PART

# 2 Cloud Application Programming and the Aneka Platform

CHAPTER 5

# Aneka

## Cloud Application Platform

c0005

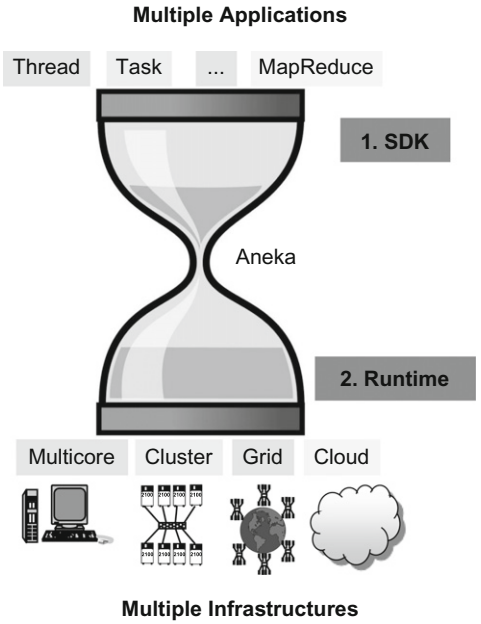
p0065 Aneka is Manjrasoft Pty. Ltd.'s solution for developing, deploying, and managing cloud applications. Aneka consists of a scalable cloud middleware that can be deployed on top of heterogeneous computing resources. It offers an extensible collection of services coordinating the execution of applications, helping administrators monitor the status of the cloud, and providing integration with existing cloud technologies. One of Aneka's key advantages is its extensible set of application programming interfaces (APIs) associated with different types of programming models—such as Task, Thread, and MapReduce—used for developing distributed applications, integrating new capabilities into the cloud, and supporting different types of cloud deployment models: public, private, and hybrid (see Figure 5.1). These features differentiate Aneka from infrastructure management software and characterize it as a platform for developing, deploying, and managing execution of applications on various types of clouds.

p0070 This chapter provides a complete overview of the framework by first describing the architecture of the system. It introduces Aneka's components and the fundamental services that make up the Aneka Cloud and discusses some common deployment scenarios.

## s0010 5.1 Framework overview

p0075 Aneka is a software platform for developing cloud computing applications. It allows harnessing of disparate computing resources and managing them into a unique virtual domain—the Aneka Cloud—in which applications are executed. According to the Cloud Computing Reference Model presented in Chapter 1, Aneka is a *pure PaaS* solution for cloud computing. Aneka is a cloud middleware product that can be deployed on a heterogeneous set of resources: a network of computers, a multicore server, datacenters, virtual cloud infrastructures, or a mixture of these. The framework provides both middleware for managing and scaling distributed applications and an extensible set of APIs for developing them.

p0080 Figure 5.2 provides a complete overview of the components of the Aneka framework. The core infrastructure of the system provides a uniform layer that allows the framework to be deployed over different platforms and operating systems. The physical and virtual resources representing the bare metal of the cloud are managed by the Aneka container, which is installed on each node and constitutes the basic building block of the middleware. A collection of interconnected containers constitute the Aneka Cloud: a single domain in which services are made available to users and

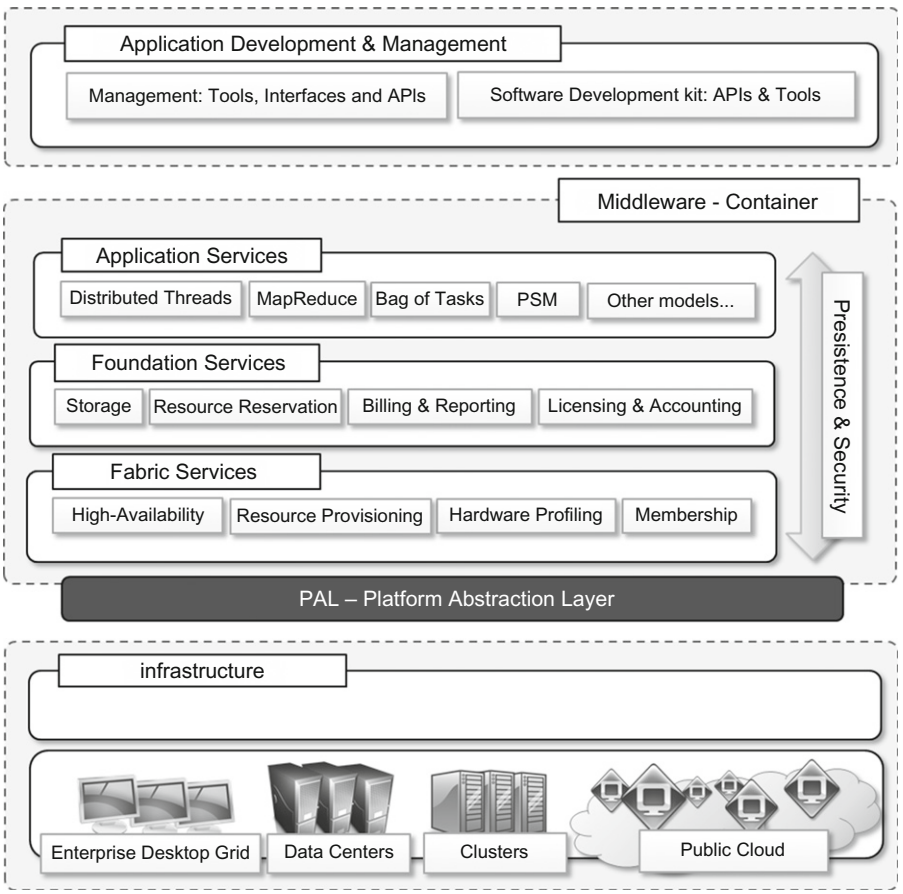



f0010 **FIGURE 5.1**

Aneka's capabilities at a glance.

developers. The container features three different classes of services: *Fabric Services*, *Foundation Services*, and *Execution Services*. These take care of infrastructure management, supporting services for the Aneka Cloud, and application management and execution, respectively. These services are made available to developers and administrators by means of the application management and development layer, which includes interfaces and APIs for developing cloud applications and the management tools and interfaces for controlling Aneka Clouds.

p0085 Aneka implements a service-oriented architecture (SOA), and services are the fundamental components of an Aneka Cloud. Services operate at container level and, except for the platform abstraction layer, they provide developers, users, and administrators with all features offered by the framework. Services also constitute the extension and customization point of Aneka Clouds: The infrastructure allows for the integration of new services or replacement of the existing ones with a different implementation. The framework includes the basic services for infrastructure and node management, application execution, accounting, and system monitoring; existing services can be extended and new features can be added to the cloud by dynamically plugging new ones into the container. Such extensible and flexible infrastructure enables Aneka Clouds to support different programming and execution models for applications. A programming model represents a collection of abstractions that developers can use to express distributed applications; the runtime support for a programming model is constituted by a collection of execution and foundation services interacting together to carry out application execution. Thus, the implementation of a new model requires the development of the specific programming abstractions used by application developers and the




f0015 **FIGURE 5.2**

Aneka framework overview.

services, providing runtime support for them. Programming models are just one aspect of application management and execution. Within an Aneka Cloud environment, there are different aspects involved in providing a scalable and elastic infrastructure and distributed runtime for applications. These services involve:

u0050 • *Elasticity and scaling*. By means of the dynamic provisioning service, Aneka supports dynamically upsizing and downsizing of the infrastructure available for applications.

u0055 • *Runtime management*. The runtime machinery is responsible for keeping the infrastructure up and running and serves as a hosting environment for services. It is primarily represented by the container and a collection of services that manage service membership and lookup, infrastructure maintenance, and profiling.

u0060 • *Resource management*. Aneka is an elastic infrastructure in which resources are added and removed dynamically according to application needs and user requirements. To provide

QoS-based execution, the system not only allows dynamic provisioning but also provides capabilities for reserving nodes for exclusive use by specific applications.

u0065 • *Application management.* A specific subset of services is devoted to managing applications. These services include scheduling, execution, monitoring, and storage management.

u0070 • *User management.* Aneka is a multitenant distributed environment in which multiple applications, potentially belonging to different users, are executed. The framework provides an extensible user system via which it is possible to define users, groups, and permissions. The services devoted to user management build up the security infrastructure of the system and constitute a fundamental element for accounting management.

u0075 • *QoS/SLA management and billing.* Within a cloud environment, application execution is metered and billed. Aneka provides a collection of services that coordinate together to take into account the usage of resources by each application and to bill the owning user accordingly.

p0120 All these services are available to specific interfaces and APIs on top of which the software development kit (SDK) and management kit are built. The SDK mainly relates to application development and modeling; it provides developers with APIs to develop applications with the existing programming models and an object model for creating new models. The management kit is mostly focused on interacting with the runtime services for managing the infrastructure, users, and applications. The management kit gives a complete view of Aneka Clouds and allows monitoring Aneka's status, whereas the SDK is more focused on the single application and provides means to control its execution from a single user. Both components are meant to provide an easy-to-use interface via which to interact and manage containers that are the core component of the Aneka framework.

## s0015 5.2 Anatomy of the Aneka container

p0125 The Aneka container constitutes the building blocks of Aneka Clouds and represents the runtime machinery available to services and applications. The container, the unit of deployment in Aneka Clouds, is a lightweight software layer designed to host services and interact with the underlying operating system and hardware. The main role of the container is to provide a lightweight environment in which to deploy services and some basic capabilities such as communication channels through which it interacts with other nodes in the Aneka Cloud. Almost all operations performed within Aneka are carried out by the services managed by the container. The services installed in the Aneka container can be classified into three major categories:

u0080 • Fabric Services

u0085 • Foundation Services

u0090 • Application Services

p0145 The services stack resides on top of the *Platform Abstraction Layer (PAL)*, representing the interface to the underlying operating system and hardware. It provides a uniform view of the software and hardware environment in which the container is running. Persistence and security traverse all the services stack to provide a secure and reliable infrastructure. In the following sections we discuss the components of these layers in more detail.

s0020 ### 5.2.1 From the ground up: the platform abstraction layer

p0150 The core infrastructure of the system is based on the .NET technology and allows the Aneka container to be portable over different platforms and operating systems. Any platform featuring an ECMA-334 [52] and ECMA-335 [53] compatible environment can host and run an instance of the Aneka container.

p0155 The *Common Language Infrastructure (CLI)*, which is the specification introduced in the ECMA-334 standard, defines a common runtime environment and application model for executing programs but does not provide any interface to access the hardware or to collect performance data from the hosting operating system. Moreover, each operating system has a different file system organization and stores that information differently. The *Platform Abstraction Layer (PAL)* addresses this heterogeneity and provides the container with a uniform interface for accessing the relevant hardware and operating system information, thus allowing the rest of the container to run unmodified on any supported platform.

p0160 The PAL is responsible for detecting the supported hosting environment and providing the corresponding implementation to interact with it to support the activity of the container. The PAL provides the following features:

u0095 • Uniform and platform-independent implementation interface for accessing the hosting platform
u0100 • Uniform access to extended and additional properties of the hosting platform
u0105 • Uniform and platform-independent access to remote nodes
u0110 • Uniform and platform-independent management interfaces

p0185 The PAL is a small layer of software that comprises a detection engine, which automatically configures the container at boot time, with the platform-specific component to access the above information and an implementation of the abstraction layer for the Windows, Linux, and Mac OS X operating systems.

p0190 The collectible data that are exposed by the PAL are the following:

u0115 • Number of cores, frequency, and CPU usage
u0120 • Memory size and usage
u0125 • Aggregate available disk space
u0130 • Network addresses and devices attached to the node

p0215 Moreover, additional custom information can be retrieved by querying the properties of the hardware. The PAL interface provides means for custom implementations to pull additional information by using name-value pairs that can host any kind of information about the hosting platform. For example, these properties can contain additional information about the processor, such as the model and family, or additional data about the process running the container.

s0025 ### 5.2.2 Fabric services

p0220 *Fabric Services* define the lowest level of the software stack representing the Aneka Container. They provide access to the resource-provisioning subsystem and to the monitoring facilities implemented in Aneka. Resource-provisioning services are in charge of dynamically providing new nodes on demand by relying on virtualization technologies, while monitoring services allow for hardware profiling and implement a basic monitoring infrastructure that can be used by all the services installed in the container.

s0030 #### 5.2.2.1 Profiling and monitoring

p0225 Profiling and monitoring services are mostly exposed through the *Heartbeat*, *Monitoring*, and *Reporting Services*. The first makes available the information that is collected through the PAL; the other two implement a generic infrastructure for monitoring the activity of any service in the Aneka Cloud.

p0230 The Heartbeat Service periodically collects the dynamic performance information about the node and publishes this information to the membership service in the Aneka Cloud. These data are collected by the index node of the Cloud, which makes them available for services such as reservations and scheduling in order to optimize the use of a heterogeneous infrastructure. As already discussed, basic information about memory, disk space, CPU, and operating system is collected. Moreover, additional data are pulled into the "alive" message, such as information about the installed software in the system and any other useful information. More precisely, the infrastructure has been designed to carry over any type of data that can be expressed by means of text-valued properties. As previously noted, the information published by the Heartbeat Service is mostly concerned with the properties of the node. A specific component, called *Node Resolver*, is in charge of collecting these data and making them available to the Heartbeat Service. Aneka provides different implementations for such component in order to cover a wide variety of hosting environments. A variety of operating systems are supported with different implementations of the PAL, and different node resolvers allow Aneka to capture other types of data that do not strictly depend on the hosting operating system. For example, the retrieval of the public IP of the node is different in the case of physical machines or virtual instances hosted in the infrastructure of an IaaS provider such as EC2 or GoGrid. In virtual deployment, a different node resolver is used so that all other components of the system can work transparently.

p0235 The set of built-in services for monitoring and profiling is completed by a generic monitoring infrastructure, which allows any custom service to report its activity. This infrastructure is composed of the Reporting and Monitoring Services. The *Reporting Service* manages the store for monitored data and makes them accessible to other services or external applications for analysis purposes. On each node, an instance of the *Monitoring Service* acts as a gateway to the *Reporting Service* and forwards to it all the monitored data that has been collected on the node. Any service that wants to publish monitoring data can leverage the local monitoring service without knowing the details of the entire infrastructure. Currently several built-in services provide information through this channel:

u0135 • The *Membership Catalogue* tracks the performance information of nodes.
u0140 • The *Execution Service* monitors several time intervals for the execution of jobs.
u0145 • The *Scheduling Service* tracks the state transitions of jobs.
u0150 • The *Storage Service* monitors and makes available information about data transfer, such as upload and download times, filenames, and sizes.
u0155 • The *Resource Provisioning Service* tracks the provisioning and lifetime information of virtual nodes.

p0265 All this information can be stored on a relational database management system (RDBMS) or a flat file and can be further analyzed by specific applications. For example, the Management Console provides a view on such data for administrative purposes.

s0035 #### 5.2.2.2 Resource management

p0270 Resource management is another fundamental feature of Aneka Clouds. It comprises several tasks: resource membership, resource reservation, and resource provisioning. Aneka provides a collection of services that are in charge of managing resources. These are the *Index Service* (or *Membership Catalogue*), *Reservation Service*, and *Resource Provisioning Service*.

p0275 The *Membership Catalogue* is Aneka's fundamental component for resource management; it keeps track of the basic node information for all the nodes that are connected or disconnected. The Membership Catalogue implements the basic services of a directory service, allowing the search for services using attributes such as names and nodes. During container startup, each instance publishes its information to the Membership Catalogue and updates it constantly during its lifetime. Services and external applications can query the membership catalogue to discover the available services and interact with them. To speed up and enhance the performance of queries, the membership catalogue is organized as a distributed database: All the queries that pertain to information maintained locally are resolved locally; otherwise the query is forwarded to the main index node, which has a global knowledge of the entire Cloud. The Membership Catalogue is also the collector of the dynamic performance data of each node, which are then sent to the local monitoring service to be persisted in the long term.

p0280 Indexing and categorizing resources are fundamental to resource management. On top of the basic indexing service, provisioning completes the set of features that are available for resource management within Aneka. Deployment of container instances and their configuration are performed by the infrastructure management layer and are not part of the Fabric Services.

p0285 Dynamic resource provisioning allows the integration and management of virtual resources leased from IaaS providers into the Aneka Cloud. This service changes the structure of the Aneka Cloud by allowing it to scale up and down according to different needs: handling node failures, ensuring the quality of service for applications, or maintaining a constant performance and throughput of the Cloud. Aneka defines a very flexible infrastructure for resource provisioning whereby it is possible to change the logic that triggers provisioning, support several back-ends, and change the runtime strategy with which a specific back-end is selected for provisioning. The resource-provisioning infrastructure built into Aneka is mainly concentrated in the *Resource Provisioning Service*, which includes all the operations that are needed for provisioning virtual instances. The implementation of the service is based on the idea of *resource pools*. A resource pool abstracts the interaction with a specific IaaS provider by exposing a common interface so that all the pools can be managed uniformly. A resource pool does not necessarily map to an IaaS provider but can be used to expose as dynamic resources a private cloud managed by a Xen Hypervisor or a collection of physical resources that are only used sporadically. The system uses an open protocol, allowing for the use of metadata to provide additional information for describing resource pools and to customize provisioning requests. This infrastructure simplifies the implementation of additional features and the support of different implementations that can be transparently integrated into the existing system.

p0290 Resource provisioning is a feature designed to support QoS requirements-driven execution of applications. Therefore, it mostly serves requests coming from the Reservation Service or the Scheduling Services. Despite this, external applications can directly leverage Aneka's resource-provisioning capabilities by dynamically retrieving a client to the service and interacting with the infrastructure to it. This extends the resource-provisioning scenarios that can be handled by Aneka, which can also be used as a virtual machine manager.

s0040 ### 5.2.3 Foundation services

p0295 Fabric Services are fundamental services of the Aneka Cloud and define the basic infrastructure management features of the system. *Foundation Services* are related to the logical management of the distributed system built on top of the infrastructure and provide supporting services for the execution of distributed applications. All the supported programming models can integrate with and leverage these services to provide advanced and comprehensive application management. These services cover:

u0160 • Storage management for applications
u0165 • Accounting, billing, and resource pricing
u0170 • Resource reservation

p0315 Foundation Services provide a uniform approach to managing distributed applications and allow developers to concentrate only on the logic that distinguishes a specific programming model from the others. Together with the Fabric Services, Foundation Services constitute the core of the Aneka middleware. These services are mostly consumed by the execution services and Management Consoles. External applications can leverage the exposed capabilities for providing advanced application management.

s0045 #### *5.2.3.1 Storage management*

p0320 Data management is an important aspect of any distributed system, even in computing clouds. Applications operate on data, which are mostly persisted and moved in the format of files. Hence, any infrastructure that supports the execution of distributed applications needs to provide facilities for file/data transfer management and persistent storage. Aneka offers two different facilities for storage management: a centralized file storage, which is mostly used for the execution of compute-intensive applications, and a distributed file system, which is more suitable for the execution of data-intensive applications. The requirements for the two types of applications are rather different. Compute-intensive applications mostly require powerful processors and do not have high demands in terms of storage, which in many cases is used to store small files that are easily transferred from one node to another. In this scenario, a centralized storage node, or a pool of storage nodes, can constitute an appropriate solution. In contrast, data-intensive applications are characterized by large data files (gigabytes or terabytes), and the processing power required by tasks does not constitute a performance bottleneck. In this scenario, a distributed file system harnessing the storage space of all the nodes belonging to the cloud might be a better and more scalable solution.

p0325 Centralized storage is implemented through and managed by Aneka's *Storage Service*. The service constitutes Aneka's data-staging facilities. It provides distributed applications with the basic file transfer facility and abstracts the use of a specific protocol to end users and other components of the system, which are dynamically configured at runtime according to the facilities installed in the cloud. The option that is currently installed by default is normal File Transfer Protocol (FTP).

p0330 To support different protocols, the system introduces the concept of a *file channel* that identifies a pair of components: a file channel controller and a file channel handler. The *file channel controller* constitutes the server component of the channel, where files are stored and made available; the *file channel handler* represents the client component, which is used by user applications or other components of the system to upload, download, or browse files. The storage service uses the

configured file channel factory to first create the server component that will manage the storage and then create the client component on demand. User applications that require support for file transfer are automatically configured with the appropriate file channel handler and transparently upload input files or download output files during application execution. In the same way, worker nodes are configured by the infrastructure to retrieve the required files for the execution of the jobs and to upload their results.

p0335 An interesting property of the file channel abstraction is the ability to chain two different channels to move files by using two different protocols. Each file in Aneka contains metadata that helps the infrastructure select the appropriate channel for moving the file. For example, an output file whose final location is an S3 bucket can be moved from the worker node to the Storage Service using the internal FTP protocol and then can be staged out on S3 by the FTP channel controller managed by the service. The Storage Service supports the execution of task-based programming such as the *Task* and the *Thread Model* as well as *Parameter Sweep*-based applications.

p0340 Storage support for data-intensive applications is provided by means of a distributed file system. The reference model for the distributed file system is the Google File System [54], which features a highly scalable infrastructure based on commodity hardware. The architecture of the file system is based on a master node, which contains a global map of the file system and keeps track of the status of all the storage nodes, and a pool of chunk servers, which provide distributed storage space in which to store files. Files are logically organized into a directory structure but are persisted on the file system using a flat namespace based on a unique ID. Each file is organized as a collection of *chunks* that are all of the same size. File chunks are assigned unique IDs and stored on different servers, eventually replicated to provide high availability and failure tolerance. The model proposed by the Google File System provides optimized support for a specific class of applications that expose the following characteristics:

u0175 • Files are huge by traditional standards (multi-gigabytes).
u0180 • Files are modified by appending new data rather than rewriting existing data.
u0185 • There are two kinds of major workloads: large streaming reads and small random reads.
u0190 • It is more important to have a sustained bandwidth than a low latency.

p0365 Moreover, given the huge number of commodity machines that the file system harnesses together, failure (process or hardware failure) is the norm rather than an exception. These characteristics strongly influenced the design of the storage, which provides the best performance for applications specifically designed to operate on data as described. Currently, the only programming model that makes use of the distributed file system is *MapReduce* [55], which has been the primary reason for the Google File System implementation. Aneka provides a simple distributed file system (DFS), which relies on the file system services of the Windows operating system.

#### s0050 5.2.3.2 Accounting, billing, and resource pricing

p0370 *Accounting Services* keep track of the status of applications in the Aneka Cloud. The collected information provides a detailed breakdown of the distributed infrastructure usage and is vital for the proper management of resources.

p0375 The information collected for accounting is primarily related to infrastructure usage and application execution. A complete history of application execution and storage as well as other resource

utilization parameters is captured and minded by the Accounting Services. This information constitutes the foundation on which users are charged in Aneka.

p0380 Billing is another important feature of accounting. Aneka is a multitenant cloud programming platform in which the execution of applications can involve provisioning additional resources from commercial IaaS providers. Aneka Billing Service provides detailed information about each user's usage of resources, with the associated costs. Each resource can be priced differently according to the set of services that are available on the corresponding Aneka container or the installed software in the node. The accounting model provides an integrated view of budget spent for each application, a summary view of the costs associated to a specific user, and the detailed information about the execution cost of each job.

p0385 The accounting capabilities are concentrated within the *Accounting Service* and the *Reporting Service*. The former keeps track of the information that is related to application execution, such as the distribution of jobs among the available resources, the timing of each of job, and the associated cost. The latter makes available the information collected from the monitoring services for accounting purposes: storage utilization and CPU performance. This information is primarily consumed by the Management Console.

#### s0055 5.2.3.3 Resource reservation

p0390 Aneka's *Resource Reservation* supports the execution of distributed applications and allows for reserving resources for exclusive use by specific applications. Resource reservation is built out of two different kinds of services: Resource Reservation and the Allocation Service. Resource Reservation keeps track of all the reserved time slots in the Aneka Cloud and provides a unified view of the system. The *Allocation Service* is installed on each node that features execution services and manages the database of information regarding the allocated slots on the local node. Applications that need to complete within a given deadline can make a reservation request for a specific number of nodes in a given timeframe. If it is possible to satisfy the request, the Reservation Service will return a reservation identifier as proof of the resource booking. During application execution, such an identifier is used to select the nodes that have been reserved, and they will be used to execute the application. On each reserved node, the execution services will check with the Allocation Service that each job has valid permissions to occupy the execution timeline by verifying the reservation identifier. Even though this is the general reference model for the reservation infrastructure, Aneka allows for different implementations of the service, which mostly vary in the protocol that is used to reserve resources or the parameters that can be specified while making a reservation request. Different protocol and strategies are integrated in a completely transparent manner, and Aneka provides extensible APIs for supporting advanced services. At the moment, the framework supports three different implementations:

u0195 • *Basic Reservation*. Features the basic capability to reserve execution slots on nodes and implements the *alternate offers* protocol, which provides alternative options in case the initial reservation requests cannot be satisfied.

u0200 • *Libra Reservation.* Represents a variation of the previous implementation that features the ability to price nodes differently according to their hardware capabilities.

u0205 • *Relay Reservation.* Constitutes a very thin implementation that allows a resource broker to reserve nodes in Aneka Clouds and control the logic with which these nodes are reserved. This

implementation is useful in integration scenarios in which Aneka operates in an intercloud environment.

p0410 Resource reservation is fundamental to ensuring the quality of service that is negotiated for applications. It allows Aneka to have a predictable environment in which applications can complete within the deadline or not be executed at all. The assumptions made by the reservation service for accepting reservation requests are based on the static allocation of such requests to the existing physical (or virtual) infrastructure available at the time of the requests and by taking into account the current and future load. This solution is sensitive to node failures that could make Aneka unable to fulfill the service-level agreement (SLA) made with users. Specific implementations of the service tend to delay the allocation of nodes to reservation requests as late as possible in order to cope with temporary failures or limited outages, but in the case of serious outages in which the remaining available nodes are not able to cover the demand, this strategy is not enough. In this case, resource provisioning can provide an effective solution: Additional nodes can be provisioned from external resource providers in order to cover the outage and meet the SLA defined for applications. The current implementation of the resource reservation infrastructure leverages the provisioning capabilities of the fabric layer when the current availability in the system is not able to address the reservation requests already confirmed. Such behavior solves the problems of both insufficient resources and temporary failures.

### s0060 5.2.4 Application services

p0415 *Application Services* manage the execution of applications and constitute a layer that differentiates according to the specific programming model used for developing distributed applications on top of Aneka. The types and the number of services that compose this layer for each of the programming models may vary according to the specific needs or features of the selected model. It is possible to identify two major types of activities that are common across all the supported models: scheduling and execution. Aneka defines a reference model for implementing the runtime support for programming models that abstracts these two activities in corresponding services: the *Scheduling Service* and the *Execution Service*. Moreover, it also defines base implementations that can be extended in order to integrate new models.

#### s0065 *5.2.4.1 Scheduling*

p0420 *Scheduling Services* are in charge of planning the execution of distributed applications on top of Aneka and governing the allocation of jobs composing an application to nodes. They also constitute the integration point with several other Foundation and Fabric Services, such as the Resource Provisioning Service, the Reservation Service, the Accounting Service, and the Reporting Service. Common tasks that are performed by the scheduling component are the following:

u0210 • Job to node mapping
u0215 • Rescheduling of failed jobs
u0220 • Job status monitoring
u0225 • Application status monitoring

p0445 Aneka does not provide a centralized scheduling engine, but each programming model features its own scheduling service that needs to work in synergy with the existing services of the middleware. As already mentioned, these services mostly belong to the fabric and the foundation layers of the architecture shown in Figure 5.2. The possibility of having different scheduling engines for different models gives great freedom in implementing scheduling and resource allocation strategies but, at the same time, requires a careful design of use of shared resources. In this scenario, common situations that have to be appropriately managed are the following: multiple jobs sent to the same node at the same time; jobs without reservations sent to reserved nodes; and jobs sent to nodes where the required services are not installed. Aneka's Foundation Services provide sufficient information to avoid these cases, but the runtime infrastructure does not feature specific policies to detect these conditions and provide corrective action. The current design philosophy in Aneka is to keep the scheduling engines completely separate from each other and to leverage existing services when needed. As a result, it is possible to enforce that only one job per programming model is run on each node at any given time, but the execution of applications is not mutually exclusive unless Resource Reservation is used.

s0070 #### 5.2.4.2 Execution

p0450 *Execution Services* control the execution of single jobs that compose applications. They are in charge of setting up the runtime environment hosting the execution of jobs. As happens for the scheduling services, each programming model has its own requirements, but it is possible to identify some common operations that apply across all the range of supported models:

u0230 • Unpacking the jobs received from the scheduler
u0235 • Retrieval of input files required for job execution
u0240 • Sandboxed execution of jobs
u0245 • Submission of output files at the end of execution
u0250 • Execution failure management (i.e., capturing sufficient contextual information useful to identify the nature of the failure)
u0255 • Performance monitoring
u0260 • Packing jobs and sending them back to the scheduler

p0490 Execution Services constitute a more self-contained unit with respect to the corresponding scheduling services. They handle less information and are required to integrate themselves only with the Storage Service and the local Allocation and Monitoring Services. Aneka provides a reference implementation of execution services that has built-in integration with all these services, and currently two of the supported programming models specialize on the reference implementation.

p0495 Application Services constitute the runtime support of the programming model in the Aneka Cloud. Currently there are several supported models:

u0265 • *Task Model*. This model provides the support for the independent "bag of tasks" applications and many computing tasks. In this model, an application is modeled as a collection of tasks that are independent from each other and whose execution can be sequenced in any order.
u0270 • *Thread Model*. This model provides an extension to the classical multithreaded programming to a distributed infrastructure and uses the abstraction of *Thread* to wrap a method that is executed remotely.

u0275 • *MapReduce Model*. This is an implementation of MapReduce as proposed by Google on top of Aneka.

u0280 • *Parameter Sweep Model*. This model is a specialization of the Task Model for applications that can be described by a template task whose instances are created by generating different combinations of parameters, which identify a specific point into the domain of interest.

p0520 Other programming models have been developed for internal use and are at an experimental stage. These are the Dataflow Model [56], the Message-Passing Interface, and the Actor Model [57].

## s0075 5.3 Building aneka clouds

p0525 Aneka is primarily a platform for developing distributed applications for clouds. As a software platform it requires infrastructure on which to be deployed; this infrastructure needs to be managed. Infrastructure management tools are specifically designed for this task, and building clouds is one of the primary tasks of administrators. Aneka supports various deployment models for public, private, and hybrid clouds.

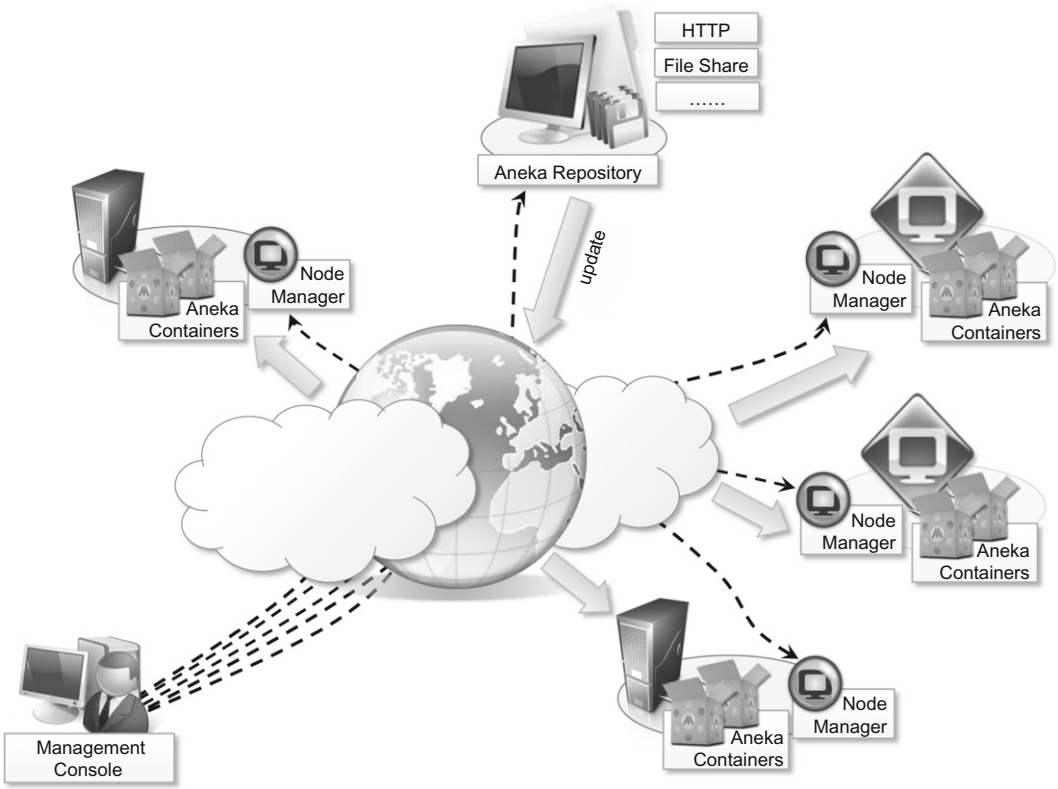
### s0080 5.3.1 Infrastructure organization

p0530 Figure 5.3 provides an overview of Aneka Clouds from an infrastructure point of view. The scenario is a reference model for all the different deployments Aneka supports. A central role is played by the Administrative Console, which performs all the required management operations. A fundamental element for Aneka Cloud deployment is constituted by *repositories*. A repository provides storage for all the libraries required to lay out and install the basic Aneka platform. These libraries constitute the software image for the node manager and the container programs. Repositories can make libraries available through a variety of communication channels, such as HTTP, FTP, common file sharing, and so on. The Management Console can manage multiple repositories and select the one that best suits the specific deployment. The infrastructure is deployed by harnessing a collection of nodes and installing on them the Aneka node manager, also called the *Aneka daemon*. The daemon constitutes the remote management service used to deploy and control container instances. The collection of resulting containers identifies the Aneka Cloud.

p0535 From an infrastructure point of view, the management of physical or virtual nodes is performed uniformly as long as it is possible to have an Internet connection and remote administrative access to the node. A different scenario is constituted by the dynamic provisioning of virtual instances; these are generally created by prepackaged images already containing an installation of Aneka, which only need to be configured to join a specific Aneka Cloud. It is also possible to simply install the container or install the Aneka daemon, and the selection of the proper solution mostly depends on the lifetime of virtual resources.

### s0085 5.3.2 Logical organization

p0540 The logical organization of Aneka Clouds can be very diverse, since it strongly depends on the configuration selected for each of the container instances belonging to the Cloud. The most common



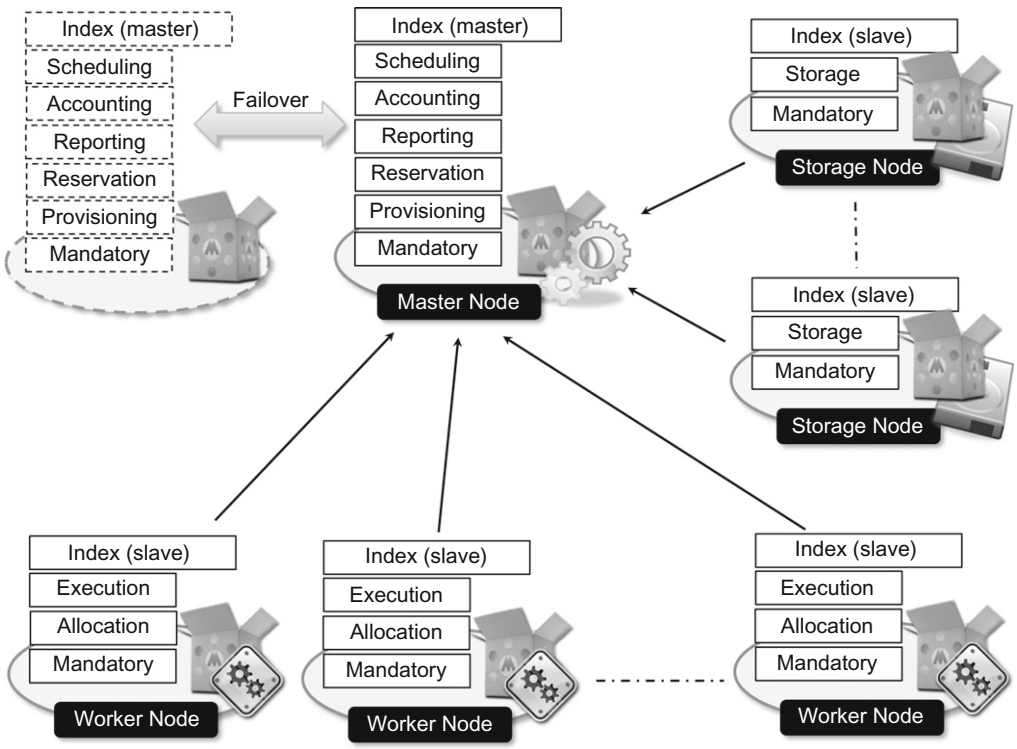

f0020 **FIGURE 5.3**

Aneka cloud infrastructure overview.

scenario is to use a master-worker configuration with separate nodes for storage, as shown in Figure 5.4.

p0545 The master node features all the services that are most likely to be present in one single copy and that provide the intelligence of the Aneka Cloud. What specifically characterizes a node as a master node is the presence of the *Index Service* (or Membership Catalogue) configured in master mode; all the other services, except for those that are mandatory, might be present or located in other nodes. A common configuration of the master node is as follows:

u0285 • Index Service (master copy)
u0290 • Heartbeat Service
u0295 • Logging Service
u0300 • Reservation Service
u0305 • Resource Provisioning Service
u0310 • Accounting Service
u0315 • Reporting and Monitoring Service
u0320 • Scheduling Services for the supported programming models




f0025 **FIGURE 5.4**

Logical organization of an Aneka cloud.

p0590 The master node also provides connection to an RDBMS facility where the state of several services is maintained. For the same reason, all the scheduling services are maintained in the master node. They share the application store that is normally persisted on the RDBMS in order to provide a fault-tolerant infrastructure. The master configuration can then be replicated in several nodes to provide a highly available infrastructure based on the failover mechanism.

p0595 The worker nodes constitute the workforce of the Aneka Cloud and are generally configured for the execution of applications. They feature the mandatory services and the specific execution services of each of the supported programming models in the Cloud. A very common configuration is the following:

u0325 • Index Service
u0330 • Heartbeat Service
u0335 • Logging Service
u0340 • Allocation Service
u0345 • Monitoring Service
u0350 • Execution Services for the supported programming models

p0630 A different option is to partition the pool of worker nodes with a different selection of execution services in order to balance the load between programming models and reserve some nodes for a specific class of applications.

p0635 Storage nodes are optimized to provide storage support to applications. They feature, among the mandatory and usual services, the presence of the Storage Service. The number of storage nodes strictly depends on the predicted workload and storage consumption of applications. Storage nodes mostly reside on machines that have considerable disk space to accommodate a large quantity of files. The common configuration of a storage node is the following:

u0355 • Index Service
u0360 • Heartbeat Service
u0365 • Logging Service
u0370 • Monitoring Service
u0375 • Storage Service

p0665 In specific cases, when the data transfer requirements are not demanding, there might be only one storage node. In some cases, for very small deployments, there is no need to have a separate storage node, and the Storage Service is installed and hosted on the master node.

p0670 All nodes are registered with the master node and transparently refer to any failover partner in the case of a high-availability configuration.

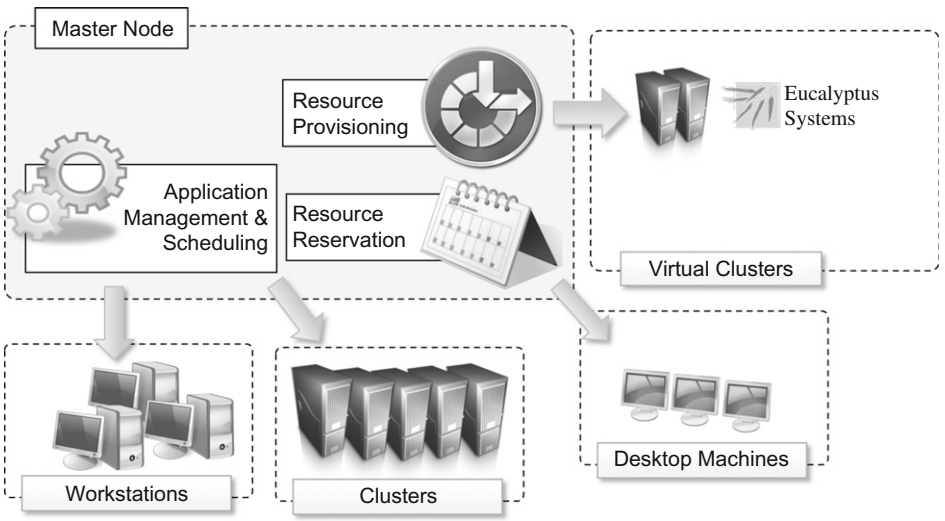
### s0090 5.3.3 Private cloud deployment mode

p0675 A private deployment mode is mostly constituted by local physical resources and infrastructure management software providing access to a local pool of nodes, which might be virtualized. In this scenario Aneka Clouds are created by harnessing a heterogeneous pool of resources such has desktop machines, clusters, or workstations. These resources can be partitioned into different groups, and Aneka can be configured to leverage these resources according to application needs. Moreover, leveraging the Resource Provisioning Service, it is possible to integrate virtual nodes provisioned from a local resource pool managed by systems such as XenServer, Eucalyptus, and OpenStack.

p0680 Figure 5.5 shows a common deployment for a private Aneka Cloud. This deployment is acceptable for a scenario in which the workload of the system is predictable and a local virtual machine manager can easily address excess capacity demand. Most of the Aneka nodes are constituted of physical nodes with a long lifetime and a static configuration and generally do not need to be reconfigured often. The different nature of the machines harnessed in a private environment allows for specific policies on resource management and usage that can be accomplished by means of the Reservation Service. For example, desktop machines that are used during the day for office automation can be exploited outside the standard working hours to execute distributed applications. Workstation clusters might have some specific legacy software that is required for supporting the execution of applications and should be preferred for the execution of applications with special requirements.

### s0095 5.3.4 Public cloud deployment mode

p0685 Public Cloud deployment mode features the installation of Aneka master and worker nodes over a completely virtualized infrastructure that is hosted on the infrastructure of one or more resource



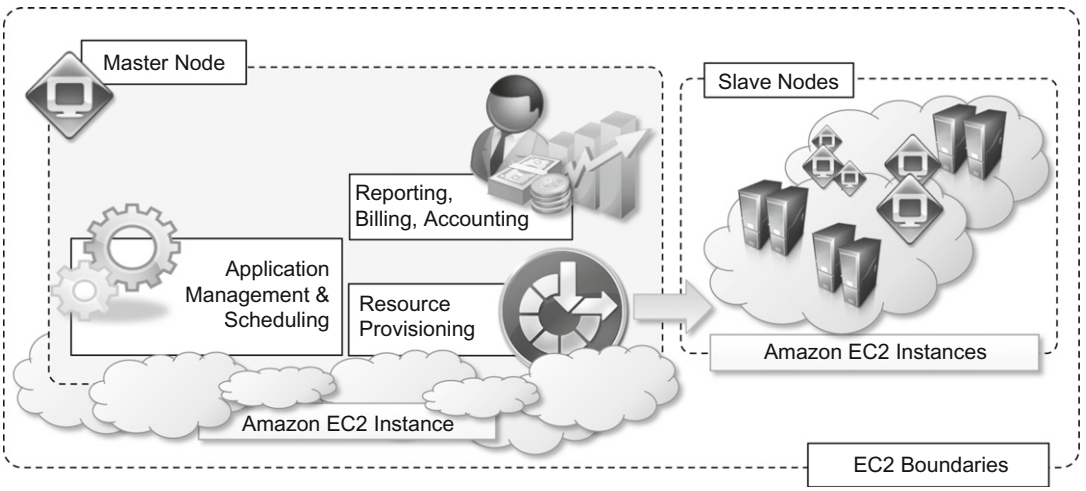

f0030 **FIGURE 5.5**

Private cloud deployment.

providers such as Amazon EC2 or GoGrid. In this case it is possible to have a static deployment where the nodes are provisioned beforehand and used as though they were real machines. This deployment merely replicates a classic Aneka installation on a physical infrastructure without any dynamic provisioning capability. More interesting is the use of the elastic features of IaaS providers and the creation of a Cloud that is completely dynamic. Figure 5.6 provides an overview of this scenario.

p0690 The deployment is generally contained within the infrastructure boundaries of a single IaaS provider. The reasons for this are to minimize the data transfer between different providers, which is generally priced at a higher cost, and to have better network performance. In this scenario it is possible to deploy an Aneka Cloud composed of only one node and to completely leverage dynamic provisioning to elastically scale the infrastructure on demand. A fundamental role is played by the Resource Provisioning Service, which can be configured with different images and templates to instantiate. Other important services that have to be included in the master node are the Accounting and Reporting Services. These provide details about resource utilization by users and applications and are fundamental in a multitenant Cloud where users are billed according to their consumption of Cloud capabilities.

p0695 Dynamic instances provisioned on demand will mostly be configured as worker nodes, and, in the specific case of Amazon EC2, different images featuring a different hardware setup can be made available to instantiate worker containers. Applications with specific requirements for computing capacity or memory can provide additional information to the scheduler that will trigger the appropriate provisioning request. Application execution is not the only use of dynamic instances; any service requiring elastic scaling can leverage dynamic provisioning. Another example is the Storage Service. In multitenant Clouds, multiple applications can leverage the support for storage;




f0035 **FIGURE 5.6**

Public Aneka cloud deployment.

in this scenario it is then possible to introduce bottlenecks or simply reach the quota limits allocated for storage on the node. Dynamic provisioning can easily solve this issue as it does for increasing the computing capability of an Aneka Cloud.

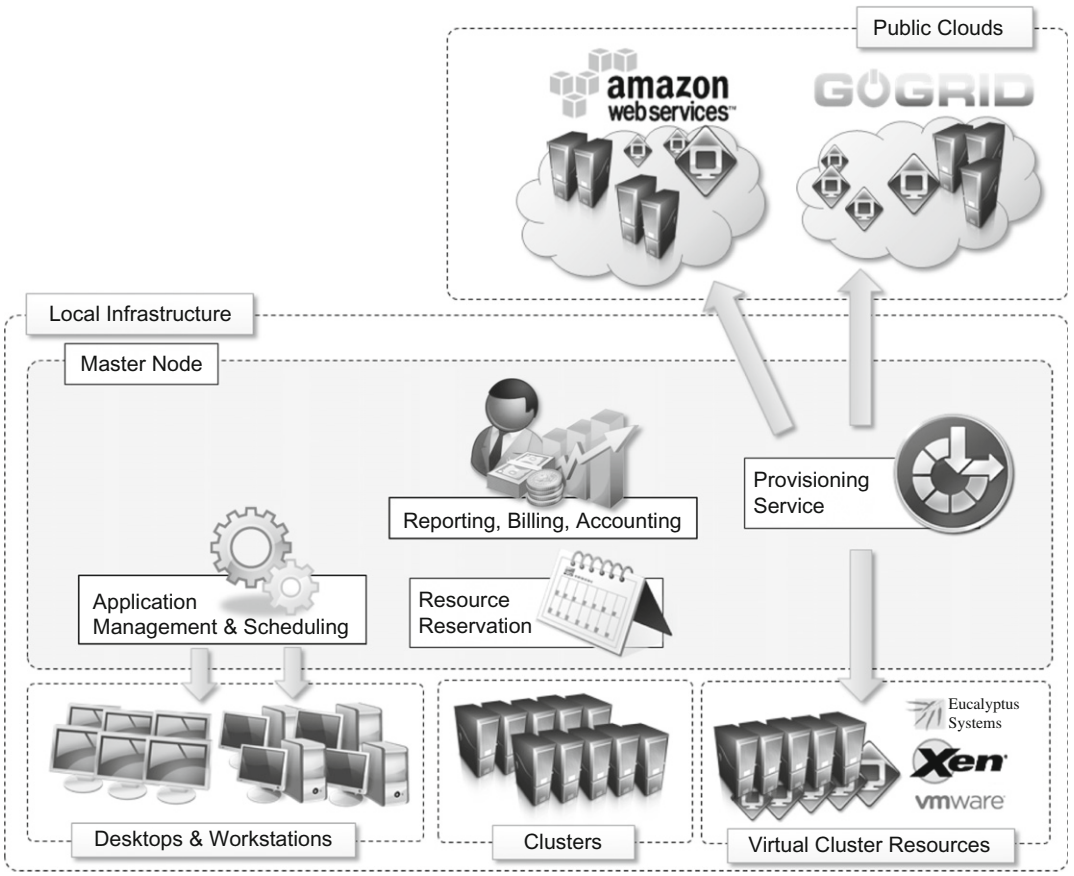
p0700 Deployments using different providers are unlikely to happen because of the data transfer costs among providers, but they might be a possible scenario for federated Aneka Clouds [58]. In this scenario resources can be shared or leased among providers under specific agreements and more convenient prices. In this case the specific policies installed in the Resource Provisioning Service can discriminate among different resource providers, mapping different IaaS providers to provide the best solution to a provisioning request.

### s0100 5.3.5 Hybrid cloud deployment mode

p0705 The hybrid deployment model constitutes the most common deployment of Aneka. In many cases, there is an existing computing infrastructure that can be leveraged to address the computing needs of applications. This infrastructure will constitute the static deployment of Aneka that can be elastically scaled on demand when additional resources are required. An overview of this deployment is presented in Figure 5.7.

p0710 This scenario constitutes the most complete deployment for Aneka that is able to leverage all the capabilities of the framework:

u0380 • Dynamic Resource Provisioning
u0385 • Resource Reservation
u0390 • Workload Partitioning
u0395 • Accounting, Monitoring, and Reporting




f0040 **FIGURE 5.7**

Hybrid cloud deployment.

p0735 Moreover, if the local premises offer some virtual machine management capabilities, it is possible to provide a very efficient use of resources, thus minimizing the expenditure for application execution.

p0740 In a hybrid scenario, heterogeneous resources can be used for different purposes. As we discussed in the case of a private cloud deployment, desktop machines can be reserved for low priority workload outside the common working hours. The majority of the applications will be executed on workstations and clusters, which are the nodes that are constantly connected to the Aneka Cloud. Any additional computing capability demand can be primarily addressed by the local virtualization facilities, and if more computing power is required, it is possible to leverage external IaaS providers.

p0745 Different from the Aneka Public Cloud deployment is the case in which it makes more sense to leverage a variety of resource providers to provision virtual resources. Since part of the infrastructure is local, a cost in data transfer to the external IaaS infrastructure cannot be avoided. It is then important to select the most suitable option to address application needs. The Resource Provisioning

Service implemented in Aneka exposes the capability of leveraging several resource pools at the same time and configuring specific policies to select the most appropriate pool for satisfying a provisioning request. These features simplify the development of custom policies that can better serve the needs of a specific hybrid deployment.

## 5.4 Cloud programming and management

p0750 Aneka's primary purpose is to provide a scalable middleware product in which to execute distributed applications. Application development and management constitute the two major features that are exposed to developers and system administrators. To simplify these activities, Aneka provides developers with a comprehensive and extensible set of APIs and administrators with powerful and intuitive management tools. The APIs for development are mostly concentrated in the Aneka SDK; management tools are exposed through the Management Console.

### 5.4.1 Aneka SDK

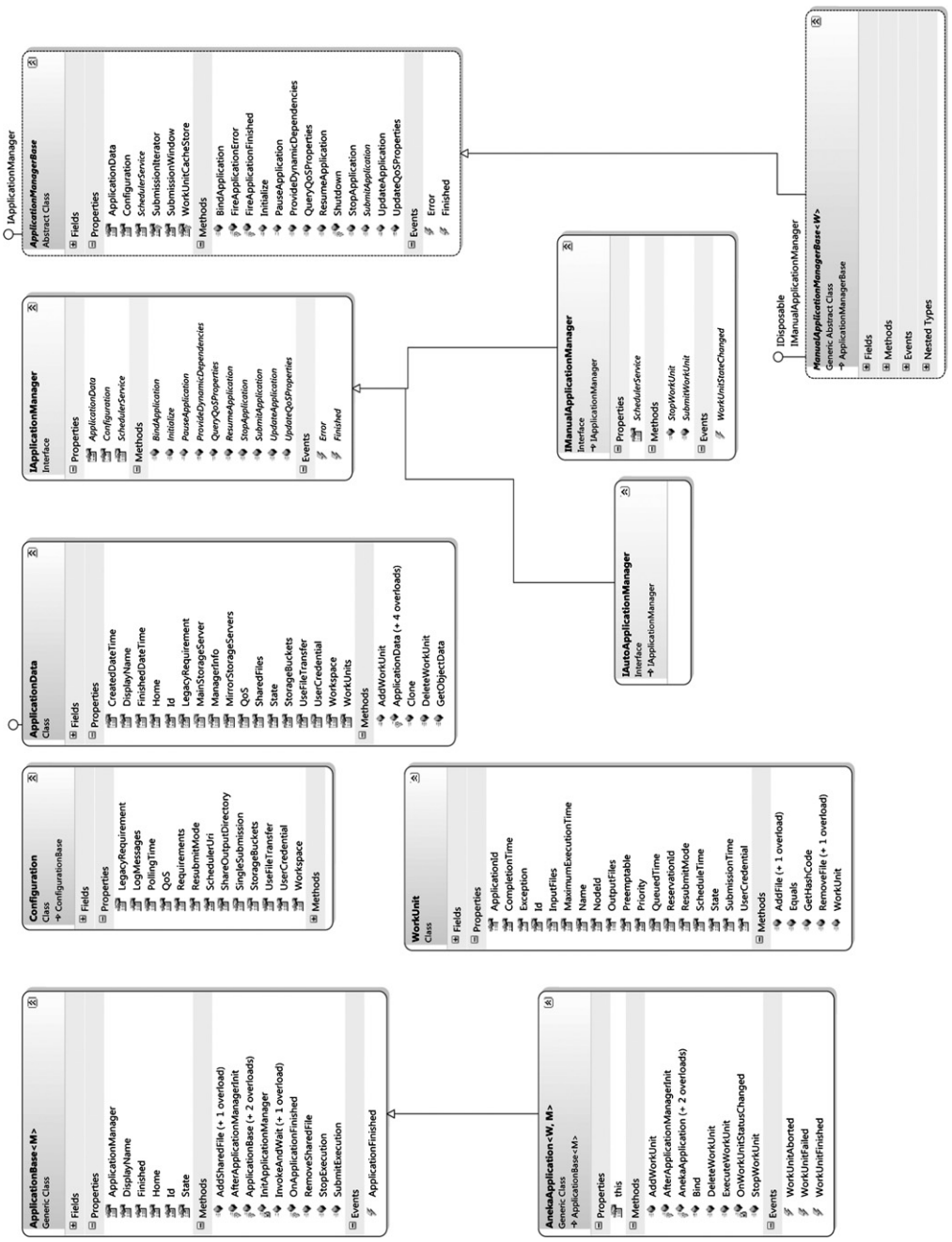
p0755 Aneka provides APIs for developing applications on top of existing programming models, implementing new programming models, and developing new services to integrate into the Aneka Cloud. The development of applications mostly focuses on the use of existing features and leveraging the services of the middleware, while the implementation of new programming models or new services enriches the features of Aneka. The SDK provides support for both programming models and services by means of the *Application Model* and the *Service Model*. The former covers the development of applications and new programming models; the latter defines the general infrastructure for service development.

#### 5.4.1.1 Application model

p0760 Aneka provides support for distributed execution in the Cloud with the abstraction of programming models. A programming model identifies both the abstraction used by the developers and the runtime support for the execution of programs on top of Aneka. The *Application Model* represents the minimum set of APIs that is common to all the programming models for representing and programming distributed applications on top of Aneka. This model is further specialized according to the needs and the particular features of each of the programming models.

p0765 An overview of the components that define the Aneka Application Model is shown in Figure 5.8. Each distributed application running on top of Aneka is an instance of the *ApplicationBase* $< M >$ class, where *M* identifies the specific type of application manager used to control the application. Application classes constitute the developers' view of a distributed application on Aneka Clouds, whereas application managers are internal components that interact with Aneka Clouds in order to monitor and control the execution of the application. Application managers are also the first element of specialization of the model and vary according to the specific programming model used.

p0770 Whichever the specific model used, a distributed application can be conceived as a set of tasks for which the collective execution defines the execution of the application on the Cloud. Aneka further specializes applications into two main categories: (1) applications whose tasks are generated




f0045 **FIGURE 5.8**

The Aneka application model.

by the user and (2) applications whose tasks are generated by the runtime infrastructure. These two categories generally correspond to different application base classes and different implementations of the application manager.

p0775 The first category is the most common and it is used as a reference for several programming models supported by Aneka: the *Task Model*, the *Thread Model*, and the *Parameter Sweep Model*. Applications that fall into this category are composed of a collection of units of work submitted by the user and represented by the *WorkUnit* class. Each unit of work can have input and output files, the transfer of which is transparently managed by the runtime. The specific type of *WorkUnit* class used to represent the unit of work depends on the programming model used (*AnekaTask* for the *Task Model* and *AnekaThread* for the *Thread Model*). All the applications that fall into this category inherit or are instances *of AnekaApplication* $< W,M >$, where $W$ is the specific type of *WorkUnit* class used, and $M$ is the type of application manager used to implement the *IManualApplicationManager* interface.

p0780 The second category covers the case of *MapReduce* and all those other scenarios in which the units of work are generated by the runtime infrastructure rather than the user. In this case there is no common unit-of-work class used, and the specific classes used by application developers strictly depend on the requirements of the programming model used. For example, in the case of the *MapReduce* programming model, developers express their distributed applications in terms of two functions, *map* and *reduce*; hence, the *MapReduceApplication* class provides an interface for specifying the *Mapper* $< K,V >$ and *Reducer* $< K,V >$ types and the input files required by the application. Other programming models might have different requirements and expose different interfaces. For this reason there are no common base types for this category except for *ApplicationBase* $< M >$, where $M$ implements *IAutoApplicationManager*.

p0785 A set of additional classes completes the object model. Among these classes, the most notable are the *Configuration* class, which is used to specify the settings required to initialize the application and customize its behavior, and the *ApplicationData* class, which contains the runtime information of the application.

p0790 Table 5.1 summarizes the features that are available in the Aneka Application Model and the way they reflect into the supported programming model. The model has been designed to be extensible, and these classes can be used as a starting point to implement a new programming model. This can be done by augmenting the features (or specializing) an existing implementation of a

t0010 **Table 5.1** Aneka's Application Model Features

| Category | Description | Base Application Type | Work Units? | Programming Models |
|---|---|---|---|---|
| Manual | Units of work are generated by the user and submitted through the application. | *AnekaApplication* $< W,M >$<br>*IManualApplicationManager* $< W >$<br>*ManualApplicationManager* $< W >$ | Yes | Task Model<br>Thread Model<br>Parameter Sweep Model |
| Auto | Units of work are generated by the runtime infrastructure and managed internally. | *ApplicationBase* $< M >$<br>*IAutoApplicationManager* | No | *MapReduce* Model |

programming model or by using the base classes to define new models and abstractions. For example, the Parameter Sweep Model is a specialization of the Task Model, and it has been implemented in the context of management of applications on Aneka. It is achieved by providing a different interface to end users who just need to define a template task and the parameters that customize it.

s0120 #### *5.4.1.2 Service model*

p0795 The Aneka *Service Model* defines the basic requirements to implement a service that can be hosted in an Aneka Cloud. The container defines the runtime environment in which services are hosted. Each service that is hosted in the container must be compliant with the *IService* interface, which exposes the following methods and properties:

u0400 • Name and status
u0405 • Control operations such as *Start*, *Stop*, *Pause*, and *Continue* methods
u0410 • Message handling by means of the *HandleMessage* method

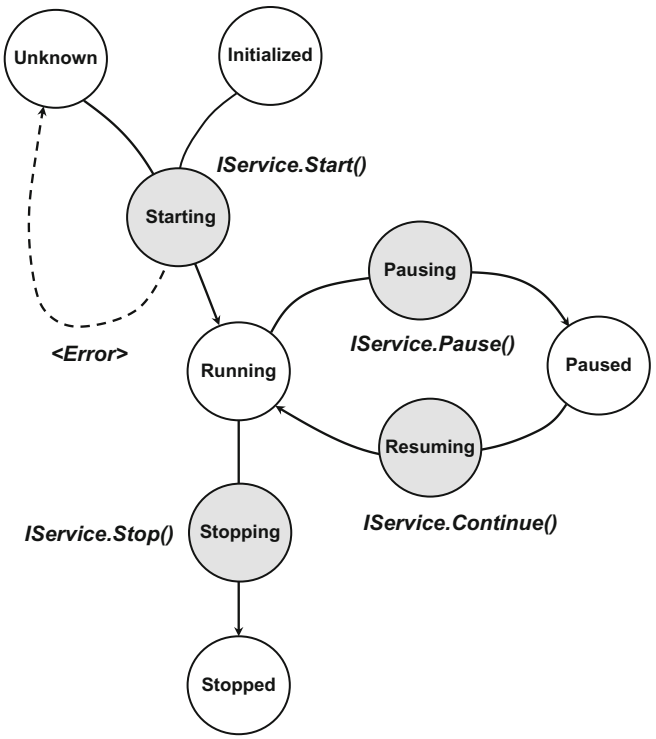
p0815 Specific services can also provide clients if they are meant to directly interact with end users. Examples of such services might be Resource Provisioning and Resource Reservation Services, which ship their own clients for allowing resource provisioning and reservation. Apart from control operations, which are used by the container to set up and shut down the service during the container life cycle, the core logic of a service resides in its message-processing functionalities that are contained in the *HandleMessage* method. Each operation that is requested to a service is triggered by a specific message, and results are communicated back to the caller by means of messages.

p0820 Figure 5.9 describes the reference life cycle of each service instance in the Aneka container. The shaded balloons indicate transient states; the white balloons indicate steady states. A service instance can initially be in the *Unknown* or *Initialized* state, a condition that refers to the creation of the service instance by invoking its constructor during the configuration of the container. Once the container is started, it will iteratively call the *Start* method on each service method. As a result the service instance is expected to be in a *Starting* state until the startup process is completed, after which it will exhibit the *Running* state. This is the condition in which the service will last as long as the container is active and running. This is the only state in which the service is able to process messages. If an exception occurs while starting the service, it is expected that the service will fall back to the *Unknown* state, thus signaling an error.

p0825 When a service is running it is possible to pause its activity by calling the *Pause* method and resume it by calling *Continue*. As described in the figure, the service moves first into the *Pausing* state, thus reaching the *Paused* state. From this state, it moves into the *Resuming* state while restoring its activity to return to the *Running* state. Not all the services need to support the pause/continue operations, and the current implementation of the framework does not feature any service with these capabilities.

p0830 When the container shuts down, the *Stop* method is iteratively called on each service running, and services move first into the transient *Stopping* state to reach the final *Stopped* state, where all resources that were initially allocated have been released.

p0835 Aneka provides a default base class for simplifying service implementation and a set of guidelines that service developers should follow to design and implement services that are compliant with Aneka. In particular, the guidelines define a *ServiceBase* class that can be further extended to




f0050 **FIGURE 5.9**

Service life cycle.

provide a proper implementation. This class is the base class of several services in the framework and provides some built-in features:

u0415 • Implementation of the basic properties exposed by *IService*
u0420 • Implementation of the control operations with logging capabilities and state control
u0425 • Built-in infrastructure for delivering a service specific client
u0430 • Support for service monitoring

p0860 Developers are provided with template methods for specializing the behavior of control operations, implementing their own message-processing logic, and providing a service-specific client.

p0865 Aneka uses a strongly typed message-passing communication model, whereby each service defines its own messages, which are in turn the only ones that the service is able to process. As a result, developers who implement new services in Aneka need also to define the type of messages that the services will use to communicate with services and clients. Each message type inherits from the base class *Message* defining common properties such as:

u0435 • Source node and target node
u0440 • Source service and target service

u0445 • Security credentials

p0885 Additional properties are added to carry the specific information for each type. Messages are generally used inside the Aneka infrastructure. In case the service exposes features directly used by applications, they may expose a service client that provides an object-oriented interface to the operations exposed by the service. Aneka features a ready-to-use infrastructure for dynamically injecting service clients into applications by querying the middleware. Services inheriting from the *ServiceBase* class already support such a feature and only need to define an interface and a specific implementation for the service client. Service clients are useful to integrate Aneka services into existing applications that do not necessarily need support for the execution of distributed applications or require access to additional services.

p0890 Aneka also provides advanced capabilities for service configuration. Developers can define editors and configuration classes that allow Aneka's management tools to integrate the configuration of services within the common workflow required by the container configuration.

## s0125 5.4.2 Management tools

p0895 Aneka is a pure PaaS implementation and requires virtual or physical hardware to be deployed. Hence, infrastructure management, together with facilities for installing logical clouds on such infrastructure, is a fundamental feature of Aneka's management layer. This layer also includes capabilities for managing services and applications running in the Aneka Cloud.

### s0130 *5.4.2.1 Infrastructure management*

p0900 Aneka leverages virtual and physical hardware in order to deploy Aneka Clouds. Virtual hardware is generally managed by means of the Resource Provisioning Service, which acquires resources on demand according to the need of applications, while physical hardware is directly managed by the Administrative Console by leveraging the Aneka management API of the PAL. The management features are mostly concerned with the provisioning of physical hardware and the remote installation of Aneka on the hardware.

### s0135 *5.4.2.2 Platform management*

p0905 Infrastructure management provides the basic layer on top of which Aneka Clouds are deployed. The creation of Clouds is orchestrated by deploying a collection of services on the physical infrastructure that allows the installation and the management of containers. A collection of connected containers defines the platform on top of which applications are executed. The features available for platform management are mostly concerned with the logical organization and structure of Aneka Clouds. It is possible to partition the available hardware into several Clouds variably configured for different purposes. Services implement the core features of Aneka Clouds and the management layer exposes operations for some of them, such as Cloud monitoring, resource provisioning and reservation, user management, and application profiling.

### s0140 *5.4.2.3 Application management*

p0910 Applications identify the user contribution to the Cloud. The management APIs provide administrators with monitoring and profiling features that help them track the usage of resources and relate them to users and applications. This is an important feature in a cloud computing scenario in which

users are billed for their resource usage. Aneka exposes capabilities for giving summary and detailed information about application execution and resource utilization.

p0915 All these features are made accessible through the Aneka Cloud Management Studio, which constitutes the main Administrative Console for the Cloud.

s0145 ## SUMMARY

p0920 In this chapter we introduced Aneka, a platform for application programming in the cloud. Aneka is a pure PaaS implementation of the Cloud Computing Reference Model and constitutes a middleware product that enables the creation of computing clouds on top of heterogeneous hardware: desktop machines, clusters, and public virtual resources.

p0925 One of the key aspects of Aneka's framework is its configurable runtime environment, which allows for the creation of a service-based middleware where applications are executed. A fundamental element of the infrastructure is the container, which represents the deployment unit of Aneka Clouds. The container hosts a collection of services that define the capabilities of the middleware. Fundamental services in the Aneka middleware are:

u0450 • Fabric Services for monitoring, resource provisioning, hardware profiling, and membership
u0455 • Foundation Services for storage, resource reservation, billing, accounting, and reporting
u0460 • Application Services for scheduling and execution

p0945 From an application programming point of view, Aneka provides the capability of supporting different programming models, thus allowing developers to express distributed applications with different abstractions. The framework currently supports three different models: independent "bag of tasks" applications, multithreaded applications, and *MapReduce*.

p0950 The infrastructure is extensible, and Aneka provides both an application model and a service model that can be easily extended to integrate new services and programming models.

s0150 ## Review questions

o0010 **1.** Describe in a few words the main characteristics of Aneka.
o0015 **2.** What is the Aneka container and what is its use?
o0020 **3.** Which types of services are hosted inside the Aneka container?
o0025 **4.** Describe Aneka's resource-provisioning capabilities.
o0030 **5.** Describe the storage architecture implemented in Aneka.
o0035 **6.** What is a programming model?
o0040 **7.** List the programming models supported by Aneka.
o0045 **8.** Which are the components that compose the Aneka infrastructure?
o0050 **9.** Discuss the logical organization of an Aneka Cloud.
o0055 **10.** Which services are hosted in a worker node?
o0060 **11.** Discuss the private deployment of Aneka Clouds.
o0065 **12.** Discuss the public deployment of Aneka Clouds.

o0070 **13.** Discuss the role of dynamic provisioning in hybrid deployments.
o0075 **14.** Which facilities does Aneka provide for development?
o0080 **15.** Discuss the major features of the Aneka Application Model.
o0085 **16.** Discuss the major features of the Aneka Service Model.
o0090 **17.** Describe the features of the Aneka management tools in terms of infrastructure, platform, and applications.

**NON-PRINT ITEM**

**Abstract**

This chapter provides a complete overview of the cloud application framework by first describing the architecture of the system. It introduces Aneka's components and the fundamental services that make up the Aneka Cloud and discusses some common deployment scenarios.

**Keywords**

Platform-as-a-Service (PaaS), Pure PaaS, middleware, programming models, Aneka, Cloud Application Platform