Chapter 7

Cloud Infrastructure Mechanisms

Cloud infrastructure mechanisms are foundational building blocks of cloud environments that establish primary artifacts to form the basis of fundamental cloud technology architecture.

The following cloud infrastructure mechanisms are described in this chapter:

- Logical Network Perimeter
- Virtual Server
- Cloud Storage Device
- Cloud Usage Monitor
- Resource Replication
- Ready-Made Environment

Not all of these mechanisms are necessarily broad-reaching, nor does each establish an individual architectural layer. Instead, they should be viewed as core components that are common to cloud platforms.

7.1 Logical Network Perimeter

Defined as the isolation of a network environment from the rest of a communications network, the *logical network perimeter* establishes a virtual network boundary that can encompass and isolate a group of related cloud-based IT resources that may be physically distributed (Figure 7.1).

This mechanism can be implemented to:

- isolate IT resources in a cloud from non-authorized users
- isolate IT resources in a cloud from non-users
- isolate IT resources in a cloud from cloud consumers
- control the bandwidth that is available to isolated IT resources

Figure 7.1
The dashed line notation used to indicate the boundary of a logical network perimeter.

Logical network perimeters are typically established via network devices that supply and control the connectivity of a data center and are commonly deployed as virtualized IT environments that include:

- *Virtual Firewall* – An IT resource that actively filters network traffic to and from the isolated network while controlling its interactions with the Internet.
- *Virtual Network* – Usually acquired through VLANs, this IT resource isolates the network environment within the data center infrastructure.

Figure 7.2 introduces the notation used to denote these two IT resources. Figure 7.3 depicts a scenario in which one logical network perimeter contains a cloud consumer's on-premise environment, while another contains a cloud provider's cloud-based environment. These perimeters are connected through a VPN that protects communications, since the VPN is typically implemented by point-to-point encryption of the data packets sent between the communicating endpoints.

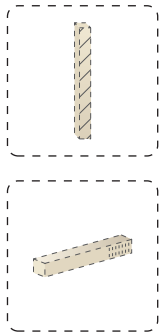

Figure 7.2
The symbols used to represent a virtual firewall (top) and a virtual network (bottom).

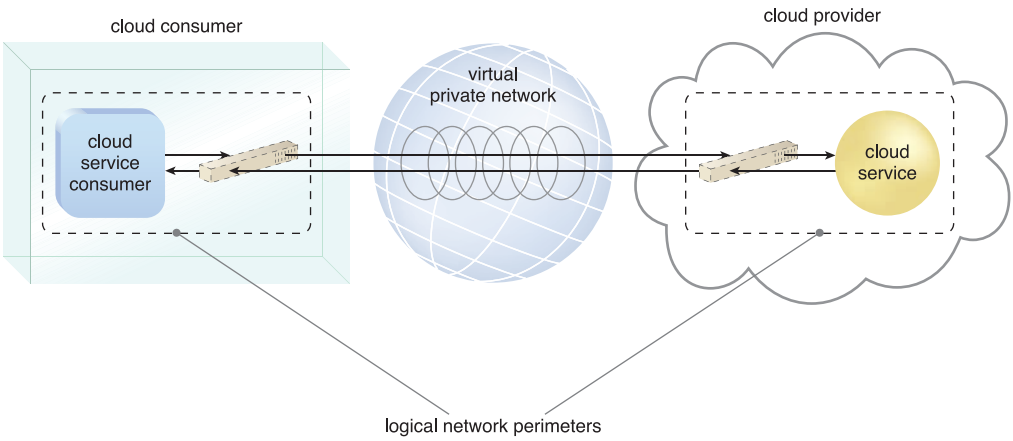

Figure 7.3
Two logical network perimeters surround the cloud consumer and cloud provider environments.

CASE STUDY EXAMPLE

DTGOV has virtualized its network infrastructure to produce a logical network layout favoring network segmentation and isolation. Figure 7.4 depicts the logical network perimeter implemented at each DTGOV data center, as follows:

- The routers that connect to the Internet and extranet are networked to external firewalls, which provide network control and protection to the furthest external network boundaries using virtual networks that logically abstract the external network and extranet perimeters. Devices connected to these network perimeters are loosely isolated and protected from external users. No cloud consumer IT resources are available within these perimeters.
- A logical network perimeter classified as a demilitarized zone (DMZ) is established between the external firewalls and its own firewalls. The DMZ is abstracted as a virtual network hosting the proxy servers (not shown in Figure 7.3) that intermediate access to commonly used network services (DNS, e-mail, Web portal), as well as Web servers with external management functions.
- The network traffic leaving the proxy servers passes through a set of management firewalls that isolate the management network perimeter, which hosts the servers providing the bulk of the management services that cloud consumers can externally access. These services are provided in direct support of self-service and on-demand allocation of cloud-based IT resources.
- All of the traffic to cloud-based IT resources flows through the DMZ to the cloud service firewalls that isolate every cloud consumer's perimeter network, which is abstracted by a virtual network that is also isolated from other networks.
- Both the management perimeter and isolated virtual networks are connected to the intra-data center firewalls, which regulate the network traffic to and from the other DTGOV data centers that are also connected to intra-data center routers at the intra-data center network perimeter.

The virtual firewalls are allocated to and controlled by a single cloud consumer in order to regulate its virtual IT resource traffic. These IT resources are connected through a virtual network that is isolated from other cloud consumers. The virtual firewall and the isolated virtual network jointly form the cloud consumer's logical network perimeter.

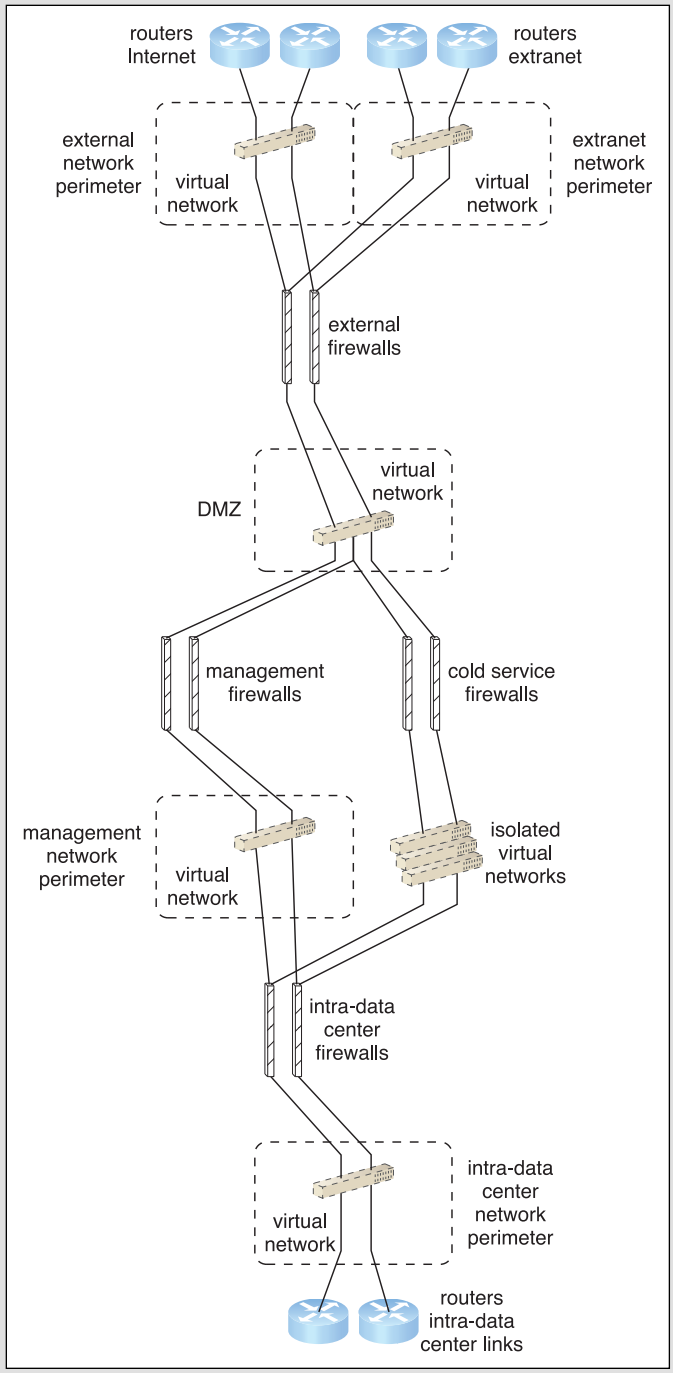

Figure 7.4
A logical network layout is established through a set of logical network perimeters using various firewalls and virtual networks.

7.2 Virtual Server

A *virtual server* is a form of virtualization software that emulates a physical server. Virtual servers are used by cloud providers to share the same physical server with multiple cloud consumers by providing cloud consumers with individual virtual server instances. Figure 7.5 shows three virtual servers being hosted by two physical servers. The number of instances a given physical server can share is limited by its capacity.

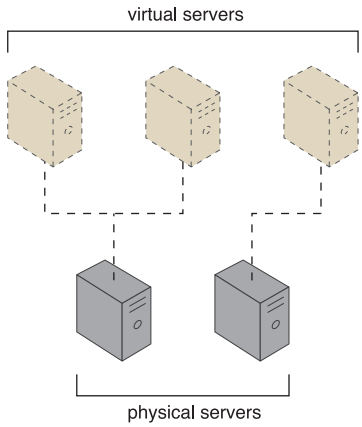

Figure 7.5
The first physical server hosts two virtual servers, while the second physical server hosts one virtual server.

NOTE

- The terms virtual server and virtual machine (VM) are used synonymously throughout this book.
- The hypervisor mechanism referenced in this chapter is described in the *Hypervisor* section in Chapter 8.
- The virtual infrastructure manager (VIM) referenced in this chapter is described in Chapter 9 as part of the *Resource Management System* section.

As a commodity mechanism, the virtual server represents the most foundational building block of cloud environments. Each virtual server can host numerous IT resources, cloud-based solutions, and various other cloud computing mechanisms. The instantiation of virtual servers from image files is a resource allocation process that can be completed rapidly and on-demand.

Cloud consumers that install or lease virtual servers can customize their environments independently from other cloud consumers that may be using virtual servers hosted by the same underlying physical server. Figure 7.6 depicts a virtual server that hosts a cloud service being accessed by Cloud Service Consumer B, while Cloud Service Consumer A accesses the virtual server directly to perform an administration task.

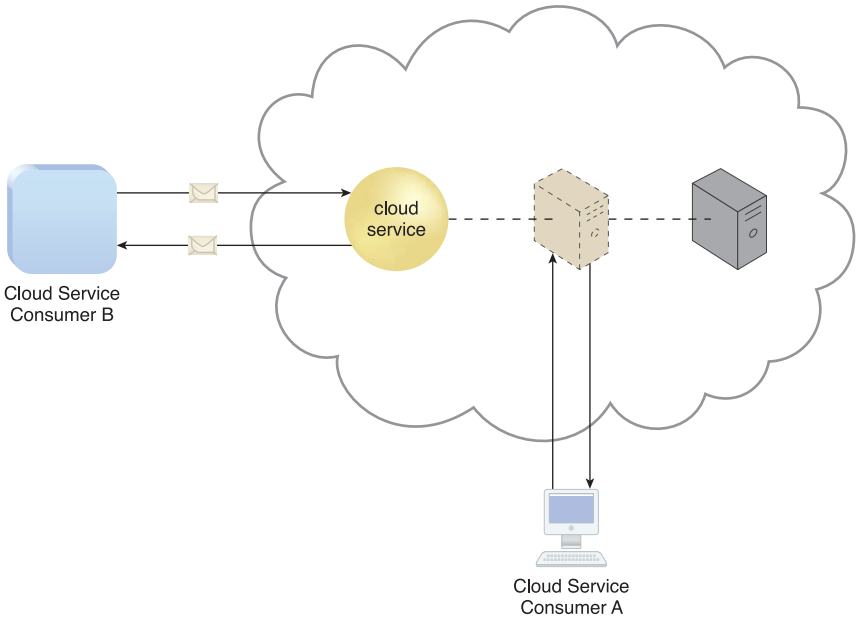

Figure 7.6
A virtual server hosts an active cloud service and is further accessed by a cloud consumer for administrative purposes.

CASE STUDY EXAMPLE

DTGOV's IaaS environment contains hosted virtual servers that were instantiated on physical servers running the same hypervisor software that controls the virtual servers. Their VIM is used to coordinate the physical servers in relation to the creation of virtual server instances. This approach is used at each data center to apply a uniform implementation of the virtualization layer.

Figure 7.7 depicts several virtual servers running over physical servers, all of which are jointly controlled by a central VIM.

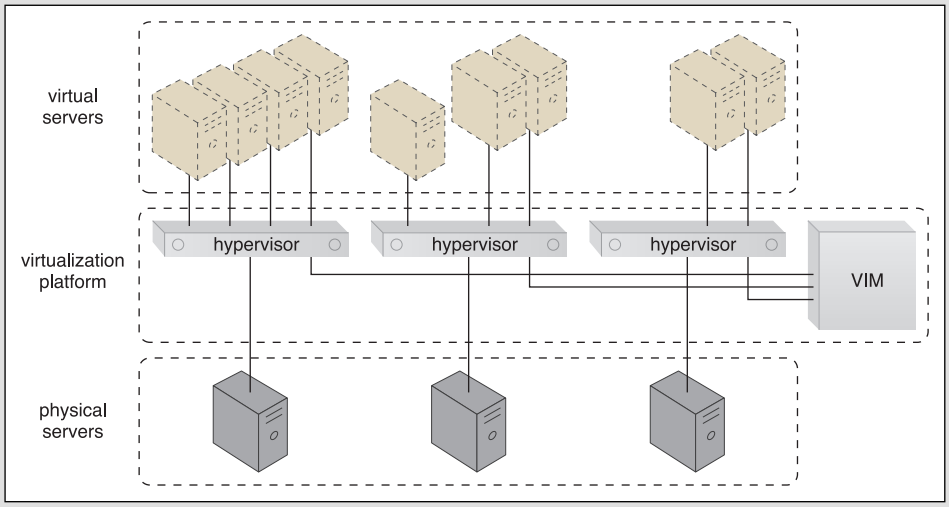

Figure 7.7
Virtual servers are created via the physical servers' hypervisors and a central VIM.

In order to enable the on-demand creation of virtual servers, DTGOV provides cloud consumers with a set of template virtual servers that are made available through pre-made VM images.

These VM images are files that represent the virtual disk images used by the hypervisor to boot the virtual server. DTGOV enables the template virtual servers to have various initial configuration options that differ, based on operating system, drivers, and management tools being used. Some template virtual servers also have additional, pre-installed application server software.

The following virtual server packages are offered to DTGOV's cloud consumers. Each package has different pre-defined performance configurations and limitations:

- *Small Virtual Server Instance* – 1 virtual processor core, 4 GB of virtual RAM, 20 GB of storage space in the root file system
- *Medium Virtual Server Instance* – 2 virtual processor cores, 8 GB of virtual RAM, 20 GB of storage space in the root file system

- *Large Virtual Server Instance* – 8 virtual processor cores, 16 GB of virtual RAM, 20 GB of storage space in the root file system
- *Memory Large Virtual Server Instance* – 8 virtual processor cores, 64 GB of virtual RAM, 20 GB of storage space in the root file system
- *Processor Large Virtual Server Instance* – 32 virtual processor cores, 16 GB of virtual RAM, 20 GB of storage space in the root file system
- *Ultra-Large Virtual Server Instance* – 128 virtual processor cores, 512 GB of virtual RAM, 40 GB of storage space in the root file system

Additional storage capacity can be added to a virtual server by attaching a virtual disk from a cloud storage device. All of the template virtual machine images are stored on a common cloud storage device that is accessible only through the cloud consumers' management tools that are used to control the deployed IT resources. Once a new virtual server needs to be instantiated, the cloud consumer can choose the most suitable virtual server template from the list of available configurations. A copy of the virtual machine image is made and allocated to the cloud consumer, who can then assume the administrative responsibilities.

The allocated VM image is updated whenever the cloud consumer customizes the virtual server. After the cloud consumer initiates the virtual server, the allocated VM image and its associated performance profile is passed to the VIM, which creates the virtual server instance from the appropriate physical server.

DTGOV uses the process described in Figure 7.8 to support the creation and management of virtual servers that have different initial software configurations and performance characteristics.

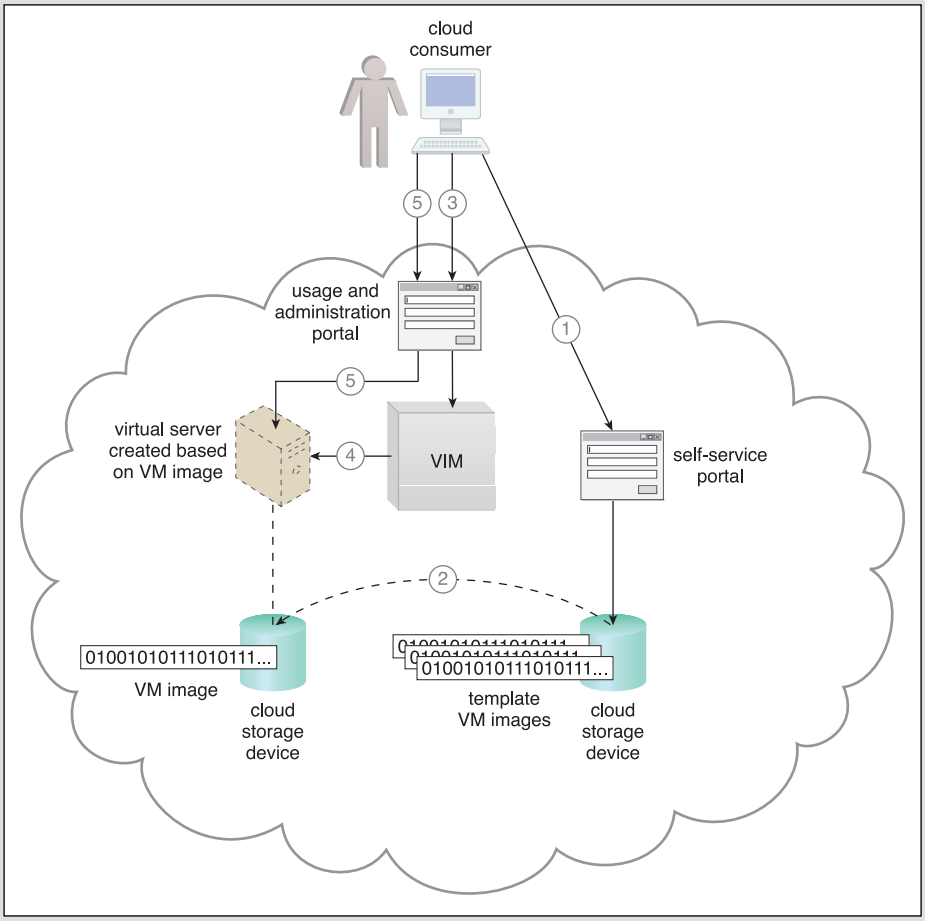

Figure 7.8

The cloud consumer uses the self-service portal to select a template virtual server for creation (1). A copy of the corresponding VM image is created in a cloud consumer-controlled cloud storage device (2). The cloud consumer initiates the virtual server using the usage and administration portal (3), which interacts with the VIM to create the virtual server instance via the underlying hardware (4). The cloud consumer is able to use and customize the virtual server via other features on the usage and administration portal (5). (Note that the self-service portal and usage and administration portal are explained in Chapter 9.)

7.3 Cloud Storage Device

The *cloud storage device* mechanism represents storage devices that are designed specifically for cloud-based provisioning. Instances of these devices can be virtualized, similar to how physical servers can spawn virtual server images. Cloud storage devices are commonly able to provide fixed-increment capacity allocation in support of the pay-per-use mechanism. Cloud storage devices can be exposed for remote access via cloud storage services.

NOTE

This is a parent mechanism that represents cloud storage devices in general. There are numerous specialized cloud storage devices, several of which are described in the architectural models covered in Part III of this book.

A primary concern related to cloud storage is the security, integrity, and confidentiality of data, which becomes more prone to being compromised when entrusted to external cloud providers and other third parties. There can also be legal and regulatory implications that result from relocating data across geographical or national boundaries. Another issue applies specifically to the performance of large databases. LANs provide locally stored data with network reliability and latency levels that are superior to those of WANs.

Cloud Storage Levels

Cloud storage device mechanisms provide common logical units of data storage, such as:

- *Files* – Collections of data are grouped into files that are located in folders.
- *Blocks* – The lowest level of storage and the closest to the hardware, a block is the smallest unit of data that is still individually accessible.
- *Datasets* – Sets of data are organized into a table-based, delimited, or record format.
- *Objects* – Data and its associated metadata are organized as Web-based resources.

Each of these data storage levels is commonly associated with a certain type of technical interface which corresponds to a particular type of cloud storage device and cloud storage service used to expose its API (Figure 7.9).

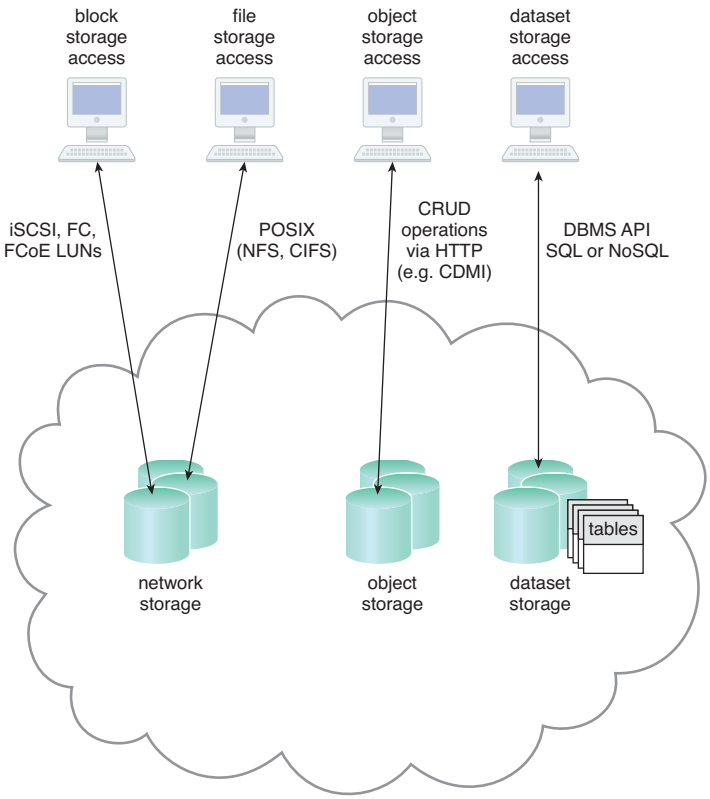

Figure 7.9
Different cloud service consumers utilize different technologies to interface with virtualized cloud storage devices. (Adapted from the CDMI Cloud Storage Reference Model.)

Network Storage Interfaces

Legacy network storage most commonly falls under the category of network storage interfaces. It includes storage devices in compliance with industry standard protocols, such as SCSI for storage blocks and the server message block (SMB), common Internet file system (CIFS), and network file system (NFS) for file and network storage. File storage entails storing individual data in separate files that can be different sizes and formats and organized into folders and subfolders. Original files are often replaced by the new files that are created when data has been modified.

When a cloud storage device mechanism is based on this type of interface, its data searching and extraction performance will tend to be suboptimal. Storage processing levels and thresholds for file allocation are usually determined by the file system itself.

Block storage requires data to be in a fixed format (known as a *data block*), which is the smallest unit that can be stored and accessed and the storage format closest to hardware. Using either the logical unit number (LUN) or virtual volume block-level storage will typically have better performance than file-level storage.

Object Storage Interfaces

Various types of data can be referenced and stored as Web resources. This is referred to as object storage, which is based on technologies that can support a range of data and media types. Cloud Storage Device mechanisms that implement this interface can typically be accessed via REST or Web service-based cloud services using HTTP as the prime protocol. The Storage Networking Industry Association's Cloud Data Management Interface (SNIA's CDMI) supports the use of object storage interfaces.

Database Storage Interfaces

Cloud storage device mechanisms based on database storage interfaces typically support a query language in addition to basic storage operations. Storage management is carried out using a standard API or an administrative user-interface.

This classification of storage interface is divided into two main categories according to storage structure, as follows.

Relational Data Storage

Traditionally, many on-premise IT environments store data using relational databases or relational database management systems (RDBMSs). Relational databases (or relational storage devices) rely on tables to organize similar data into rows and columns. Tables can have relationships with each other to give the data increased structure, to protect data integrity, and to avoid data redundancy (which is referred to as data normalization). Working with relational storage commonly involves the use of the industry standard Structured Query Language (SQL).

A cloud storage device mechanism implemented using relational data storage could be based on any number of commercially available database products, such as IBM DB2, Oracle Database, Microsoft SQL Server, and MySQL.

Challenges with cloud-based relational databases commonly pertain to scaling and performance. Scaling a relational cloud storage device vertically can be more complex and cost-ineffective than horizontal scaling. Databases with complex relationships and/or containing large volumes of data can be afflicted with higher processing overhead and latency, especially when accessed remotely via cloud services.

Non-Relational Data Storage

Non-relational storage (also commonly referred to as *NoSQL* storage) moves away from the traditional relational database model in that it establishes a "looser" structure for stored data with less emphasis on defining relationships and realizing data normalization. The primary motivation for using non-relational storage is to avoid the potential complexity and processing overhead that can be imposed by relational databases. Also, non-relational storage can be more horizontally scalable than relational storage.

The trade-off with non-relational storage is that the data loses much of the native form and validation due to limited or primitive schemas or data models. Furthermore, non-relational repositories don't tend to support relational database functions, such as transactions or joins.

Normalized data exported into a non-relational storage repository will usually become denormalized, meaning that the size of the data will typically grow. An extent of normalization can be preserved, but usually not for complex relationships. Cloud providers often offer non-relational storage that provides scalability and availability of stored data over multiple server environments. However, many non-relational storage mechanisms are proprietary and therefore can severely limit data portability.

CASE STUDY EXAMPLE

DTGOV provides cloud consumers access to a cloud storage device based on an object storage interface. The cloud service that exposes this API offers basic functions on stored objects, such as search, create, delete, and update. The search function uses a hierarchical object arrangement that resembles a file system. DTGOV further offers a cloud service that is used exclusively with virtual servers and enables the creation of cloud storage devices via a block storage network interface. Both cloud services use APIs that are compliant with SNIA's CDMI v1.0.

The object-based cloud storage device has an underlying storage system with variable storage capacity, which is directly controlled by a software component that also exposes the interface. This software enables the creation of isolated cloud storage devices that are allocated to cloud consumers. The storage system uses a security credential management system to administer user-based access control to the device's data objects (Figure 7.10).

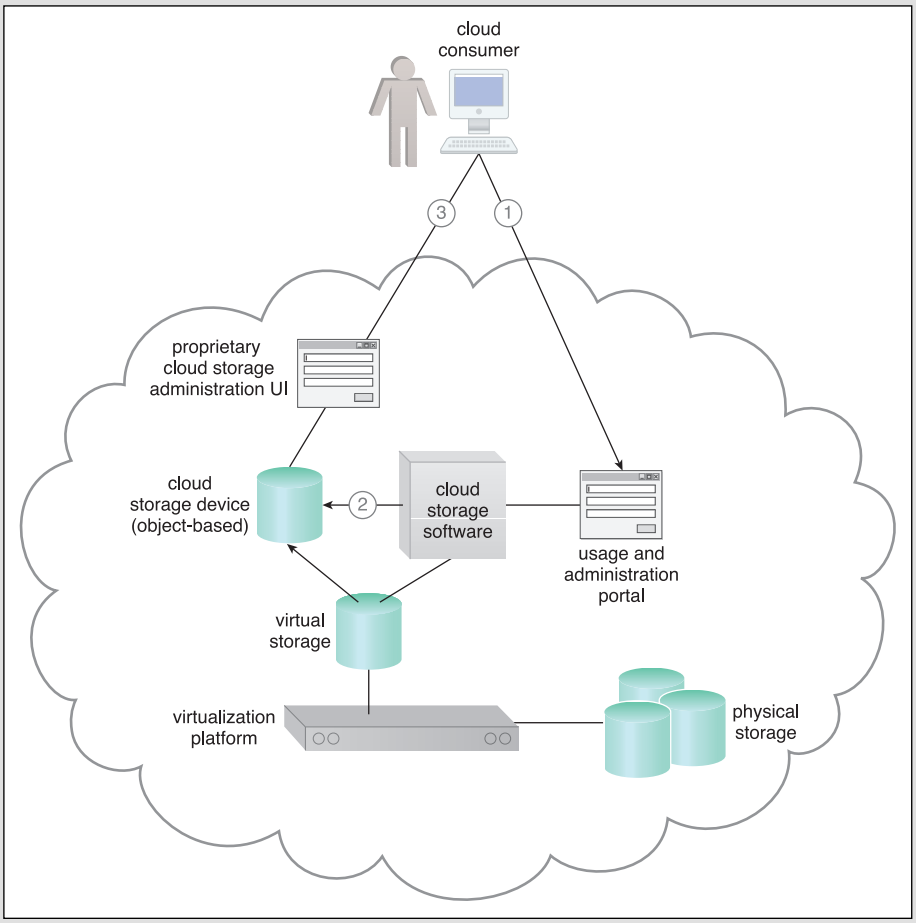

Figure 7.10

The cloud consumer interacts with the usage and administration portal to create a cloud storage device and define access control policies (1). The usage and administration portal interact with the cloud storage software to create the cloud storage device instance and apply the required access policy to its data objects (2). Each data object is assigned to a cloud storage device and all of the data objects are stored in the same virtual storage volume. The cloud consumer uses the proprietary cloud storage device UI to interact directly with the data objects (3). (Note that the usage and administration portal is explained in Chapter 9.)

Access control is granted on a per-object basis and uses separate access policies for creating, reading from, and writing to each data object. Public access permissions are allowed, although they are read-only. Access groups are formed by nominated users that must be previously registered via the credential management system. Data objects can be accessed from both Web applications and Web service interfaces, which are implemented by the cloud storage software.

The creation of the cloud consumers' block-based cloud storage devices is managed by the virtualization platform, which instantiates the LUN's implementation of the virtual storage (Figure 7.11). The cloud storage device (or the LUN) must be assigned by the VIM to an existing virtual server before it can be used. The capacity of block-based cloud storage devices is expressed by one GB increments. It can be created as fixed storage that cloud consumers can modify administratively or as variable size storage that has an initial 5 GB capacity that automatically increases and decreases by 5 GB increments according to usage demands.

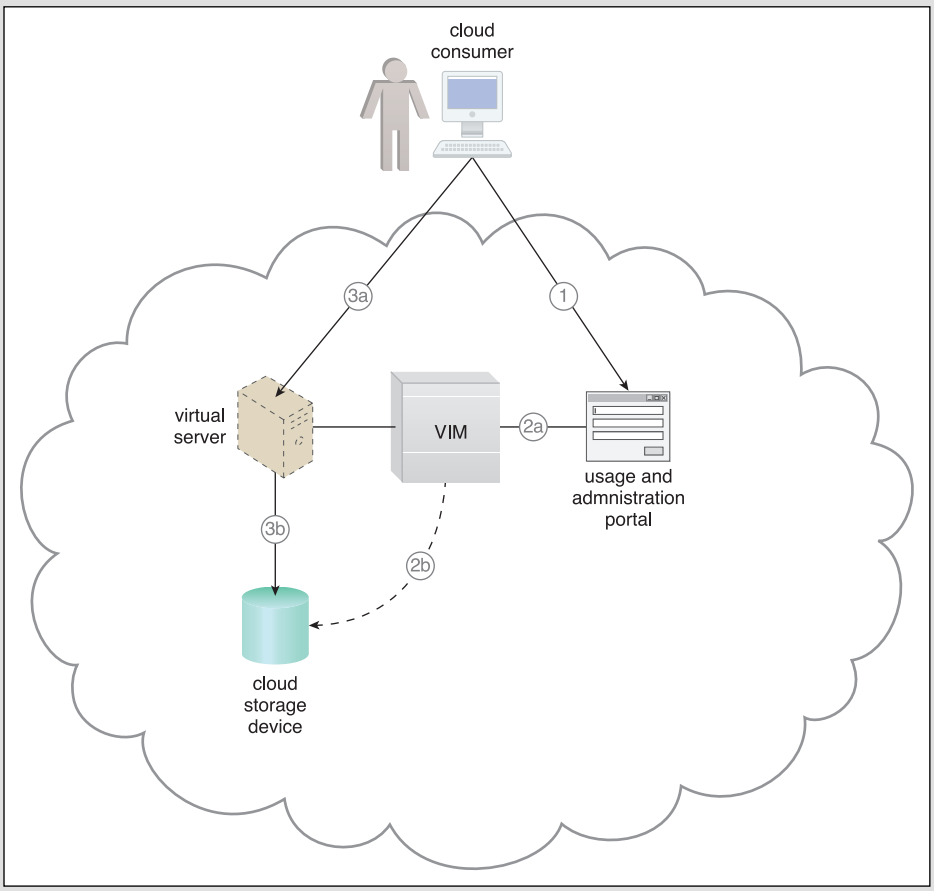

Figure 7.11

The cloud consumer uses the usage and administration portal to create and assign a cloud storage device to an existing virtual server (1). The usage and administration portal interacts with the VIM software (2a), which creates and configures the appropriate LUN (2b). Each cloud storage device uses a separate LUN controlled by the virtualization platform. The cloud consumer remotely logs into the virtual server directly (3a) to access the cloud storage device (3b).

7.4 Cloud Usage Monitor

The *cloud usage monitor* mechanism is a lightweight and autonomous software program responsible for collecting and processing IT resource usage data.

NOTE

This is a parent mechanism that represents a broad range of cloud usage monitors, several of which are established as specialized mechanisms in Chapter 8 and several more of which are described in the cloud architectural models covered in Part III of this book.

Depending on the type of usage metrics they are designed to collect and the manner in which usage data needs to be collected, cloud usage monitors can exist in different formats. The upcoming sections describe three common agent-based implementation formats. Each can be designed to forward collected usage data to a log database for post-processing and reporting purposes.

Monitoring Agent

A *monitoring agent* is an intermediary, event-driven program that exists as a service agent and resides along existing communication paths to transparently monitor and analyze dataflows (Figure 7.12). This type of cloud usage monitor is commonly used to measure network traffic and message metrics.

Resource Agent

A *resource agent* is a processing module that collects usage data by having event-driven interactions with specialized resource software (Figure 7.13). This module is used to monitor usage metrics based on pre-defined, observable events at the resource software level, such as initiating, suspending, resuming, and vertical scaling.

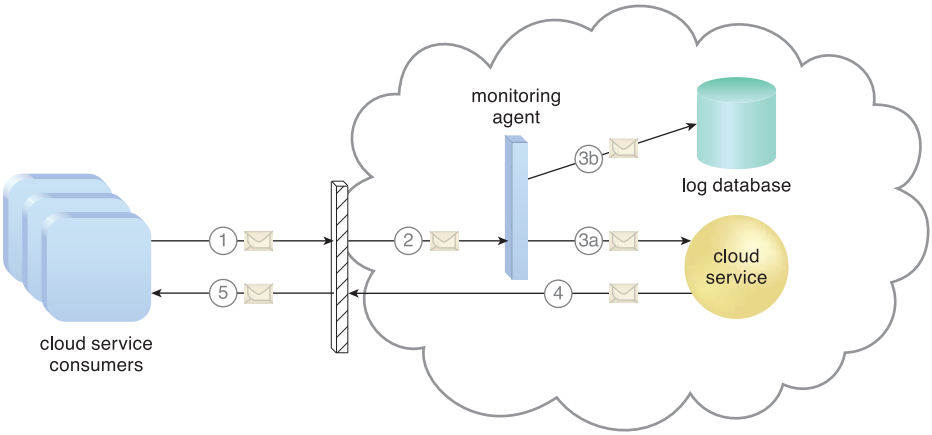

Figure 7.12

A cloud service consumer sends a request message to a cloud service (1). The monitoring agent intercepts the message to collect relevant usage data (2) before allowing it to continue to the cloud service (3a). The monitoring agent stores the collected usage data in a log database (3b). The cloud service replies with a response message (4) that is sent back to the cloud service consumer without being intercepted by the monitoring agent (5).

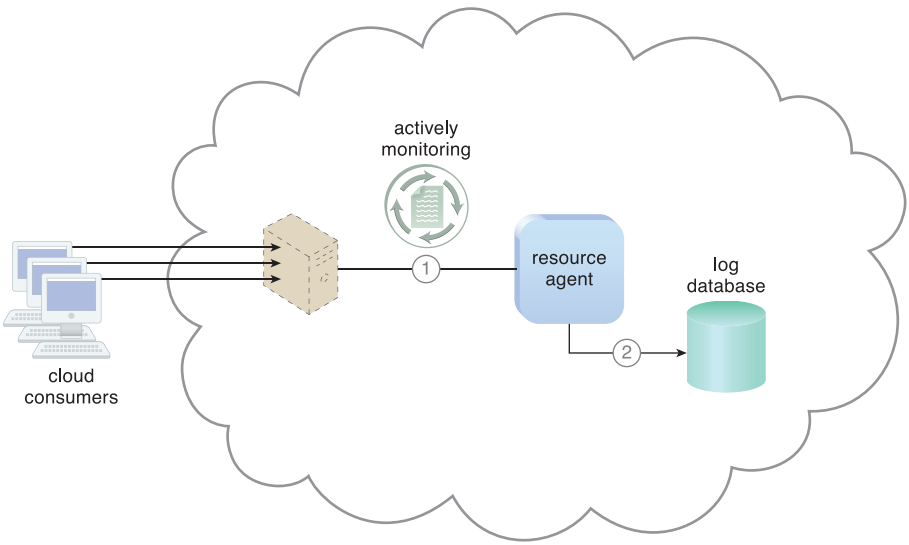

Figure 7.13

The resource agent is actively monitoring a virtual server and detects an increase in usage (1). The resource agent receives a notification from the underlying resource management program that the virtual server is being scaled up and stores the collected usage data in a log database, as per its monitoring metrics (2).

Polling Agent

A *polling agent* is a processing module that collects cloud service usage data by polling IT resources. This type of cloud service monitor is commonly used to periodically monitor IT resource status, such as uptime and downtime (Figure 7.14).

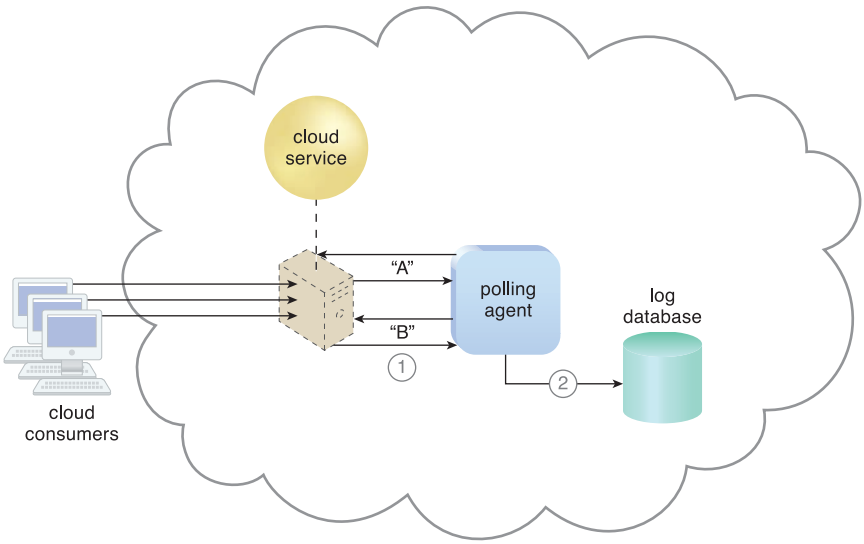

Figure 7.14

A polling agent monitors the status of a cloud service hosted by a virtual server by sending periodic polling request messages and receiving polling response messages that report usage status "A" after a number of polling cycles, until it receives a usage status of "B" (1), upon which the polling agent records the new usage status in the log database (2).

CASE STUDY EXAMPLE

One of the challenges encountered during DTGOV's cloud adoption initiative has been ensuring that their collected usage data is accurate. The resource allocation methods of previous IT outsourcing models had resulted in their clients being billed chargeback fees based on the number of physical servers that was listed in annual leasing contracts, regardless of actual usage.

DTGOV now needs to define a model that allows virtual servers of varying performance levels to be leased and billed hourly. Usage data needs to be at an extremely granular level in order to achieve the necessary degree of accuracy. DTGOV

implements a resource agent that relies on the resource usage events generated by the VIM platform to calculate the virtual server usage data.

The resource agent is designed with logic and metrics that are based on the following rules:

1. Each resource usage event that is generated by the VIM software can contain the following data:
   - Event Type (EV_TYPE) – Generated by the VIM platform, there are five types of events:

     VM Starting (creation at the hypervisor)

     VM Started (completion of the boot procedure)

     VM Stopping (shutting down)

     VM Stopped (termination at the hypervisor)

     VM Scaled (change of performance parameters)
   - VM Type (VM_TYPE) – This represents a type of virtual server, as dictated by its performance parameters. A predefined list of possible virtual server configurations provides the parameters that are described by the metadata whenever a VM starts or scales.
   - Unique VM Identifier (VM_ID) – This identifier is provided by the VIM platform.
   - Unique Cloud Consumer Identifier (CS_ID) – Another identifier provided by the VIM platform to represent the cloud consumer.
   - Event Timestamp (EV_T) – An identification of an event occurrence that is expressed in date-time format, with the time zone of the data center and referenced to UTC as defined in RFC 3339 (as per the ISO 8601 profile).
2. Usage measurements are recorded for every virtual server that a cloud consumer creates.

3. Usage measurements are recorded for a measurement period whose length is defined by two timestamps called t_{start} and t_{end}. The start of the measurement period defaults to the beginning of the calendar month (t_{start} = 2012-12-01T00:00:00-08:00) and finishes at the end of the calendar month (t_{end} = 2012-12-31T23:59:59-08:00). Customized measurement periods are also supported.
4. Usage measurements are recorded at each minute of usage. The virtual server usage measurement period starts when the virtual server is created at the hypervisor and stops at its termination.
5. Virtual servers can be started, scaled, and stopped multiple times during the measurement period. The time interval between each occurrence i (i = 1, 2, 3, …) of these pairs of successive events that are declared for a virtual server is called a usage cycle that is known as T_{cycle_i}:
   - VM_Starting, VM_Stopping – VM size is unchanged at the end of the cycle
   - VM_Starting, VM_Scaled – VM size has changed at the end of the cycle
   - VM_Scaled, VM_Scaled – VM size has changed while scaling, at the end of the cycle
   - VM_Scaled, VM_Stopping – VM size has changed at the end of the cycle
6. The total usage, U_{total}, for each virtual server during the measurement period is calculated using the following resource usage event log database equations:
   - For each VM_TYPE and VM_ID in the log database: $U_{total\_VM\_type\_j} = \sum_{t_{start}}^{t_{end}} T_{cycle_i}$
   - As per the total usage time that is measured for each VM_TYPE, the vector of usage for each VM_ID is U_{total}: U_{total} = {type 1, $U_{total\_VM\_type\_1}$, type 2, $U_{total\_VM\_type\_2}$, …}

Figure 7.15 depicts the resource agent interacting with the VIM's event-driven API.

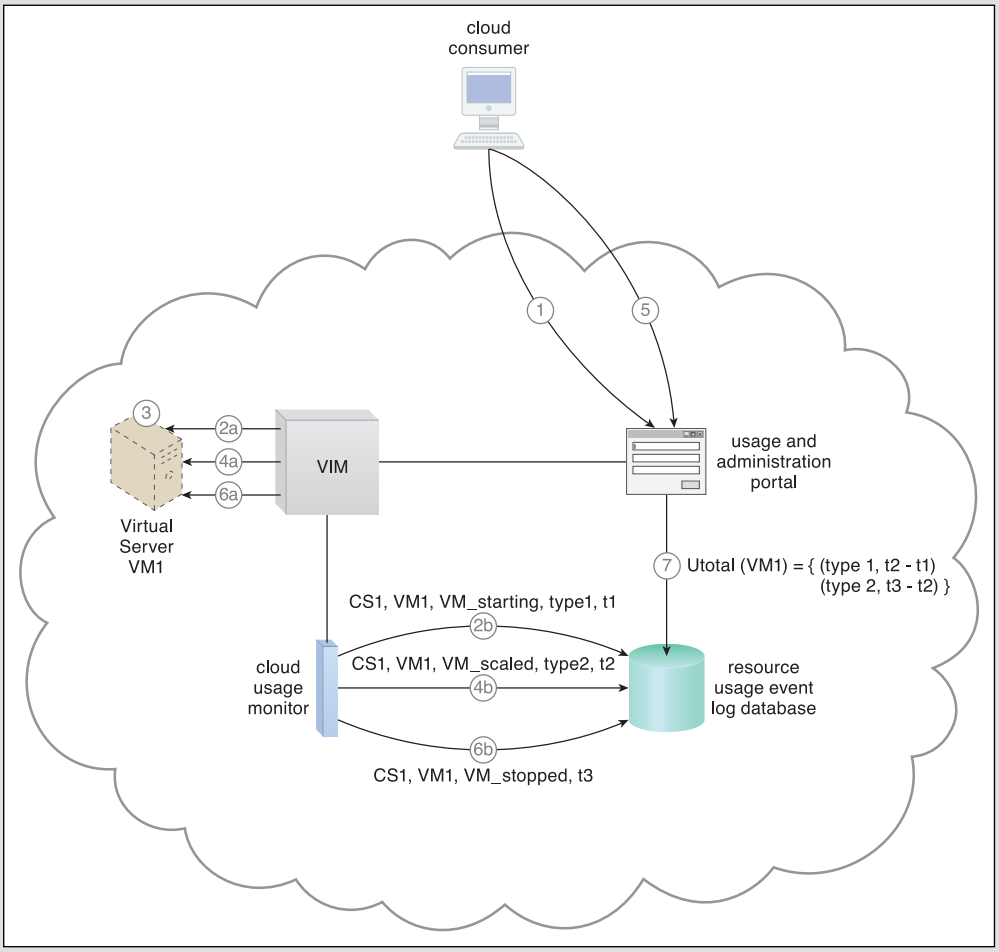

Figure 7.15

The cloud consumer (CS_ID = CS1) requests the creation of a virtual server (VM_ID = VM1) of configuration size *type 1* (VM_TYPE = type1) (1). The VIM creates the virtual server (2a). The VIM's event-driven API generates a resource usage event with timestamp = *t1*, which the cloud usage monitor software agent captures and records in the resource usage event log database (2b). Virtual server usage increases and reaches the auto-scaling threshold (3). The VIM scales up Virtual Server VM1 (4a) from configuration *type 1* to *type 2* (VM_TYPE = type2). The VIM's event-driven API generates a resource usage event with timestamp = *t2*, which is captured and recorded at the resource usage event log database by the cloud usage monitor software agent (4b). The cloud consumer shuts down the virtual server (5). The VIM stops Virtual Server VM1 (6a) and its event-driven API generates a resource usage event with timestamp = *t3*, which the cloud usage monitor software agent captures and records at the log database (6b). The usage and administration portal accesses the log database and calculates the total usage (Utotal) for Virtual Server Utotal VM1 (7).

7.5 Resource Replication

Defined as the creation of multiple instances of the same IT resource, replication is typically performed when an IT resource's availability and performance need to be enhanced. Virtualization technology is used to implement the *resource replication* mechanism to replicate cloud-based IT resources (Figure 7.16).

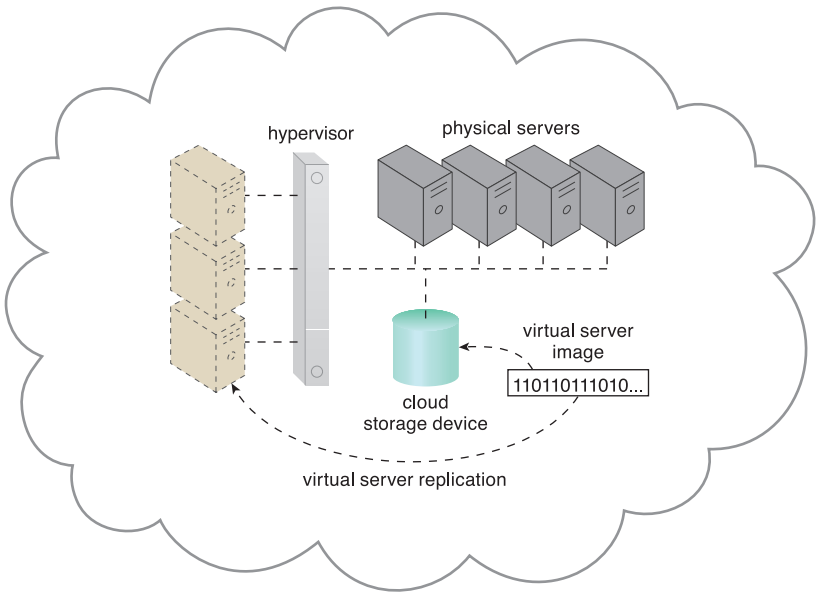

Figure 7.16
The hypervisor replicates several instances of a virtual server, using a stored virtual server image.

NOTE

This is a parent mechanism that represents different types of software programs capable of replicating IT resources. The most common example is the hypervisor mechanism described in Chapter 8. For example, the virtualization platform's hypervisor can access a virtual server image to create several instances, or to deploy and replicate ready-made environments and entire applications. Other common types of replicated IT resources include cloud service implementations and various forms of data and cloud storage device replication.

CASE STUDY EXAMPLE

DTGOV establishes a set of high-availability virtual servers that can be automatically relocated to physical servers running in different data centers in response to severe failure conditions. This is illustrated in the scenario depicted in Figures 7.17 to 7.19, where a virtual server that resides on a physical server running at one data center experiences a failure condition. VIMs from different data centers coordinate to overcome the unavailability by reallocating the virtual server to a different physical server running in another data center.

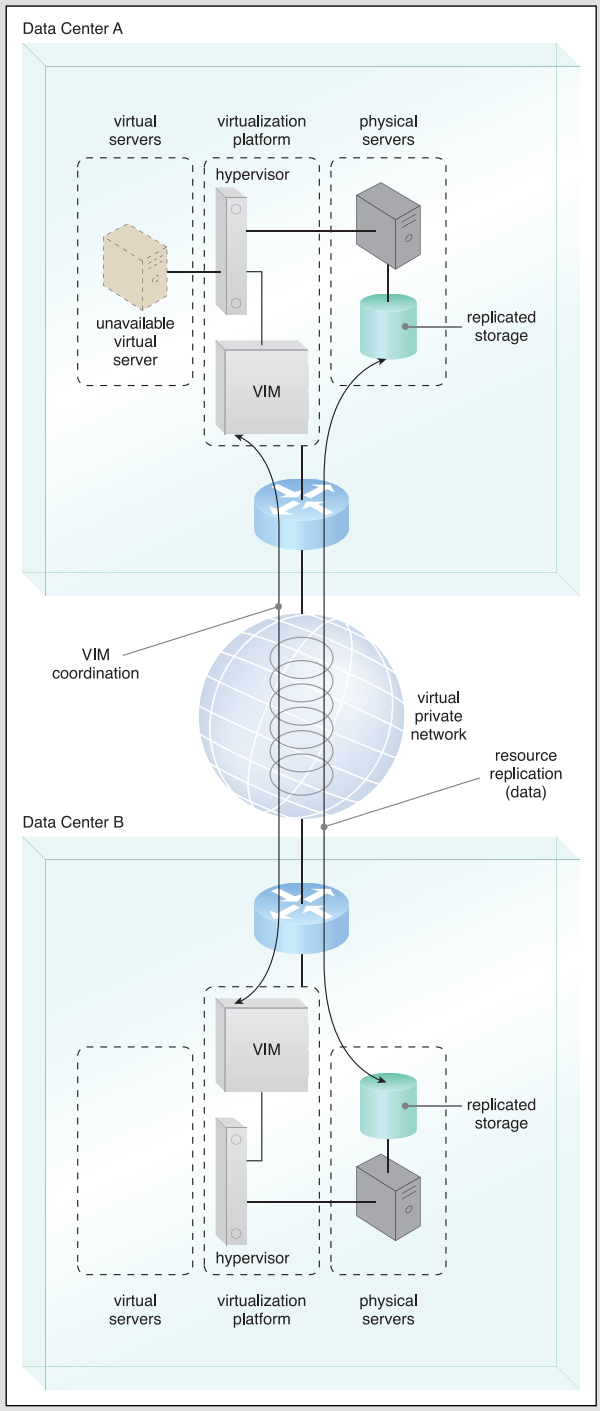

Figure 7.17
A high-availability virtual server is running in Data Center A. VIM instances in Data Centers A and B are executing a coordination function that allows detection of failure conditions. Stored VM images are replicated between data centers as a result of the high-availability architecture.

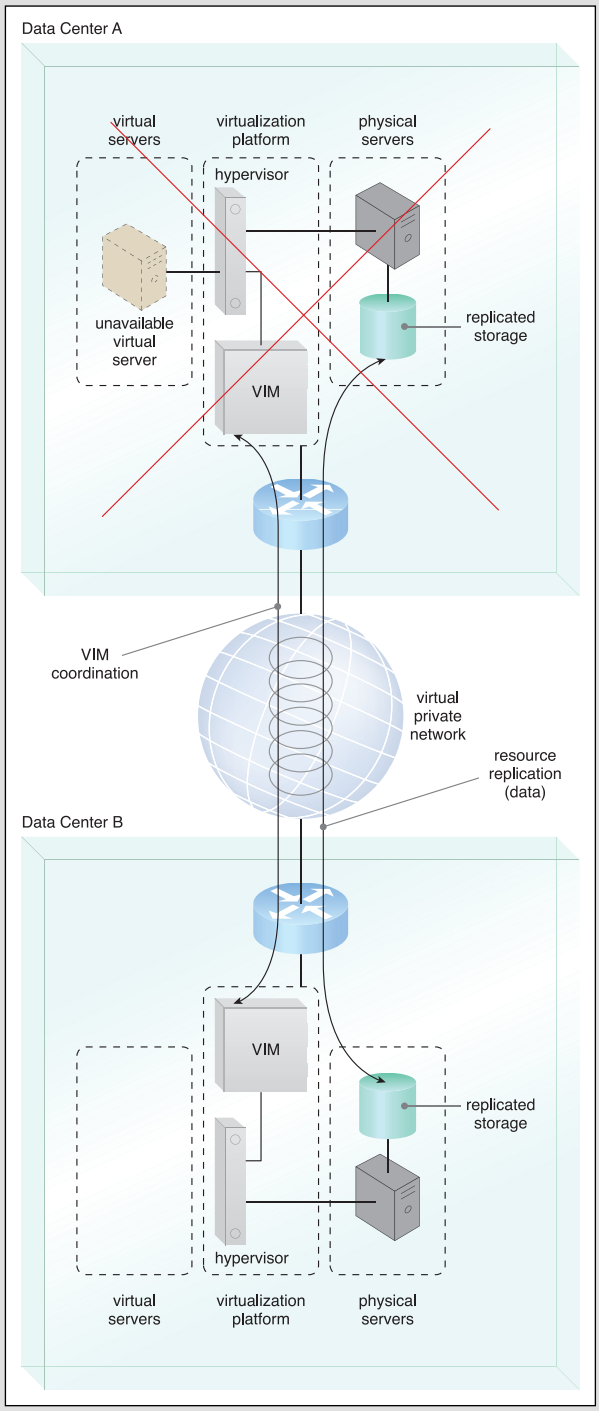

Figure 7.18
The virtual server becomes unavailable in Data Center A. The VIM in Data Center B detects the failure condition and starts to reallocate the high-availability server from Data Center A to Data Center B.

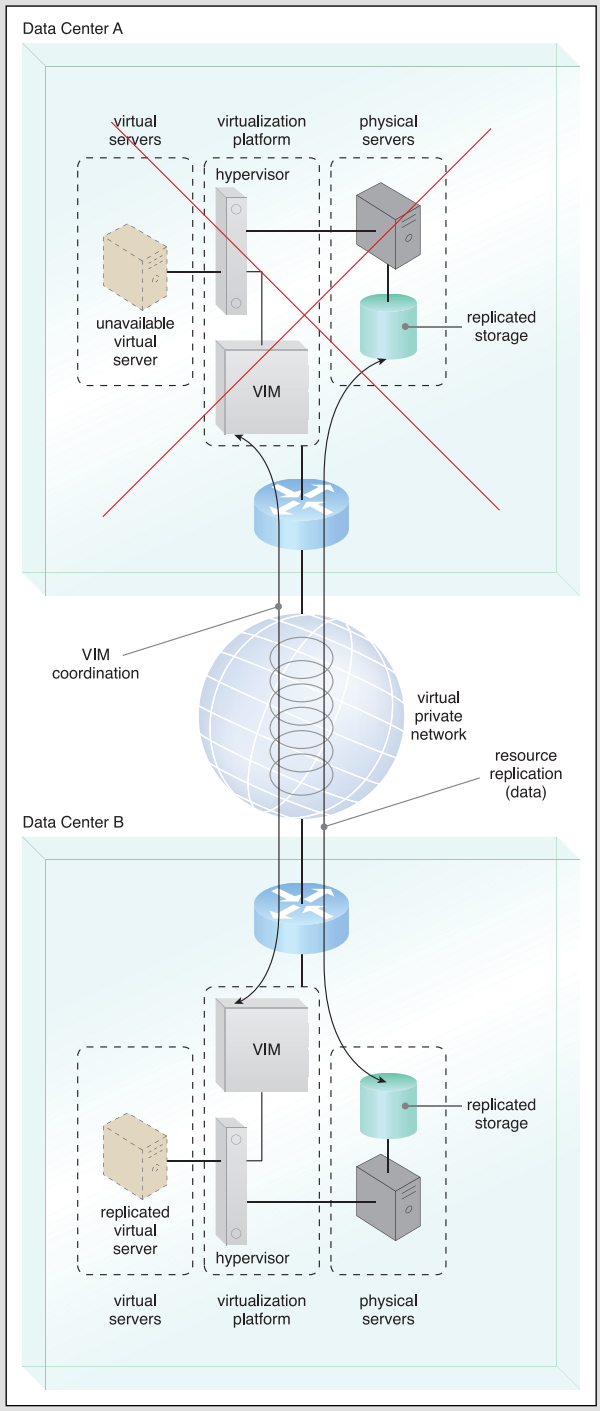

Figure 7.19
A new instance of the virtual server is created and made available in Data Center B.

7.6 Ready-Made Environment

The *ready-made environment* mechanism (Figure 7.20) is a defining component of the PaaS cloud delivery model that represents a pre-defined, cloud-based platform comprised of a set of already installed IT resources, ready to be used and customized by a cloud consumer. These environments are utilized by cloud consumers to remotely develop and deploy their own services and applications within a cloud. Typical ready-made environments include pre-installed IT resources, such as databases, middleware, development tools, and governance tools.

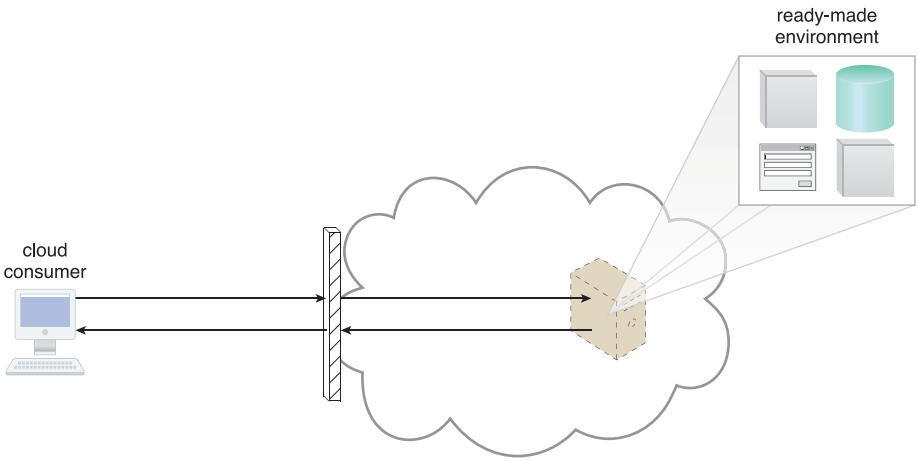

Figure 7.20
A cloud consumer accesses a ready-made environment hosted on a virtual server.

A ready-made environment is generally equipped with a complete software development kit (SDK) that provides cloud consumers with programmatic access to the development technologies that comprise their preferred programming stacks.

Middleware is available for multitenant platforms to support the development and deployment of Web applications. Some cloud providers offer runtime execution environments for cloud services that are based on different runtime performance and billing parameters. For example, a front-end instance of a cloud service can be configured to respond to time-sensitive requests more effectively than a back-end instance. The former variation will be billed at a different rate than the latter.

As further demonstrated in the upcoming case study example, a solution can be partitioned into groups of logic that can be designated for both frontend and backend instance invocation so as to optimize runtime execution and billing.

CASE STUDY EXAMPLE

ATN developed and deployed several non-critical business applications using a leased PaaS environment. One was a Java-based Part Number Catalog Web application used for the switches and routers they manufacture. This application is used by different factories, but it does not manipulate transaction data, which is instead processed by a separate stock control system.

The application logic was split into front-end and back-end processing logic. The front-end logic was used to process simple queries and updates to the catalog. The back-end part contains the logic required to render the complete catalog and correlate similar components and legacy part numbers.

Figure 7.21 illustrates the development and deployment environment for ATN's Part Number Catalog application. Note how the cloud consumer assumes both the developer and end-user roles.

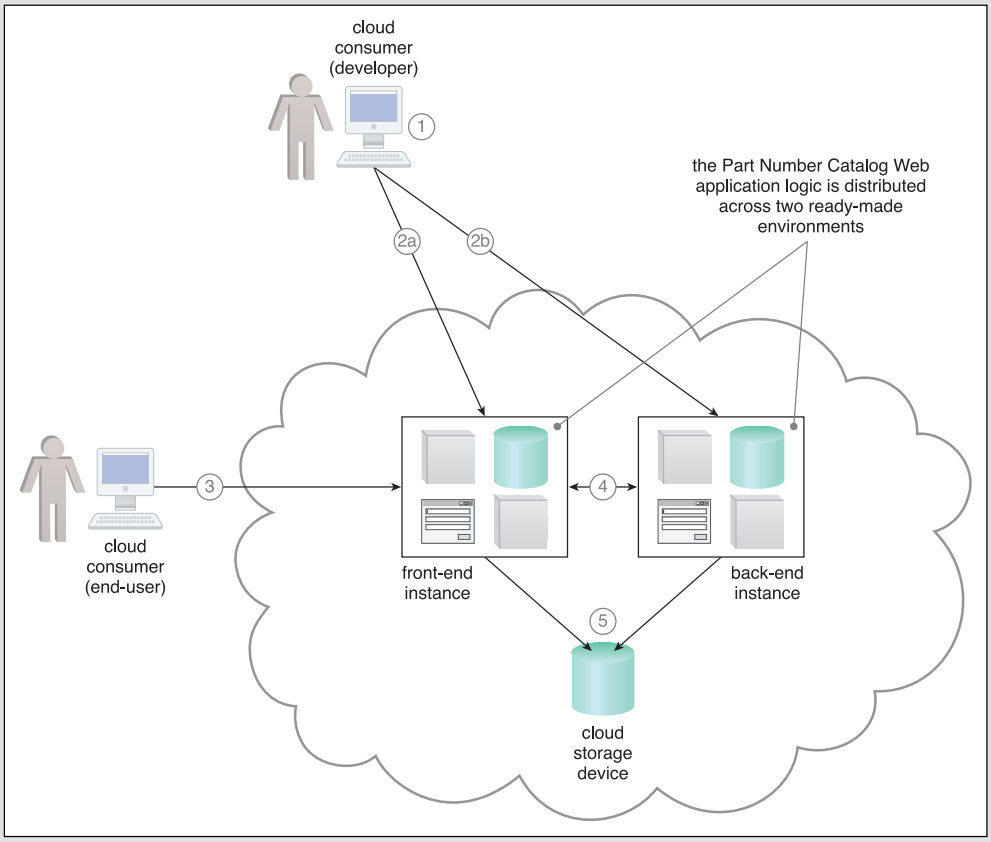

Figure 7.21

The developer uses the provided SDK to develop the Part Number Catalog Web application (1). The application software is deployed on a Web platform that was established by two ready-made environments called the front-end instance (2a) and the back-end instance (2b). The application is made available for usage and one end-user accesses its front-end instance (3). The software running in the front-end instance invokes a long-running task at the back-end instance that corresponds to the processing required by the end-user (4). The application software deployed at both the front-end and back-end instances is backed by a cloud storage device that provides persistent storage of the application data (5).