use these pools to dynamically meet their storage requirements. For the data transfers, a special storage area network (SAN) or a local company network (LAN) is used. Data for cloud offerings is mostly available in the form of Web objects that can be retrieved or manipulated over the Internet. An additional abstract administration layer is interposed between the clients and the storage landscape so that the representation of a datum is decoupled from its physical storage. This has a variety of advantages with respect to data management and access scalability. Central management also allows to operate the distributed storage systems at a lower cost.

Moreover, different categories of data storage can be organized in storage hierarchies (tier concept). This makes it possible to implement an automated lifecycle management for data sets, from tier-0 with the most stringent availability and bandwidth requirements to lower and cheaper tier levels with a correspondingly lower quality of service. The data can be migrated between these levels without affecting the service. By using snapshots, even large data quantities can be backed up without a special backup window. A further advantage of storage virtualization is that distributed mirrors may be created and managed in order to avoid service disruptions in case of malfunctions. Amazon, for instance, creates up to three copies in different data centers when storing data.

2.1.2.4 Network Virtualization

Techniques such as load balancing are essential in cloud environments because it must be possible to dynamically scale the services offered. The resources are usually implemented as Web objects. For this reason, it is recommended to apply the procedures commonly used for Web servers: Services can be accessed via virtual IP addresses. Through cluster technology, they realize load balancing as well as automatic failover in case of a failure. By forwarding DNS requests, it is also possible to integrate cloud resources into the customer's Internet namespace.

Network virtualization is also used for virtual local networks (VLANs) and virtual switches. In this case, cloud resources appear directly in the customer's network. Internal resources can thus be replaced transparently by external resources. VLAN technology has the following advantages:

- **Transparency:** Distributed devices can be pooled together in a single logical network. VLANs are very helpful when designing the IT infrastructure for geographically disparate locations.
- **Security:** Certain systems which require particular protection can be hidden in a separate virtual network.

On the other hand, VLANs involve more overhead for network administration and for programming active network components (switches, etc.).

2.1.2.5   Application Virtualization

Application virtualization is a software sales model where centrally managed applications are offered to the customers over a network. The advantages of application virtualization compared to traditional software installations are:

- Easier administration
- Automatic management of updates and patches
- Compatibility: all users work with the same software portfolio
- Global availability

  There are two different methods for deploying virtual applications:

- **Hosted application:** The application is available on the Internet and is transmitted to the client, e.g. using a streaming protocol.
- **Virtual appliance:** The application can be downloaded and used on the customer's own computer.

  In this case, a virtual environment provides all application files and components required by the program for it's execution. The virtual environment acts as a buffer between the application and the operating system, preventing conflicts with other applications or operating system components. In cloud environments, application virtualization is an important foundation of the SaaS concept (Software as a Service, see below) which is used for the dynamic provision of software components.

2.2   Service-Oriented Architectures

Besides virtualization, service-oriented architectures and Web services are to be considered as the fundamental prerequisites for cloud computing. Service-oriented architectures (SOA) are architectures whose components are implemented as independent services. They can be flexibly tied together and orchestrated and they can communicate via messages in a loosely coupled configuration. With cloud computing, virtualized IT infrastructures, platforms, and entire applications are implemented as services and made available for consumption in service-oriented architectures.

In the case of *public clouds* (see Chap. 3), services are offered on the Internet based on standardized Web protocols and interfaces. In this context, *Web services* and *RESTful services* have proven to be useful. For an SOA, however, the Web services technology is not mandatory.

2.2.1   The Properties of SOA

SOA is a style of software architecture, which defines how services are offered and used. These services may not only be used by customers, but can also be consumed

by other services and applications. Often, services are orchestrated here as business processes, which means that a customer specifies a certain order for performing calls and data exchanges with various services. The process itself can be provisioned as a service. Thus, the promise of SOA is to promote IT architectures to the business process abstraction layer, or, conversely, to enable a uniform approach for the description and realization of business processes using IT services.

Typical properties of an SOA are:

- It consists of distributed components, i.e. the services.
- Heterogeneous service consumers and service providers are interoperable across platforms; different programming languages and platforms can be used to implement individual services.
- Services are loosely coupled and will be bound dynamically at runtime. An SOA consequently allows dynamic adjustments, which have a local (but no system-wide) effect.

A short and concise definition of service-oriented architectures can be found in [13]:

> An SOA is a system architecture that represents varied, different, and possibly incompatible methods or applications as re-usable and openly accessible services and thus allows to use and re-use them in a platform- or language-independent manner.

Figure 2.3 illustrates the basic, theoretical interaction between service provider, service consumer, and service directory for the dynamic binding of services at runtime. A service consumer can locate a suitable service in a service directory or learn of its existence by using a broker. If a suitable service has been found or brokered, the service consumer receives a reference (address, endpoint) for accessing the service, i.e. exchanging messages with it. Then, the service can be called, i.e. a message can be sent. The service provider replies by sending a message back.

In practice, service endpoints are often directly communicated as part of the message. Attempts to set standards for service directories, such as Universal
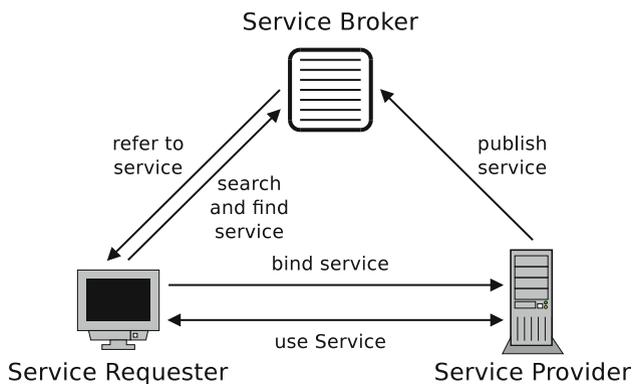

Fig. 2.3 SOA participants and actions

Description, Discovery and Integration (UDDI), were not successful. By sending endpoints directly in a message, it is possible to broker and bind services at runtime in a very dynamic, target-oriented way.

2.2.2 How Is an SOA Implemented?

An SOA may be implemented in many ways. The actual implementation depends primarily on the decisions taken with respect to communication and integration (coupling) between service provider and service consumer. Common approaches are above all Web services based on the Web Service Description Language (WSDL) and SOAP as well as RESTful services.

Services can be integrated into an SOA either by using point-to-point-connections or the hub-and-spoke approach. Each point-to-point-connection handles the connection between a service provider and a service consumer on an individual basis. For this purpose, the service consumer needs to know the endpoint of the desired service (IP address, URL). The service request is then addressed directly to the appropriate service provider.

With the hub-and-spoke approach, a broker acts as an intermediary between service providers and service consumers. The service consumer does not know the exact service provider address. For each service, a symbolic name exists, such as a URI. In the SOA context, the broker is called Enterprise Service Bus (ESB). Its tasks include routing, i.e. controlled, reliable forwarding of messages between the services across different systems and independently of the protocols in use. Another task is the transformation of data from one format to another. In its simplest case, this could refer to upgrading or downgrading different data types between 32-bit and 64-bit platforms, but also to the conversion between differing XML standards. Further tasks are the administration of a service directory and, depending on the implementation, the orchestration of messages.

Figure 2.4 contrasts point-to-point connections with the hub-and-spoke approach using an ESB.
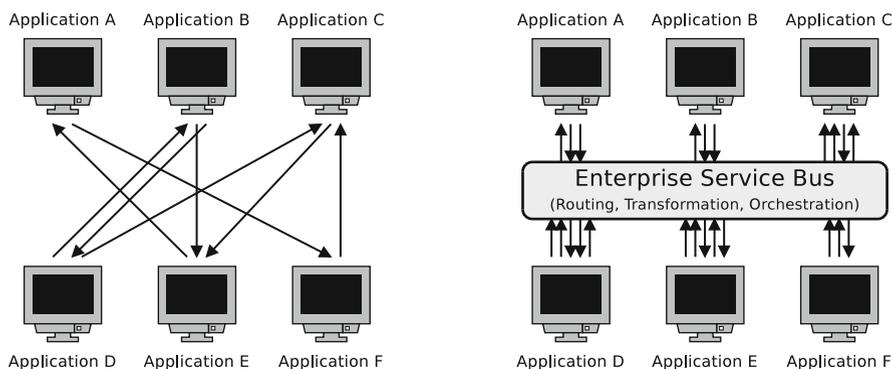

Fig. 2.4 Point-to-point-connections vs. Enterprise Service Bus

2.3   Web Services

Unlike distributed systems, which are interconnected over local networks, distributed cloud computing systems integrate heterogeneous resources which, in theory, could be located at any place on this earth where an Internet connection is available. The disadvantage of Internet connections compared to local networks is that they automatically entail problems, such as slow response times, low data rates, and potentially unreliable connections. In such environments, it is recommended to use a loosely coupled, asynchronous, message-based communication via Web services.

Just like with service-oriented architectures, a large number of different definitions for Web services exist. The Web Services Architecture Working Group of the W3C defines Web services as follows [137]:

> A Web service is a software application identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via internet-based protocols.

Wohlstadter and Tai [32] define Web services as distributed middleware, which enables machine-to-machine communication on the basis of Web protocols. Web services are defined not so much by the technology they use, but rather by their intended use. They propagate a compositional approach for application development. Functions can be integrated into an application using external, distributed services; and it is also possible to address legacy systems via Web services.

2.3.1   Interoperability

Web services describe the standards required to format and process messages as well as the standards for service interfaces. Two popular approaches are SOAP/WSDL-based Web services and RESTful (REpresentational State Transfer) services. SOAP is a messaging protocol and WSDL (Web Services Description Language) is an interface description language. Thus, SOAP/WSDL-based services have programmatic interfaces. REST, on the other hand, describes a style of software architecture, which is built on top of HTTP. RESTful services can only be invoked from the uniform HTTP interface. Both approaches use uniform resource identifiers (URIs) to identify the required services.

In their basic form, Web services describe only the primitives required to exchange documents (data) between service consumers and service providers. There are standards for transactional, reliable, and secure services for SOAP/WSDL. The Web Services Platform Architecture (WS-*) describes a set of extensions, which can be composed in a modular way; each of these extensions addresses a desired quality-of-service property.

The most common data exchange format is XML. XML is mandatory for SOAP/ WSDL and commonly used with RESTful services. The conversion of XML data structures to programming language-specific data structures is done using standardized mapping mechanisms. An alternative data exchange format is JSON (JavaScript Object Notation), which has become quite popular, in particular when RESTful services are consumed directly in the Web browser by JavaScript.

The Web service interface for SOAP/WSDL is described in WSDL, an extension of the XML schema specification. It describes Web services both abstractly in terms of types, messages, operations and port types, and concretely in terms of transport protocol-specific, available addresses (endpoints) to invoke the service. REST uses the generic HTTP methods (i.e. GET, PUT, POST, DELETE, etc.) rather than specific operations. Thus, REST is derived directly from the principles of the WWW and also uses its advantages, such as simple error handling, through standardized HTTP error codes.

2.3.2 SOAP Versus REST

SOAP is a messaging standard, which defines an XML-based message format, specifies processing rules for messages, describes conventions, and allows mappings to different Internet transport protocols (including HTTP).

SOAP messages are always XML documents consisting of three parts: A virtual envelope, the *SOAP envelope*, contains two elements: the optional *SOAP header* that may contain information such as routing and security details (authentication and authorization), and the mandatory *SOAP body*. The body element accommodates the actual message. It may include information on the data exchange or instructions for a remote procedure call.

SOAP is based on the *chain of responsibility* pattern; it is possible to define a series of distributed steps to process a SOAP message. This results in interesting options for Web service intermediaries or for the integration of specialized middleware functionality to support different qualities of service as addressed by the Web Services Platform Architecture.

REST, however, uses the HTTP semantics and thus prescribes a stateless communication (i.e. the server does not maintain any client state information). This is why REST features point-to-point connections where any necessary infor- mation is encoded in the message itself. This facilitates, e.g. the interposition of caches or the replication of servers.

Springer