

**WEB SERVICE PATTERNS FOR GLOBALLY DISTRIBUTED
SERVICE FABRIC
BACKGROUND**

[0001] Web-based applications include files that are located on web servers along
5 with data that is stored in databases. For example, there are a large number of servers
located within different networks to handle the traffic that is directed to the service. These
networks often have unreliable communications as well as computers and software that are
unreliable. Managing the operations of the online service that includes a large number of
servers is a time consuming process that requires a large operations staff that is subject to
10 human error.

SUMMARY

[0002] This Summary is provided to introduce a selection of concepts in a simplified
form that are further described below in the Detailed Description. This Summary is not
intended to identify key features or essential features of the claimed subject matter, nor is
15 it intended to be used as an aid in determining the scope of the claimed subject matter.

[0003] An idempotent and asynchronous application programming interface (API)
that can not rely on a reliable network is used by a cloud manager to receive and process
requests. The cloud manager system is a central coordination service that receives
requests using the API to perform update operations and get operations relating to the
20 online service. For example, the API includes methods for deploying machines, updating
machines, removing machines, performing configuration changes on servers, Virtual
Machines (VMs), as well as performing other tasks relating to the management of the
online service. Receiving and processing a same API call multiple times results in a same
result.

25 **BRIEF DESCRIPTION OF THE DRAWINGS**

[0004] FIGURE 1 illustrates a cloud manager system for managing networks that are
associated with an online service, such as a content management service;

[0005] FIGURE 2 shows a cloud manager including managers and associated
databases;

30 [0006] FIGURE 3 shows an exemplary job record stored within a row of a database;

[0007] FIGURE 4 shows an example system for a network including front-end and
back-end servers for an online service;

[0008] FIGURE 5 illustrates a computer architecture for a computer;

[0009] FIGURE 6 shows a cloud manager for an online service that receives requests from a web service API; and

[0010] FIGURE 7 shows a process for using a Web Service API for managing and deploying machines in an online service.

5

DETAILED DESCRIPTION

[0011] Referring now to the drawings, in which like numerals represent like elements, various embodiment will be described.

[0012] Generally, program modules include routines, programs, components, data structures, and other types of structures that perform particular tasks or implement particular abstract data types. Other computer system configurations may also be used, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, minicomputers, mainframe computers, and the like. Distributed computing environments may also be used where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

[0013] FIGURE 1 illustrates a cloud management system for managing networks that are associated with an online service. System 100 illustrates cloud manager 105 that is connected to and manages different networks potentially distributed across the world. Each of the networks is configured to provide content services for one or more tenants (e.g. clients, customers). The networks may be hosted within a cloud service and/or in an on-premises data center. Cloud manager 105 is used in deploying, configuring and managing the networks. The cloud manager is configured to receive requests through an idempotent and asynchronous application web service application programming interface (API) that can tolerate intermittent network failures.

[0014] As illustrated, cloud manager 105 comprises work manager 110, machine manager 115, application specific manager 120, scripts 130 and a central repository, such as data store(s) 140 (e.g. databases). The functionality that is not included within one of the illustrated managers may reside in some other location of the cloud manager. According to one embodiment, application manager 120 is a SharePoint tenant manager that comprises SharePoint specific logic.

[0015] Work manager 110 manages the execution of tasks and enables scheduling and retry of longer running tasks. Work manager 110 starts jobs stored in job queue 112 and keeps track of running jobs. When a predetermined time has elapsed, work manager

110 may automatically cancel the task and perform some further processing relating to the task. According to one embodiment, the tasks in job queue 112 are executed by work manager 110 by invoking one or more scripts 130. For example, a scripting language such as Microsoft's PowerShell® may be used to program the tasks that are executed by work manager 110. Each script may be run as a new process. While executing each script as a new process may have a fairly high CPU overhead, this system is scalable and helps to ensure a clean environment for each script execution plus full cleanup when the script is completed.

[0016] Machine manager 115 is configured to manage the physical machines in the networks (e.g. Network 1, Network 2, Network 3). Generally, machine manager 115 understands Networks, Physical Machines, Virtual Machines (VMs), VM Images (VHDs), and the like. The machine manager does not have a strong binding to the specific services running within the networks but keeps track of the various components in the networks in terms of "roles." For example machine manager 115 could be requested through API 150 to deploy a VM of type "Foo" with version 12.34.56.78 on Network 3. In response to a request to cloud manager 105, machine manager 115 locates a suitable Physical Machine that is located on Network 3 and configures the VM according to the VM Image associated with the VM's Role. The physical machine is configured with a VHD of type Foo with version 12.34.56.78 that is stored within a data store, such as data store 140. The images used within the network may also be stored in other locations, such as a local data share for one or more of the networks. Scripts may be run to perform the installation of the VHD on the physical machine as well as for performing any post-deployment configuration. Machine manager 115 keeps track of the configuration of the machines each network. For example, machine manager 115 may keep track of a VM's role (type of VM), state of the VM (Provisioning, Running, Stopped, Failed), version and whether the VM exists in a given farm (which implies their network).

[0017] Scripts 130 is configured to store scripts that are executed to perform work both locally for cloud manager 105 and remotely on one or more of the networks. One or more of the scripts 130 may also be stored in other locations. For example, scripts to be performed on a network (e.g. Network 1, Network 2, Network 3) may be stored locally to that network. The scripts may be used for many different purposes. For example, the scripts may be used to perform configurations of machines in one or more of the networks, changing settings on previously configured machines, add a new VM, add a new database, move data from one machine to another, move tenants, change schemas, and the like.

According to one embodiment, the scripts are Microsoft's PowerShell® scripts. Other programming implementations may be used. For example, a compiled and/or early-bound programming language may be used to implement the functionality. Scripting, however, is a fairly concise language to express many of the tasks that are to be performed.

5 Programming the equivalent in a programming language, such as C#, would often require much more verbose implementations. The scripts are also late-bound, meaning that multiple versions of underlying code-bases can be targeted without having to constantly link to different interface DLLs. Using PowerShell scripts allows a process to be started locally by cloud manager 105 that may in turn start a process on a remote machine (i.e. a
10 physical machine in one of the attached networks). Other techniques may also be used to start a process on a remote machine, such as Secure Shell (SSH) and the like.

[0018] Application specific information that cloud manager 105 is managing is performed by application manager 120. According to one embodiment, the application specific information relates to Microsoft SharePoint®. As such, application manager 120
15 is configured to know about SharePoint Tenants, Site Collections, and the like.

[0019] Each network may be configured as a dedicated network for a tenant and/or as a multi-tenant network that services more than one client. The networks may include a changing number of physical/virtual machines with their configuration also changing after deployment. Generally, a network may continue to grow as long as the networking limits
20 (e.g. load balancer and network switches) are not exceeded. For example, a network may start out with ten servers and later expand to one hundred or more servers. The physical machines within a network may be assigned a class or type. For example, some of the machines may be compute machines (used for web front ends and app servers) and other machines may be storage machines that are provisioned with more storage than compute
25 machines. According to an embodiment, cloud manager 105 configures the machines within a network with multiple versions of the image files. According to an embodiment, farms usually have a same version of image files.

[0020] According to one embodiment, the software limits are managed by the cloud manager system 100 within the network by virtualizing the machines and managing
30 independently acting "Farms" inside the network. Each network may include one or more farms (e.g. see Network 1). According to one embodiment, a network is considered a single cluster of network load balanced machines that expose one or more VIP (Virtual IP) to the outside world and can route that traffic to any of the machines within the network.

The machines in the network generally are tightly coupled and have minimum latencies (i.e. < 1ms ping latency).

5 **[0021]** Farms are the basic grouping of machines used to coordinate applications that need tightly bound relationships. For example, content farms may be deployed within each of the networks for a content management application, such as Microsoft SharePoint®. Generally, the set of machines in each of the farms provide web service and application server functions together. Typically, the machines inside the farm are running the same build of an application (i.e. SharePoint) and are sharing a common configuration database to serve specific tenants and site collections.

10 **[0022]** Farms can contain heterogeneous sets of virtual machines. Cloud manager 105 maintains a “farm goal” within data store 140 which is a target number of machines of each role for each farm. Some roles include Content Front End, Content Central Admin, Content Timer Service, Federated Central Admin, Federated App Server etc. For example, content farms are the basic SharePoint farm that handles incoming customer requests. Federated Services farms contain SharePoint services that can operate cross farms such as search and the profile store. Farms may be used for hosting large capacity public internet sites. Some farms may contain a group of Active Directory servers and a Provisioning Daemon. Cloud manager 105 automatically deploys and/or decommissions virtual machines in the networks to help in meeting the defined target. These farms goals may be automatically and/or manually configured. For example, the farm goals may change to respond to changes in activity and capacity needs. Network Farm- there is one network farm per Network that contains all the VM roles that scale out easily as a resource to the whole Network.

25 **[0023]** The Cloud Manager Web Service APIs 150 are designed to work in the context of a massively scalable global service. The APIs assume that any network request might fail and/or hang in transit. Calls to cloud manager 105 are configured to be idempotent. In other words, the same call may be made to cloud manager 105 multiple times (as long as the parameters are identical) without changing the outcome.

30 **[0024]** Cloud manager 105 is designed to do very little processing (<10ms, <50ms) before returning a response to any given request. Cloud manager 105 maintains records to keep track of current requests. For example, cloud manager 105 updates records in a local database and if necessary schedules a “job” to perform more lengthy activity later.

[0025] Cloud manager keeps track of Images (such as Virtual Disk Images) that are the templates used to deploy new machines within a network. The Image references

may be stored in a database, such as database 140, and/or in some other location. The images may be stored in one or more shared data stores that are local to the network(s) on which the image will be deployed. According to one embodiment, each Image includes a virtual machine (VM) role type that specifies the type of VM it can deploy, the number of
5 processors that it should use, the amount of RAM that it will be assigned, a network ID used to find a nearby install point (so they don't get copied repeatedly over the cross data-center links) and a share path that the deployment code can use to access the VHD.

[0026] Generally, machines in the networks being managed by cloud system 100 are not upgraded in the traditional manner by downloading data and incorporating the data
10 into the existing software on the machine. Instead, machines are updated by replacing a VHD with an updated VHD. For example, when a new version of software is needed by a farm, a new farm is deployed that has the new version installed. When the new farm is deployed, the tenants are moved from the old farm to the new farm. In this way, downtime due to an upgrade is minimized and each machine in the farm has a same
15 version that have been tested. When a virtual machine needs to be upgraded, the VM on the machine may be deleted and replaced with the VM that is configured to run the desired service.

[0027] While upgrades to existing software are not optimal, some servers within the networks do utilize the traditional update procedure of an in-place upgrade. For
20 example, Active Directory Domain Controllers are upgraded by updating the current software on the server without completely replacing an image on the machine. The cloud manager may also be upgraded in place in some instances.

[0028] FIGURE 2 shows a cloud manager including managers and associated databases. As illustrated, cloud manager 200 comprises work manager 210, work database
25 215, machine manager 220, machine database 225, tenant manager 230, tenant database 235, secrets database 245 and web service APIs 240.

[0029] Generally, databases used within a cloud management system (e.g. system 100) are sized to enable high performance. For example, a database (such as work
30 database 215, machine database 225, tenant database 235 and secrets database 245) may not exceed a predefined size limit (e.g. 30 GB, 50 GB, 100 GB, and the like). According to an embodiment, a database is sized such that it is small enough to fit in-memory of a physical machine. This assists in high read I/O performance. The size of the database may also be selected based on performance with an application program, such as interactions with a SQL server. The databases used in the farms may also be sized to enable high

performance. For example, they may be sized to fit in-memory of the host machine and/or sized such that backup operations, move operations, copy operations, restore operations are generally performed within a predetermined period of time.

[0030] Cloud manager 200 divides the cloud manager data into four databases.
5 The work database 215 for the work manager. The machine database 225 for the machine manager 220. The tenant database 235 for the tenant manager 230 and a secrets database 245 for storing sensitive information such as system account and password information, credentials, certificates, and the like. The databases may be on the same server and or split across servers. According to an embodiment, each database is mirrored for high
10 availability and is a SQL database.

[0031] Cloud manager 200 is configured to interact with the databases using a reduced set of SQL features in order to assist in providing availability of the cloud manager 200 during upgrades of the databases. For example, foreign keys or stored procedures are attempted to be avoided. Foreign keys can make schema changes difficult
15 and cause unanticipated failure conditions. Stored procedures place more of the application in the database itself.

[0032] Communications with the SQL servers are attempted to be minimized since roundtrips can be expensive compared to the cost of the underlying operation. For example, it is usually much more efficient if all of the current SQL server interactions to a
20 single database are wrapped in a single round-trip.

[0033] Constraints are rarely used within the databases (215, 225, 235). Generally, constraints are useful when it helps provide simple updates with the right kind of error handling without extra queries. For example, the fully qualified domain name (FQDN) table has a constraint placed on the "name" to assist in preventing a tenant from
25 accidentally trying to claim the same FQDN as is already allocated to a different tenant.

[0034] Caution is used when adding indices. Indices typically improve read performance at the cost of extra I/Os for write operations. Since the data within the databases is primarily RAM resident, even full table scans are relatively fast. According to an embodiment, indices may be added once the query patterns have stabilized and a
30 performance improvement may be determined by proposed indices. According to an embodiment, if adding the index will potentially take a long time the "ONLINE=ON" option may be specified such that the table isn't locked while the index is initially built.

[0035] According to an embodiment, upgrades to databases within the cloud manager may be performed without causing downtime to the cloud manager system. In

other words, even during an upgrade of the cloud manager, the cloud manager continues processing received requests. As such, changes made to the schema are to be compatible with the previous schema. The SQL schema upgrade is run before the web servers used by the cloud manager are upgraded. When the web servers are upgraded they can start to
5 use the new features enabled in the database. Database upgrades are limited such that operations involved in the upgrade are quick and efficient. For example, tables may be added and new nullable columns may be added to existing columns. New columns may be added at the end of a table. Generally, time consuming operations to the databases are avoided. For example, adding a default value to a newly added column at creation time
10 may be a very time consuming operation when there is a large amount of data. Adding a nullable column, however, is a very quick operation. As discussed above, adding new indices are allowed, but caution should be taken when adding a new constraint to help ensure sure that the schema upgrade won't break with the existing data. For example, when a constraint is added it may be set to a state that is not checked and avoids a costly
15 validation of existing rows and potential errors. Old tables and unused columns are removed after a new version is being used and the cloud manager is not accessing those tables and columns.

[0036] Generally, a single row in each of the databases is used to indicate a task and/or a desired state. For example, the tenant database 235 includes a single row for each
20 tenant. A given tenant may include a Required Version record. This record is used to help ensure that the tenant is placed on a farm running the required version. For example, for tenant 1 to stay on SharePoint 14 SP1, the required version for tenant could be set to "14.1." and any version including 14.1 would match and any other versions (e.g. 14.2.xxxx) would not match. The tenant records may include other items such as
25 authorized number of users, quotas (e.g. allowed total data usage, per user data usage, etc.), time restrictions, and the like. Some organization might have multiple tenants that represent different geographies, organizations or capabilities. According to an embodiment, tenants are walled off from each other without explicit invitation of the users (via extranet or other features).

30 [0037] According to one embodiment, each tenant is locked into a specific network. Tenants are kept localized to a small set of databases. A tenant is either small (smaller than would fill one database) in which case it is in exactly one database, shared with other tenants. This implies that all the tenants sharing that database need to upgrade at the same time. When a tenant grows larger it may be moved to its own dedicated

database(s) and now might have more than one, but is not sharing databases with other tenants. Maintaining a large tenant in one or more dedicated databases helps in reducing a number of databases that are needed to be upgraded simultaneously in a single upgrade.

[0038] Similarly, the work database 215 includes a single row for each job. The machine database 225 may include a row for each physical machine, VM, farm, and the like. For example, machine manager database 225 may include a version string. According to an embodiment, each VHD, Farm, and VM within a network has an associated version string.

[0039] According to one embodiment, the cloud manager includes a simple logging system that may be configured to record a log entry for each web service call. A logging system may be implemented that includes as few/many features as desired. Generally, the logging system is used for measuring usage and performance profiling.

[0040] According to an embodiment, the Web Service APIs 240 are built using SOAP with ASP.net. The various Web Methods in the APIs follow two main patterns- Gets and Updates. Generally, the update methods take a data structure as the input and return the same structure as the output. The output structure returns the current state of the underlying object in the database, potentially differing from the input object if validation or other business logic changed some properties or else with additional properties filled in (for example record IDs or other values calculated by the cloud manager). The update methods are used for initial object creation as well as subsequent updates. In other words, callers to the web service APIs 240 can simply request the configuration they want and they don't need to keep track of whether the object already exists or not. In addition this means that updates are idempotent in that the same update call can be made twice with the identical effect to making it only once. According to an embodiment, an update method may include a LastUpdated property. When the LastUpdated property is present, the cloud manager 200 rejects the Update if the value of LastUpdate does not match the one currently stored in the database. Some Update methods include properties that are set on the first invocation of the method and are not set on other invocations of the method.

[0041] Cloud manager 200 is configured to avoid the use of callbacks. Since callbacks may be unreliable, clients interacting with cloud manager 200 may check object status using a web service API when they want to check a status of an update. According to an embodiment, a call to an update method causes cloud manager 200 to set the state of the underlying object to "Provisioning" and when the updates are completed the state is set to "Active".

[0042] FIGURE 3 shows an exemplary job record stored within a row of a database. As illustrated, record 300 comprises job identifier 302, type 304, data 306, owner 308, step 310, last run 312, expire time 314, next time 316, state 318 and status 320.

[0043] Generally, for each task that is requested to be performed, the cloud manager creates a record in database 350 (e.g. work database 215 in FIGURE 2).

[0044] Job identifier 302 is used to specify a unique identifier for the requested task.

[0045] Type 304 specifies the task to perform. For example, the type may include a name of the script to be executed. For example, when the task is to run the script named "DeployVM.ps1" then the data 306 may include the identifier (e.g. "-VMID 123"). This allows new task types to be added to the system without requiring any changes to compiled or other binary parts of the system.

[0046] Data 306 is used to store data that is associated with the task. For example, the data may be set to the tenant, machine, network, VM, etc. on which the task is to be performed. The data 306 may also store one or more values to which a value in a database is set. The process running the task may look to the job record to see what value the desired number of machines is set to. The script uses the value in the database to perform the operation.

[0047] Owner 308 specifies a process/machine that is executing the process. For example, when a cloud manager machine starts execution of a job, the machine updates the owner 308 portion of the record with an ID of the machine.

[0048] Step 310 provides an indication of a step of the current script. For example, the script may divide a task into any number of steps. As the process completes a step of the script, step 310 is updated. A process may also look at step 310 to determine what step to execute in the script and to avoid having to re-execute previously completed steps.

[0049] Last run 312 provides a time the script was last started. Each time a script is started, the last run time is updated.

[0050] Expire time 314 is a time that indicates when the process should be terminated. According to an embodiment, the expire time is a predetermined amount of time (e.g. five minutes, ten minutes...) after the process is started. The expire time may be updated by a requesting process through the web service API.

[0051] Next time 316 is a time that indicates when a task should next be executed. For example, a process may be stopped after completion of a step and be instructed to wait until the specified next time 316 to resume processing.

[0052] State 318 indicates a current state and Status 320 indicates a status of a job
5 (e.g. Created, Suspended, Resumed, Executing, Deleted).

[0053] Duplicate rows in the database can be removed before they are performed if they have the same task type and data values. For example, multiple requests may be made to perform the same task that are stored in multiple rows of the database.

[0054] A job can have one or more locks 355 associated with it. If locks are not
10 available then a job will not be scheduled to run until the locks are available. The locks may be configured in many different ways. For example, the locks may be based on a mutex, a semaphore, and the like. Generally, a mutex prevents code from being executed concurrently by more than one thread and a semaphore restricts a number of simultaneous uses of a shared resource up to a maximum number. According to an embodiment, a lock
15 is a character string that represents a resource. The resource may be any type of resource. For example, the lock may be a farm, a machine, a tenant, and the like. Generally, the locks are used to defer execution of one or more tasks. Each job may specify one or more locks that it needs before running. A job may release a lock at any time during its operation. When there is a lock, the job is not scheduled. A job needing more than one
20 lock requests all locks required at once. For example, a job already in possession of a lock may not request additional locks. Such a scheme assists in preventing possible deadlock situations caused by circular lock dependencies amongst multiple jobs.

[0055] FIGURE 4 shows an example system 400 for a network including front-end and back-end servers for an online service. The example system 400 includes clients 402
25 and 404, network 406, load balancer 408, WFE servers 410, 412, 414 and back-end servers 416-419. Greater or fewer clients, WFEs, back-end servers, load balancers and networks can be used. Additionally, some of the functionality provided by the components in system 400 may be performed by other components. For example, some load balancing may be performed in the WFEs.

[0056] In example embodiments, clients 402 and 404 are computing devices, such
30 as desktop computers, laptop computers, terminal computers, personal data assistants, or cellular telephone devices. Clients 402 and 404 can include input/output devices, a central processing unit ("CPU"), a data storage device, and a network device. In the present application, the terms client and client computer are used interchangeably.

[0057] WFEs 410, 412 and 414 are accessible to clients 402 and 404 via load balancer 408 through network 406. As discussed, the servers may be configured in farms. Back-end server 416 is accessible to WFEs 410, 412 and 414. Load balancer 408 is a dedicated network device and/or one or more server computers. Load balancer 408, 420, 5 WFEs 410, 412 and 414 and back-end server 416 can include input/output devices, a central processing unit ("CPU"), a data storage device, and a network device. In example embodiments, network 406 is the Internet and clients 402 and 404 can access WFEs 410, 412 and 414 and resources connected to WFEs 410, 412 and 414 remotely.

[0058] In an example embodiment, system 400 is an online, browser-based 10 document collaboration system. An example of an online, browser-based document collaboration system is Microsoft Sharepoint® from Microsoft Corporation of Redmond, Washington. In system 400, one or more of the back-end servers 416-419 are SQL servers, for example SQL Server from Microsoft Corporation of Redmond, Washington.

[0059] WFEs 410, 412 and 414 provide an interface between clients 402 and 404 15 and back-end servers 416-419. The load balancers 408, 420 direct requests from clients 402 and 404 to WFEs 410, 412 and 414 and from WFEs to back-end servers 416-419. The load balancer 408 uses factors such as WFE utilization, the number of connections to a WFE and overall WFE performance to determine which WFE server receives a client request. Similarly, the load balancer 420 uses factors such as back-end server utilization, 20 the number of connections to a server and overall performance to determine which back-end server receives a request.

[0060] An example of a client request may be to access a document stored on one of the back-end servers, to edit a document stored on a back-end server (e.g. 416-419) or to store a document on back-end server. When load balancer 408 receives a client request 25 over network 406, load balancer 408 determines which one of WFE server 410, 412 and 414 receives the client request. Similarly, load balancer 420 determines which one of the back-end servers 416-419 receive a request from the WFE servers. The back-end servers may be configured to store data for one or more tenants (i.e. customer).

[0061] Referring now to FIGURE 5, an illustrative computer architecture for a 30 computer 500 utilized in the various embodiments will be described. The computer architecture shown in FIGURE 5 may be configured as a server, a desktop or mobile computer and includes a central processing unit 5 ("CPU"), a system memory 7, including a random access memory 9 ("RAM") and a read-only memory ("ROM") 11, and a system bus 12 that couples the memory to the central processing unit ("CPU") 5.

[0062] A basic input/output system containing the basic routines that help to transfer information between elements within the computer, such as during startup, is stored in the ROM 11. The computer 500 further includes a mass storage device 14 for storing an operating system 16, application programs 10, data store 24, files, and a cloud
5 program 26 relating to execution of and interaction with the cloud system 100.

[0063] The mass storage device 14 is connected to the CPU 5 through a mass storage controller (not shown) connected to the bus 12. The mass storage device 14 and its associated computer-readable media provide non-volatile storage for the computer 500. Although the description of computer-readable media contained herein refers to a mass
10 storage device, such as a hard disk or CD-ROM drive, the computer-readable media can be any available media that can be accessed by the computer 100.

[0064] By way of example, and not limitation, computer-readable media may comprise computer storage media and communication media. Computer storage media includes volatile and non-volatile, removable and non-removable media implemented in
15 any method or technology for storage of information such as computer-readable instructions, data structures, program modules or other data. Computer storage media includes, but is not limited to, RAM, ROM, Erasable Programmable Read Only Memory (“EPROM”), Electrically Erasable Programmable Read Only Memory (“EEPROM”), flash memory or other solid state memory technology, CD-ROM, digital versatile disks
20 (“DVD”), or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by the computer 500.

[0065] According to various embodiments, computer 500 may operate in a networked environment using logical connections to remote computers through a network
25 18, such as the Internet. The computer 500 may connect to the network 18 through a network interface unit 20 connected to the bus 12. The network connection may be wireless and/or wired. The network interface unit 20 may also be utilized to connect to other types of networks and remote computer systems. The computer 500 may also include an input/output controller 22 for receiving and processing input from a number of
30 other devices, including a keyboard, mouse, or electronic stylus (not shown in FIGURE 5). Similarly, an input/output controller 22 may provide output to a display screen 28, a printer, or other type of output device.

[0066] As mentioned briefly above, a number of program modules and data files may be stored in the mass storage device 14 and RAM 9 of the computer 500, including an

operating system 16 suitable for controlling the operation of a networked computer, such as the WINDOWS® operating systems from MICROSOFT® CORPORATION of Redmond, Washington. The mass storage device 14 and RAM 9 may also store one or more program modules. In particular, the mass storage device 14 and the RAM 9 may
5 store one or more application programs, such as cloud program 26, that perform tasks relating to the cloud system.

[0067] FIGURE 6 shows a cloud manager for an online service that receives requests from a web service API.

[0068] Cloud manager 605 is used in deploying, configuring and managing the
10 networks for the online service. The cloud manager is configured to receive requests through an idempotent and asynchronous application web service application programming interface (API) 650 that can not rely on a reliable network.

[0069] As illustrated, cloud manager 605 comprises work manager 110, machine manager 115, application specific manager 120, scripts 130, databases 640, scripts 130,
15 images 641 and web service APIs 650. According to one embodiment, application manager 620 is a SharePoint tenant manager that comprises SharePoint specific logic.

[0070] Requests using APIs 650 may be used in the management and the deployment of servers in various topologies across different networks (Network 1, Network 2). While only two networks are shown, many more networks are generally managed (e.g. ten, one
20 hundred, one thousand, ten thousand, and the like). Cloud manager 605 operates and is configured similarly to the cloud manager system shown and described above. The web service APIs 650 includes methods to request services from work manager 110, machine manager 115 and application manager 120. For example, requests may be made using APIs 650 to update a tenant in a database, add a new SQL server, deploy a new farm, add
25 a new machine, update a VM, obtain values within a data store, and the like.

[0071] The Web Service APIs 650 are designed to work in the context of a scalable global service. As network requests are assumed to be inherently unreliable, the APIs assume that any network request might fail and/or hang in transit. Requests using the Web Service APIs 650 are configured to be idempotent. In other words, the same call with the
30 same parameters may be made utilizing the Web Service APIs 650 without changing the outcome.

[0072] Requests to the Web Service APIs 650 operate in an asynchronous manner. A requestor (e.g. requestor 602, or some other requestor in one of the networks) can repeatedly use the Web Service APIs 650 to call every few seconds to see if the request

was fulfilled. A request may also timeout after being submitted to cloud manager 605. The request may automatically be resubmitted and/or a requestor may continue to use the same method in the API until the request is satisfied.

5 [0073] According to an embodiment, the web methods in the Web Service APIs 650 follow two patterns that include get methods and update methods. The get methods are designed to retrieve something, such as a value, an object, and the like that relates to the online service. Update methods are used to act on an object within the online system and may involve one or more different processes that include one or more jobs being submitted to the work manager 110 for execution.

10 [0074] The update methods are used for initial object creation as well as subsequent updates. For example, an update farm method is used to both initially deploy a farm as well as to upgrade the farm. Using only update methods for creation and updates allows a requestor to simply request a configuration without having to determine whether the configuration already exists.

15 [0075] The update methods include a parameter, such as a data structure, as the input and returns the same structure to the requestor as the output. The output structure returns the current state of the underlying object in the database. To assist in protection against some forms of conflicts and race conditions, some update methods may include a last updated property. The last updated property may be used to assist in optimistic
20 concurrency control.

 [0076] Optimistic concurrency is generally used in environments with a low contention for data. This improves performance as no locking of records is required, and locking of records requires additional server resources. Also, in order to maintain record locks, a persistent connection to the database server is required. Because this is not the
25 case in an optimistic concurrency model, connections to the server are free to serve a larger number of clients in less time. In an optimistic concurrency model, a violation is considered to have occurred if, after a user receives a value from the database, another user modifies the value before the first user has attempted to modify it. When a non-null value is passed as the last updated property, cloud manager 605 rejects the update if the
30 value of the last updated property does not match the one in the database, signifying that some other actor has updated some of the values in the meantime.

 [0077] Some update methods have properties that are only set on the first invocation of the update method. Subsequent calls using the same update method with the same

parameters may check to see if the initial properties have already been set to determine when the update method has already been called.

[0078] Cloud manager 605 is designed to do very little processing (<10ms, <50ms) before returning a response to any given request. Cloud manager 605 maintains records to
5 keep track of currently requests. For example, cloud manager 605 updates records in a local database and if necessary schedules a “job” to perform more lengthy activity later. Once the parameters and job information are committed to the database, the response is sent to the requestor.

[0079] According to an embodiment, the Web Service APIs 650 are built using
10 SOAP with ASP.net. The various Web Methods in the APIs follow two main patterns- Gets and Updates.

[0080] Cloud manager 605 is configured to avoid the use of callbacks. Since callbacks may be unreliable, clients interacting with cloud manager 605 may check object status using a web service API when they want to check a status of an update method.
15 According to an embodiment, a call to an update method causes cloud manager 605 to set the state of the underlying object to “Provisioning” and when the updates are completed the state is set to “Active”.

[0081] Images 641 are configured to store Virtual Hard Disk (VHD) images that are in use and/or are to be deployed on one or more of the machines in one or more of the
20 networks. According to an embodiment, the MICROSOFT® VHD file format is used that specifies a virtual machine hard disk that can reside on a native host file system encapsulated within a single file. Generally, the VHD format is broadly applicable since it is agnostic to the virtualization technology, host operating system, or guest operating system with which it is used. Images that are used within a specific network may be
25 moved to a global share 645 and/or to a network share that is local to a network (e.g. network share 655). Storing the images on a network share saves time in a deployment of images since network communication time is reduced. An update method may be used to update the images within images 641 and/or in one of the global/network shares.

[0082] FIGURE 7 shows a process for using a Web Service API for managing and
30 deploying machines in an online service will be described.

[0083] When reading the discussion of the routines presented herein, it should be appreciated that the logical operations of various embodiments are implemented (1) as a sequence of computer implemented acts or program modules running on a computing system and/or (2) as interconnected machine logic circuits or circuit modules within the

computing system. The implementation is a matter of choice dependent on the performance requirements of the computing system implementing the invention. Accordingly, the logical operations illustrated and making up the embodiments described herein are referred to variously as operations, structural devices, acts or modules. These operations, structural devices, acts and modules may be implemented in software, in
5 operations, structural devices, acts and modules may be implemented in software, in firmware, in special purpose digital logic, and any combination thereof.

[0084] After a start operation, the process 700 flows to operation 710, where a request to perform a task is received using a Web Service API that is asynchronous and idempotent. The request may be to deploy a farm, add a machine to a farm, install a VM
10 on a machine, and the like. For example, a request is received through the API to update a machine (e.g. UpdateMachine(...)). UpdateMachine is used to both add a machine or update a configuration of the machine. The method includes parameters that specify a name and a network and location in which to update the machine.

[0085] Moving to operation 715, a determination is made as to whether this request
15 is the first request received through the API with the received parameters. As mentioned above, some methods in the API may be configured to set additional parameters when it is the first time the method is invoked.

[0086] When it is not the first call to the method, the process moves to operation 725.

[0087] When the method has not previously been called with the same parameters,
20 the process moves to operation 720.

[0088] At operation 720, one or more first time parameters are set. According to an embodiment, the parameters that are set are stored within a database. For example, a name of a machine may be set within the database when the request is to update a machine. Many other parameters may be set.

[0089] At operation 725, a database with the requested task and corresponding
25 information is updated. For example, the machine database may be updated to reflect a new machine within a network and set the desired values for the machine (including a state update to "provisioning" and the work database may be updated with the requested job information.

[0090] Transitioning to operation 730, the job is scheduled. At some point after the
30 job is entered within a job queue, the job is started by the work manager. For example, as soon as any jobs ahead of the job in the job queue have started, the job is started. According to an embodiment, the job is started by executing a script. Executing the script may invoke a remote call to start a script and/or perform some other task on a remote

machine. The remote machine may be located on a different network and/or different data center. According to an embodiment, the job does not contain the actual data to be changed but just a reference to an object in the database that has the real data.

5 [0091] Moving to operation 735, a response is returned to the requestor. A response to a request is returned when the action and any associated database settings are committed to the database. In this way, a requestor is not required to wait for a long time before receiving a response. The response returns a current state of the object as defined within the database. For example, when an update machine method is called, the object is a machine object.

10 [0092] At decision operation 740, the process that executes the job checks the state to determine whether the job has previously been attempted and failed. For example, a job may have failed part way through a previous attempt to perform the task. When the job has not been previously

15 [0093] When the job has previously been attempted, the operation moves to operation 750 where any clean up operations may be performed. The clean up operations relate to operations to repair any effects of previously attempting to perform the operation. According to an embodiment, the scripts determine the operations to perform and include the logic to determine the steps to perform. For example, one or more scripts may determine the operations that need to be performed based on a state of the database.

20 [0094] At decision operation 760, a determination is made as to whether the job has already been executed and completed. For example, when the state of the task is set to active, then an instance of the task has already been run. If the task has already run, the current task exits and moves to an end block.

25 [0095] When the job has not run, the process moves to operation 770 and the job is performed. After the job is completed, the process moves to operation 780 where the job information is updated. For instance, in the current example, once the machine has been configured, its state is set to Active. The job is also removed from the job queue. The job may be removed automatically after an expiration of a period of time and/or by a process calling the cloud manager specifying to remove the job.

30 [0096] The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.

WHAT IS CLAIMED IS:

1. A method for providing an Application Programming Interface (API) for use in an online service for requesting actions to be performed, comprising:
 - receiving a request through an idempotent and asynchronous application programming interface (API) at a central service to perform a task in an online service comprising different networks; wherein the request relates to at least one of: configuring, updating and managing machines in at least one of the different networks that are used in providing the online service;
 - updating a database with one or more values relating to the task, wherein the one or more values are a desired state of the task;
 - adding a job to a job queue for performing the task;
 - returning a response to a requestor of the request that is returned independently of the job being executed; and
 - executing the job, wherein executing the job comprises a process that is assigned to execute the task access the one or more values relating to the task from the database to determine the desired state of the task.
2. The method of Claim 1, wherein receiving a request multiple times for a same method in the API having the same parameters results in a same outcome as calling the same method one time.
3. The method of Claim 2, wherein the methods are web methods that are exposed through a web-based API that consists of get methods and update methods, wherein the update methods are used for both creation of an object and an update of the object, and wherein at least some of the update methods of the API comprise a last updated property that when present and has a non-null value, an update is rejected when the value of last updated property does not match a corresponding value in the database.
4. The method of Claim 3, wherein at least some of the update methods set one or more properties that are only set on a first invocation of the update method.
5. The method of Claim 3, wherein returning the response comprises returning an output structure that is a current state of an underlying object that is stored in the database.
6. The method of Claim 3, wherein parameters used by processes that perform the job are obtained from the database record for the underlying objects and not the job record itself.

7. A computer-readable storage medium having computer-executable instructions for use in an online service for requesting actions to be performed, comprising:

- receiving a request through an idempotent and asynchronous application programming interface (API) at a central service to perform a task in an online service that provides content collaboration services for different networks; wherein the API exposes methods comprising get methods and update methods; wherein the request relates to at least one of: configuring, updating and managing machines in at least one of the different networks that are used in providing the online service;
- 10 updating a row of a database with one or more values relating to the task;
- adding a job to a job queue for performing the task;
- returning a response to the requestor after the job is added to the queue; and
- executing the job, wherein executing the job comprises starting a process that accesses the one or more values relating to the task from the database.

15 8. A system for use in an online service for requesting actions to be performed, comprising:

- a processor and a computer-readable medium;
- an operating environment stored on the computer-readable medium and executing on the processor;
- 20 an idempotent and asynchronous application programming interface (API) that exposes methods comprising get methods and update methods;
- a cloud manager that is coupled to different networks that is operative to perform actions, comprising:
 - receiving a request through the API to perform a task in the online
 - 25 service from one of the different networks; wherein the request relates to at least one of: configuring, updating and managing machines in at least one of the different networks that are used in providing the online service;
 - updating a record within a database with one or more values relating to the task;
 - 30 adding a job to a job queue for performing the task;
 - returning a response to the requestor after the job is added to the queue; and

executing the job, wherein executing the job comprises starting a process that accesses the one or more values relating to the task from the database.

9. The system of Claim 8, wherein executing a method in the API having the
5 same parameters multiple times results in a same outcome as calling the method one time.

10. The system of Claim 8, wherein the update methods are used for both creation of an object and an update of the object.