US 20140229698A1

(19) United States

(12) Patent Application Publication

Sivasubramanian et al.

(10) Pub. No.: US 2014/0229698 A1

(43) Pub. Date: Aug. 14, 2014

(54) **CONTROL SERVICE FOR DATA MANAGEMENT**

(71) Applicant: **Amazon Technologies, Inc.**, Reno, NV (US)

(72) Inventors: **Swaminathan Sivasubramanian**, Seattle, WA (US); **Grant Alexander MacDonald McAlister**, Seattle, WA (US); **Paul David Franklin**, Seattle, WA (US); **Rajesh Sudhakar Sheth**, Bellevue, WA (US); **James Horsley**, Bainbridge Island, WA (US)

(73) Assignee: **Amazon Technologies, Inc.**, Reno, NV (US)

(21) Appl. No.: **14/257,927**

(22) Filed: **Apr. 21, 2014**

Related U.S. Application Data

(63) Continuation of application No. 12/415,958, filed on Mar. 31, 2009, now Pat. No. 8,713,060.

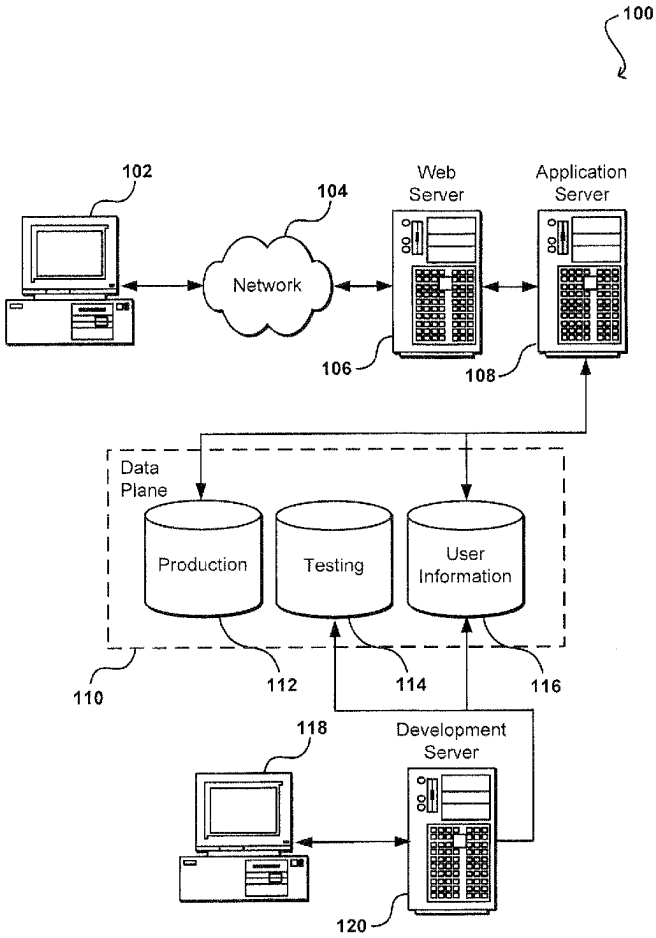
Publication Classification

(51) **Int. Cl.**
G06F 12/02 (2006.01)

(52) **U.S. Cl.**
CPC *G06F 12/02* (2013.01)
USPC **711/165**; 711/170

(57) **ABSTRACT**

Aspects of a data environment, such as the creation, provisioning, and management of data stores and instances, are managed using a separate control environment. A user can call into an externally-facing interface of the control environment, the call being analyzed to determine actions to be performed in the data environment. A monitoring component of the control plane also can periodically communicate with the data environment to determine any necessary actions to be performed, such as to recover from faults or events in the data environment. A workflow can be instantiated that includes tasks necessary to perform the action. For each task, state information can be passed to a component in the data environment operable to perform the task, until all tasks for an action are completed. Data in the data environment can be accessed directly using an externally-facing interface of the data environment, without accessing the control plane.



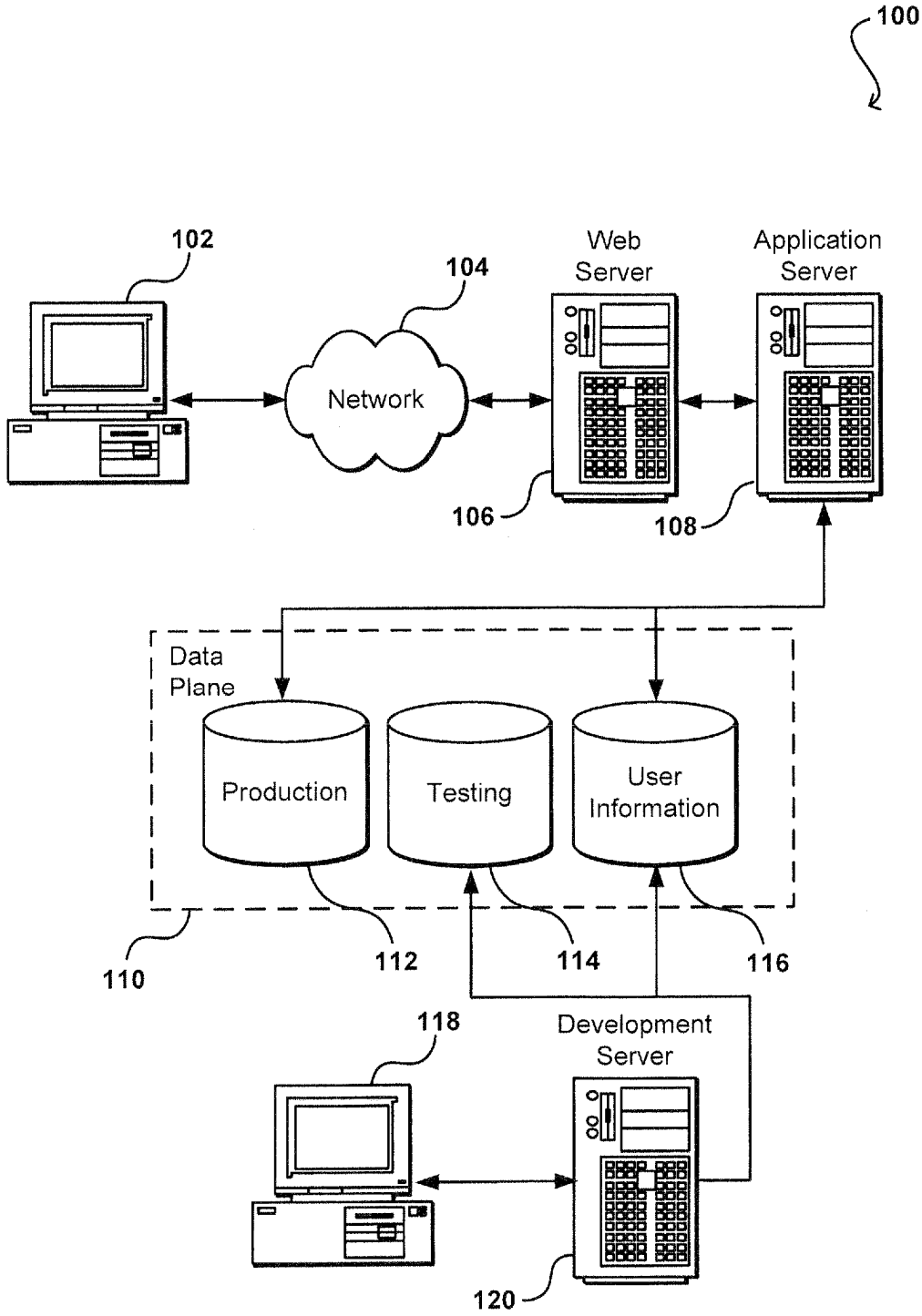

100
102
104
Network
Web Server
Application Server
106
108
Data Plane
Production
Testing
User Information
110
112
114
116
118
Development Server
120

FIG. 1

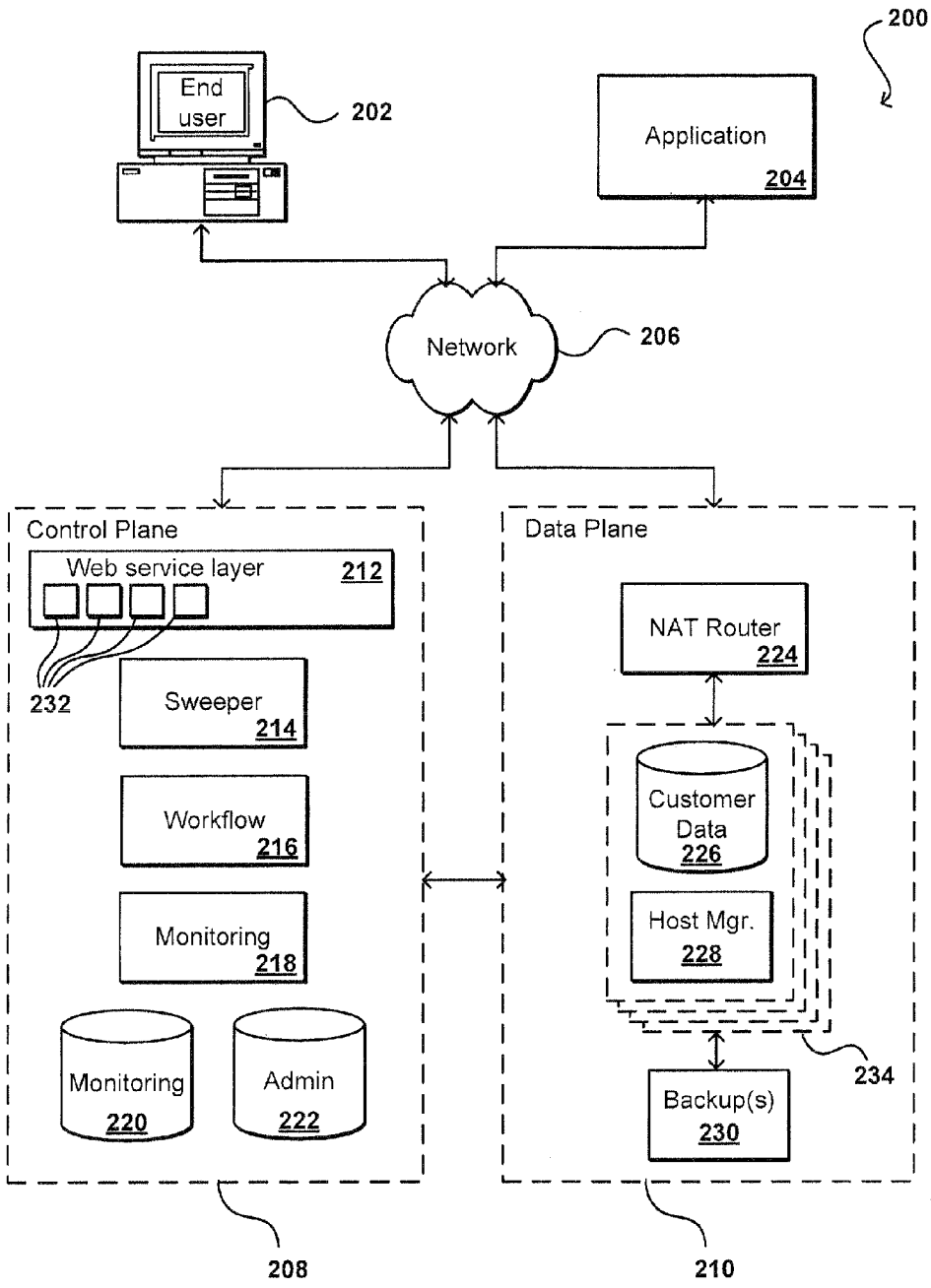

200
End user
202
Application
204
Network
206
Control Plane
Web service layer
212
232
Sweeper
214
Workflow
216
Monitoring
218
Monitoring
220
Admin
222
Data Plane
NAT Router
224
Customer Data
226
Host Mgr.
228
234
Backup(s)
230
208
210

FIG. 2

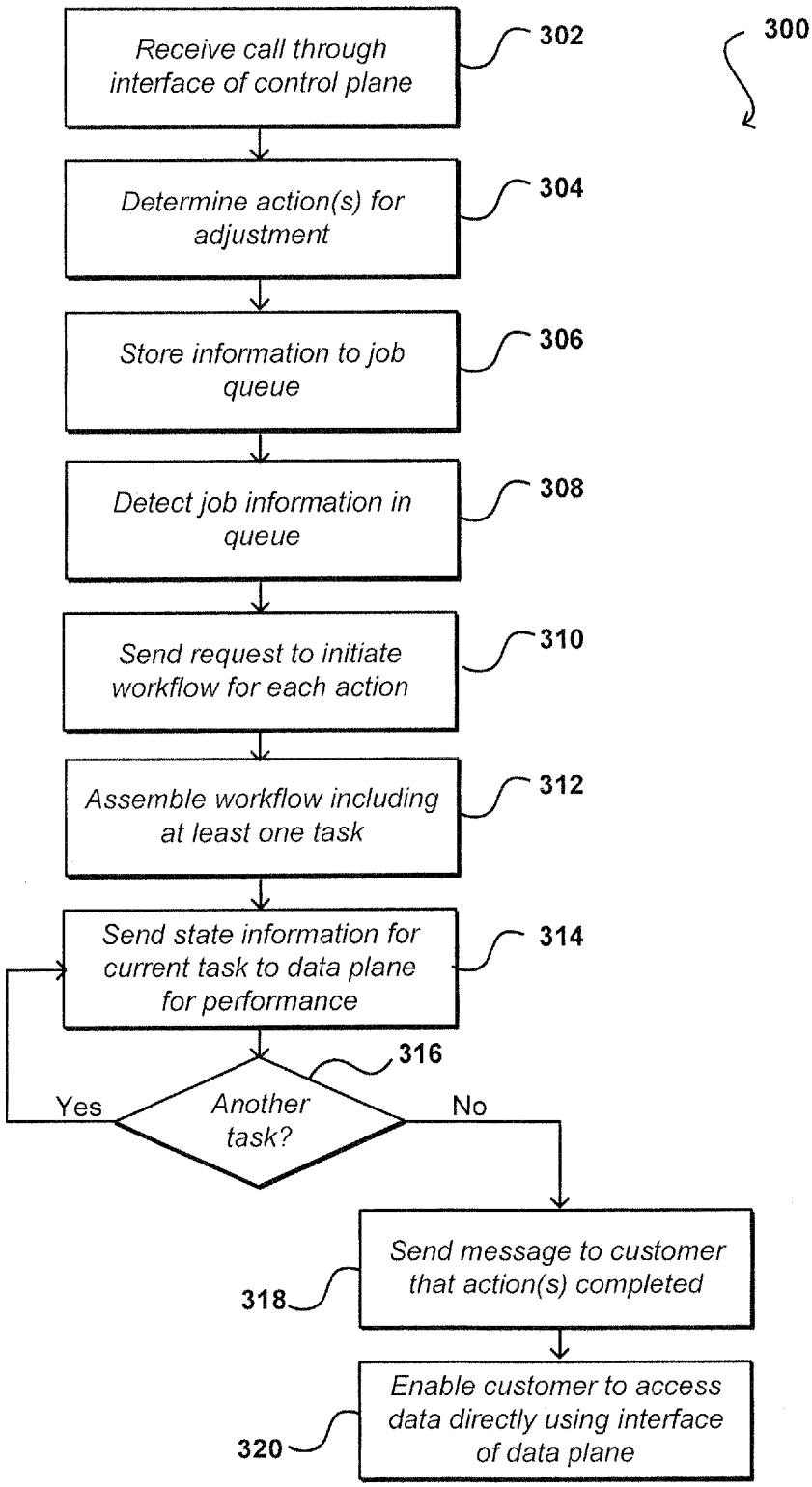

300
Receive call through interface of control plane
302
Determine action(s) for adjustment
304
Store information to job queue
306
Detect job information in queue
308
Send request to initiate workflow for each action
310
Assemble workflow including at least one task
312
Send state information for current task to data plane for performance
314
Another task?
316
Yes
No
Send message to customer that action(s) completed
318
Enable customer to access data directly using interface of data plane
320

FIG. 3

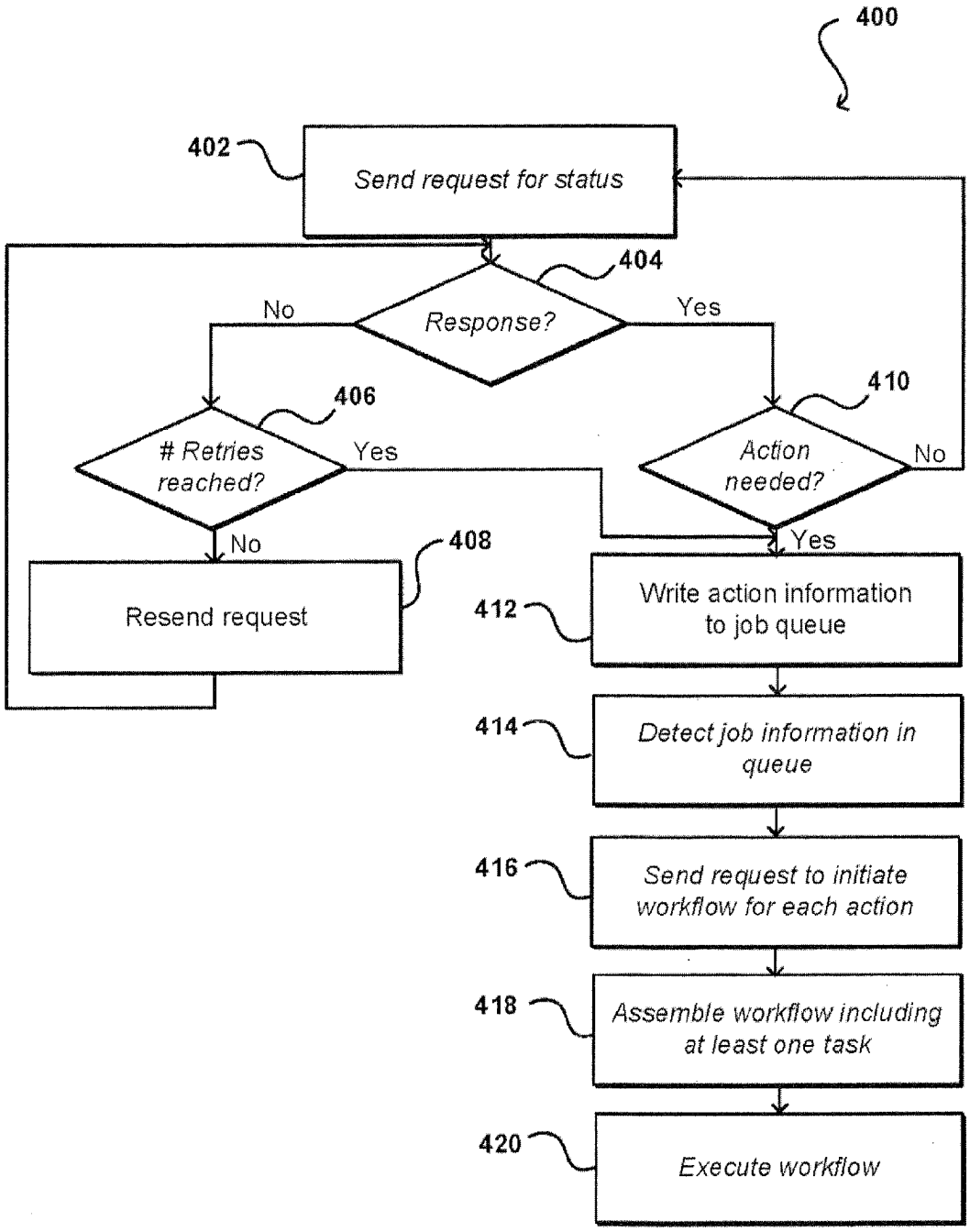

400
402
Send request for status
404
Response?
No
Yes
406
Retries reached?
Yes
No
408
Resend request
410
Action needed?
No
Yes
412
Write action information to job queue
414
Detect job information in queue
416
Send request to initiate workflow for each action
418
Assemble workflow including at least one task
420
Execute workflow

FIG. 4

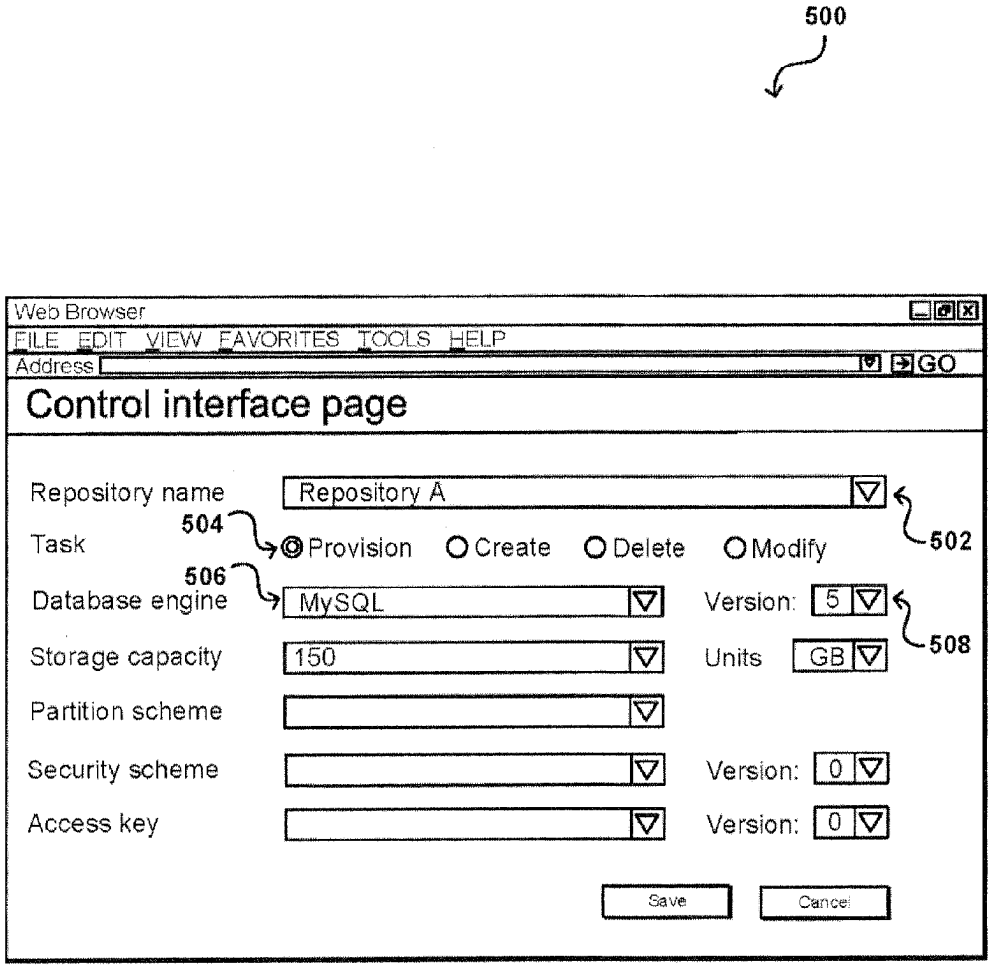

500
Web Browser
FILE EDIT VIEW FAVORITES TOOLS HELP
Address
GO
Control interface page
Repository name
Repository A
502
504
Task
Provision
Create
Delete
Modify
506
Database engine
MySQL
Version:
5
508
Storage capacity
150
Units
GB
Partition scheme
Security scheme
Version:
0
Access key
Version:
0
Save
Cancel

FIG. 5

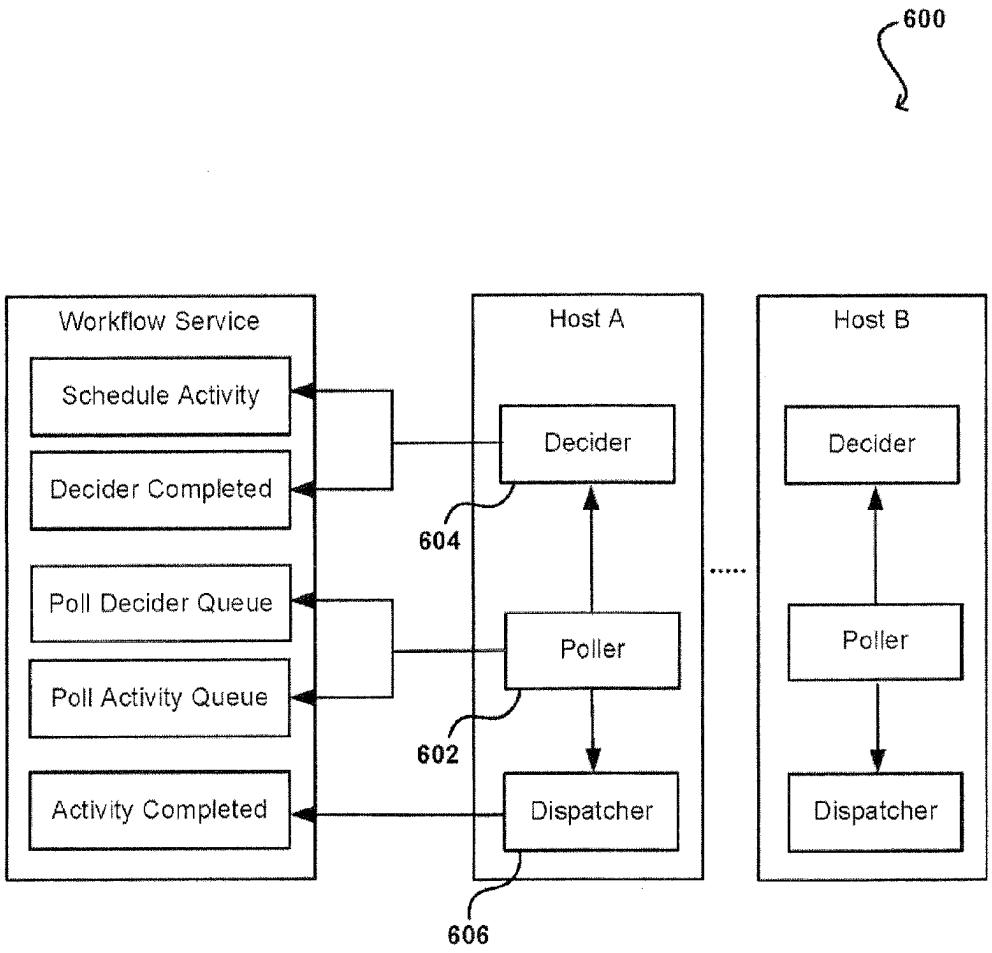

600
Workflow Service
Schedule Activity
Decider Completed
Poll Decider Queue
Poll Activity Queue
Activity Completed
Host A
Decider
Poller
Dispatcher
604
602
606
Host B
Decider
Poller
Dispatcher

FIG. 6

CONTROL SERVICE FOR DATA MANAGEMENT

CROSS REFERENCE TO RELATED APPLICATION

[0001] This application is a continuation of allowed U.S. patent application Ser. No. 12/415,958, filed Mar. 31, 2009, which is hereby incorporated herein by reference for all purposes.

COPYRIGHT NOTICE

[0002] A portion of the disclosure of this patent document contains material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

BACKGROUND

[0003] As an increasing number of applications and services are being made available over networks such as the Internet, an increasing number of content, application, and/or service providers are turning to technologies such as cloud computing. Cloud computing, in general, is an approach to providing access to electronic resources through services, such as Web services, where the hardware and/or software used to support those services is dynamically scalable to meet the needs of the services at any given time. A user or customer typically will rent, lease, or otherwise pay for access to resources through the cloud, and thus does not have to purchase and maintain the hardware and/or software to provide access to these resources.

[0004] Various services have been developed that enable a customer to adjust the compute capacity for an application or service in the cloud over time, making Web-scale computing easier for developers. In some cases, a Web service interface is provided that enables a customer to manage computing capacity, such as to obtain or boot new server instances in order to quickly scale capacity, both up and down, as computing requirements change.

[0005] The adjusting of capacity can be advantageous for applications available across a network or from a remote location. Oftentimes, these applications will rely on, or take advantage of, at least one data store or other such repository hosting data that is used by the respective application. While the processing capacity for the applications or services can be adjusted dynamically through these Web service interfaces, the data repositories upon which these applications rely are not similarly adjustable. Typically, performing tasks such as provisioning and scaling data storage are tedious manual procedures, in which a customer has to provide a database administrator (DBA) or similar expert user with configuration information and requirements, such that the DBA can determine whether the configuration is valid. The DBA typically then has to enable, tune, and optimize the data repository. There is no easy way for a customer to dynamically adjust the data storage capacity, or manage other such aspects of a data repository.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] Various embodiments in accordance with the present disclosure will be described with reference to the drawings, in which:

[0007] FIG. **1** illustrates an environment in which various embodiments can be implemented;

[0008] FIG. **2** illustrates an example separation of a control plane and a data plane that can be used in accordance with various embodiments;

[0009] FIG. **3** illustrates an example process for requesting an action through the control plane to be performed in the data plane in accordance with one embodiment;

[0010] FIG. **4** illustrates an example process for monitoring and automatically performing actions in the data plane in accordance with one embodiment;

[0011] FIG. **5** illustrates example communication flow between hosts in a data plane and a workflow service in the control plane in accordance with one embodiment; and

[0012] FIG. **6** illustrates an example of a display for generating a request for action in the data plane to be submitted to a user interface of the control plane in accordance with one embodiment.

DETAILED DESCRIPTION

[0013] Systems and methods in accordance with various embodiments of the present disclosure may overcome one or more of the aforementioned and other deficiencies experienced in conventional approaches to managing aspects of data storage in an electronic environment. In particular, various embodiments provide a separate control environment, or control plane, that can be used to control aspects of a data environment, or data plane. The functionality of a control plane can be provided as a set of Web services, enabling the control plane to act as a virtual database administrator (DBA). A user or customer can submit a request to the control plane through an externally-visible application programming interface (API), for example, which can be analyzed to determine actions to be performed in the data plane, such as actions that create, delete, modify, expand, or otherwise modify a data store or data storage instance. A monitoring component of the control plane also can be provided that can monitor the health or status of components in the data plane, and can automatically determine actions to be taken in the data plane. State information can be passed to a component of the data plane for each task necessary to perform the action, such that the control plane can manage the performance of the tasks without having direct access into the data stores or other such components of the data plane. Once provisioned, a user can native access to the data instance(s) in the data plane, and can simply point existing applications (such as MySQL applications) to the DNS (domain name system) address or other location information for the particular instance. There is no restriction or modification of query models or other such functionality, as a user can continue to use applications built on MySQL, Oracle, or other such database technology.

[0014] FIG. **1** illustrates an example of an environment **100** for implementing aspects in accordance with various embodiments. As will be appreciated, although a Web-based environment is used for purposes of explanation, different environments may be used, as appropriate, to implement various embodiments. The environment **100** shown includes both a testing or development portion (or side) and a production portion. The production portion includes an electronic client device **102**, which can include any appropriate device operable to send and receive requests, messages, or information over an appropriate network **104** and convey information back to a user of the device. Examples of such client devices include personal computers, cell phones, handheld messag-

ing devices, laptop computers, set-top boxes, personal data assistants, electronic book readers, and the like. The network can include any appropriate network, including an intranet, the Internet, a cellular network, a local area network, or any other such network or combination thereof. Components used for such a system can depend at least in part upon the type of network and/or environment selected. Protocols and components for communicating via such a network are well known and will not be discussed herein in detail. Communication over the network can be enabled by wired or wireless connections, and combinations thereof. In this example, the network includes the Internet, as the environment includes a Web server **106** for receiving requests and serving content in response thereto, although for other networks an alternative device serving a similar purpose could be used as would be apparent to one of ordinary skill in the art.

[0015] The illustrative environment includes at least one application server **108** and a data store **110**. It should be understood that there can be several application servers, layers, or other elements, processes, or components, which may be chained or otherwise configured, which can interact to perform tasks such as obtaining data from an appropriate data store. As used herein the term "data store" refers to any device or combination of devices capable of storing, accessing, and retrieving data, which may include any combination and number of data servers, databases, data storage devices, and data storage media, in any standard, distributed, or clustered environment. The application server can include any appropriate hardware and software for integrating with the data store as needed to execute aspects of one or more applications for the client device, handling a majority of the data access and business logic for an application. The application server provides access control services in cooperation with the data store, and is able to generate content such as text, graphics, audio, and/or video to be transferred to the user, which may be served to the user by the Web server in the form of HTML, XML, or another appropriate structured language in this example. The handling of all requests and responses, as well as the delivery of content between the client device **102** and the application server **108**, can be handled by the Web server. It should be understood that the Web and application servers are not required and are merely example components, as structured code discussed herein can be executed on any appropriate device or host machine as discussed elsewhere herein. Further, the environment can be architected in such a way that a test automation framework can be provided as a service to which a user or application can subscribe. A test automation framework can be provided as an implementation of any of the various testing patterns discussed herein, although various other implementations can be used as well, as discussed or suggested herein.

[0016] The environment also includes a development and/or testing side, which includes a user device **118** allowing a user such as a developer, data administrator, or tester to access the system. The user device **118** can be any appropriate device or machine, such as is described above with respect to the client device **102**. The environment also includes a development server **120**, which functions similar to the application server **108** but typically runs code during development and testing before the code is deployed and executed on the production side and is accessible to outside users, for example. In some embodiments, an application server can function as a development server, and separate production and testing storage may not be used.

[0017] The data store **110** can include several separate data tables, databases, or other data storage mechanisms and media for storing data relating to a particular aspect. For example, the data store illustrated includes mechanisms for storing production data **112** and user information **116**, which can be used to serve content for the production side. The data store also is shown to include a mechanism for storing testing data **114**, which can be used with the user information for the testing side. It should be understood that there can be many other aspects that may need to be stored in the data store, such as for page image information and access right information, which can be stored in any of the above listed mechanisms as appropriate or in additional mechanisms in the data store **110**. The data store **110** is operable, through logic associated therewith, to receive instructions from the application server **108** or development server **120**, and obtain, update, or otherwise process data in response thereto. In one example, a user might submit a search request for a certain type of item. In this case, the data store might access the user information to verify the identity of the user, and can access the catalog detail information to obtain information about items of that type. The information then can be returned to the user, such as in a results listing on a Web page that the user is able to view via a browser on the user device **102**. Information for a particular item of interest can be viewed in a dedicated page or window of the browser.

[0018] Each server typically will include an operating system that provides executable program instructions for the general administration and operation of that server, and typically will include a computer-readable medium storing instructions that, when executed by a processor of the server, allow the server to perform its intended functions. Suitable implementations for the operating system and general functionality of the servers are known or commercially available, and are readily implemented by persons having ordinary skill in the art, particularly in light of the disclosure herein.

[0019] The environment in one embodiment is a distributed computing environment utilizing several computer systems and components that are interconnected via communication links, using one or more computer networks or direct connections. However, it will be appreciated by those of ordinary skill in the art that such a system could operate equally well in a system having fewer or a greater number of components than are illustrated in FIG. **1**. Thus, the depiction of the system **100** in FIG. **1** should be taken as being illustrative in nature, and not limiting to the scope of the disclosure.

[0020] An environment such as that illustrated in FIG. **1** can be useful for a provider such as an electronic marketplace, wherein multiple hosts might be used to perform tasks such as serving content, authenticating users, performing payment transactions, or performing any of a number of other such tasks. Some of these hosts may be configured to offer the same functionality, while other servers might be configured to perform at least some different functions. The electronic environment in such cases might include additional components and/or other arrangements, such as those illustrated in the configuration **200** of FIG. **2**, discussed in detail below.

[0021] Systems and methods in accordance with one embodiment provide a relational database service ("RDS") that enables developers, customers, or other authorized users to easily and cost-effectively obtain and configure relational databases so that users can perform tasks such as storing, processing, and querying relational data sets in a cloud. While this example is discussed with respect to the Internet, Web

services, and Internet-based technology, it should be understood that aspects of the various embodiments can be used with any appropriate services available or offered over a network in an electronic environment. Further, while the service is referred to herein as a "relational database service," it should be understood that such a service can be used with any appropriate type of data repository or data storage in an electronic environment. An RDS in this example includes at least one Web service that enables users or customers to easily manage relational data sets without worrying about the administrative complexities of deployment, upgrades, patch management, backups, replication, failover, capacity management, scaling, and other such aspects of data management. Developers are thus freed to develop sophisticated cloud applications without worrying about the complexities of managing the database infrastructure.

[0022] An RDS in one embodiment provides a separate "control plane" that includes components (e.g., hardware and software) useful for managing aspects of the data storage. In one embodiment, a set of data management application programming interfaces (APIs) or other such interfaces are provided that allow a user or customer to make calls into the RDS to perform certain tasks relating to the data storage. The user still can use the direct interfaces or APIs to communicate with the data repositories, however, and can use the RDS-specific APIs of the control plane only when necessary to manage the data storage or perform a similar task.

[0023] FIG. **2** illustrates an example of an RDS implementation **200** that can be used in accordance with one embodiment. In this example, a computing device **202** for an end user is shown to be able to make calls through a network **206** into a control plane **208** to perform a task such as to provision a data repository of the data plane **210**. The user or an application **204** can access the provisioned repository directly through an interface of a data plane **210**. While an end user computing device and application are used for purposes of explanation, it should be understood that any appropriate user, application, service, device, component, or resource can access the interface(s) of the control plane and/or data plane as appropriate in the various embodiments. Further, while the components are separated into control and data "planes," it should be understood that this can refer to an actual or virtual separation of at least some resources (e.g., hardware and/or software) used to provide the respective functionality.

[0024] The control plane **208** in this example is essentially a virtual layer of hardware and software components that handles control and management actions, such as provisioning, scaling, replication, etc. The control plane in this embodiment includes a Web services layer **212**, or tier, which can include at least one Web server, for example, along with computer-executable software, application servers, or other such components. The Web services layer also can include a set of APIs **232** (or other such interfaces) for receiving Web services calls or requests from across the network **206**, which the Web services layer can parse or otherwise analyze to determine the steps or actions needed to act on or process the call. For example, a Web service call might be received that includes a request to create a data repository. In this example, the Web services layer can parse the request to determine the type of data repository to be created, the storage volume requested, the type of hardware requested (if any), or other such aspects. Information for the request can be written to an administration ("Admin") data store **222**, or other appropriate storage location or job queue, for subsequent processing.

[0025] A Web service layer in one embodiment includes a scalable set of customer-facing servers that can provide the various control plane APIs and return the appropriate responses based on the API specifications. The Web service layer also can include at least one API service layer that in one embodiment consists of stateless, replicated servers which process the customer APIs. The Web service layer can be responsible for Web service front end features such as authenticating customers based on credentials, authorizing the customer, throttling customer requests to the API servers, validating user input, and marshalling or unmarshalling requests and responses. The API layer also can be responsible for reading and writing database configuration data to/from the administration data store, in response to the API calls. In many embodiments, the Web services layer will be the only externally visible component, or the only component that is visible to, and accessible by, customers of the control service. The servers of the Web services layer can be stateless and scaled horizontally as known in the art. API servers, as well as the persistent data store, can be spread across multiple data centers in a region, for example, such that the servers are resilient to single data center failures.

[0026] The control plane in this embodiment includes what is referred to herein as a "sweeper" component **214**. A sweeper component can be any appropriate component operable to poll various components of the control plane or otherwise determine any tasks to be executed in response to an outstanding request. In this example, the Web services layer might place instructions or information for the "create database" request in the admin data store **222**, or a similar job queue, and the sweeper can periodically check the admin data store for outstanding jobs. Various other approaches can be used as would be apparent to one of ordinary skill in the art, such as the Web services layer sending a notification to a sweeper that a job exists. The sweeper component can pick up the "create database" request, and using information for the request can send a request, call, or other such command to a workflow component **216** operable to instantiate at least one workflow for the request. The workflow in one embodiment is generated and maintained using a workflow service as is discussed elsewhere herein. A workflow in general is a sequence of tasks that should be executed to perform a specific job. The workflow is not the actual work, but an abstraction of the work that controls the flow of information and execution of the work. A workflow also can be thought of as a state machine, which can manage and return the state of a process at any time during execution. A workflow component (or system of components) in one embodiment is operable to manage and/or perform the hosting and executing of workflows for tasks such as: repository creation, modification, and deletion; recovery and backup; security group creation, deletion, and modification; user credentials management; and key rotation and credential management. Such workflows can be implemented on top of a workflow service, as discussed elsewhere herein. The workflow component also can manage differences between workflow steps used for different database engines, such as MySQL, as the underlying workflow service does not necessarily change.

[0027] In this example, a workflow can be instantiated using a workflow template for creating a database and applying information extracted from the original request. For example, if the request is for a My SQL® Relational Database Management System (RDBMS) instance, as opposed to an Oracle®RDBMS or other such instance, then a specific task

will be added to the workflow that is directed toward MySQL instances. The workflow component also can select specific tasks related to the amount of storage requested, any specific hardware requirements, or other such tasks. These tasks can be added to the workflow in an order of execution useful for the overall job. While some tasks can be performed in parallel, other tasks rely on previous tasks to be completed first. The workflow component or service can include this information in the work flow, and the tasks can be executed and information passed as needed.

[0028] An example "create database" workflow for a customer might includes tasks such as provisioning a data store instance, allocating a volume of off-instance persistent storage, attaching the persistent storage volume to the data store instance, then allocating and attaching a DNS address or other address, port, interface, or identifier which the customer can use to access or otherwise connect to the data instance. In this example, a user is provided with the DNS address and a port address to be used to access the instance. The workflow also can include tasks to download and install any binaries or other information used for the specific data storage technology (e.g., MySQL). The workflow component can manage the execution of these and any related tasks, or any other appropriate combination of such tasks, and can generate a response to the request indicating the creation of a "database" in response to the "create database" request, which actually corresponds to a data store instance in the data plane **210**, and provide the DNS address to be used to access the instance. A user then can access the data store instance directly using the DNS address and port, without having to access or go through the control plane **208**. Various other workflow templates can be used to perform similar jobs, such as deleting, creating, or modifying one of more data store instances, such as to increase storage. In some embodiments, the workflow information is written to storage, and at least one separate execution component (not shown) pulls or otherwise accesses or receives tasks to be executed based upon the workflow information. For example, there might be a dedicated provisioning component that executes provisioning tasks, and this component might not be called by the workflow component, but can monitor a task queue or can receive information for a provisioning task in any of a number of related ways as should be apparent.

[0029] As mentioned, various embodiments can take advantage of a workflow service that can receive requests or calls for a current state of a process or task, such as the provisioning of a repository, and can return the current state of the process. The workflow component and/or workflow service do not make the actual calls or requests to perform each task, but instead manage the state and configuration information for the workflow that enables the components of the control plane to determine the next task to be performed, and any information needed for that task, then generate the appropriate call(s) into the data plane including that state information, whereby a component of the data plane can make the call to perform the task. Workflows and tasks can be scheduled in parallel in order to increase throughput and maximize processing resources. As discussed, the actual performing of the tasks will occur in the data plane, but the tasks will originate from the control plane. For example, the workflow component can communicate with a host manager, which can make calls into the data store. Thus, for a given task a call could be made to the workflow service passing certain parameters, whereby the workflow service generates the sequence of tasks for the workflow and provides the current state, such that a task for the present state can be performed. After the task is performed (or otherwise resolved or concluded), a component such as the host manager can reply to the service, which can then provide information about the next state in the workflow, such that the next task can be performed. Each time one of the tasks for the workflow is performed, the service can provide a new task to be performed until the workflow is completed. Further, multiple threads can be running in parallel for different workflows to accelerate the processing of the workflow.

[0030] The control plane **208** in this embodiment also includes at least one monitoring component **218**. When a data instance is created in the data plane, information for the instance can be written to a data store in the control plane, such as a monitoring data store **220**. It should be understood that the monitoring data store can be a separate data store, or can be a portion of another data store such as a distinct set of tables in an Admin data store **222**, or other appropriate repository. A monitoring component can access the information in the monitoring data store to determine active instances **234** in the data plane **210**. A monitoring component also can perform other tasks, such as collecting log and/or event information from multiple components of the control plane and/or data plane, such as the Web service layer, workflow component, sweeper component, and various host managers. Using such event information, the monitoring component can expose customer-visible events, for purposes such as implementing customer-facing APIs. A monitoring component can constantly monitor the health of all the running repositories and/or instances for the control plane, detect the failure of any of these instances, and initiate the appropriate recovery process (es).

[0031] Each instance **234** in the data plane can include at least one data store **226** and a host manager component **228** for the machine providing access to the data store. A host manager in one embodiment is an application or software agent executing on an instance and/or application server, such as a Tomcat or Java application server, programmed to manage tasks such as software deployment and data store operations, as well as monitoring a state of the data store and/or the respective instance. A host manager in one embodiment listens on a port that can only be reached from the internal system components, and is not available to customers or other outside entities. In some embodiments, the host manager cannot initiate any calls into the control plane layer. A host manager can be responsible for managing and/or performing tasks such as setting up the instances for a new repository, including setting up logical volumes and file systems, installing database binaries and seeds, and starting or stopping the repository. A host manager can monitor the health of the data store, as well as monitoring the data store for error conditions such as I/O errors or data storage errors, and can restart the data store if necessary. A host manager also perform and/or mange the installation of software patches and upgrades for the data store and/or operating system. A host manger also can collect relevant metrics, such as may relate to CPU, memory, and I/O usage.

[0032] The monitoring component can communicate periodically with each host manager **228** for monitored instances **234**, such as by sending a specific request or by monitoring heartbeats from the host managers, to determine a status of each host. In one embodiment, the monitoring component includes a set of event processors (or monitoring servers)

configured to issue commands to each host manager, such as to get the status of a particular host and/or instance. If a response is not received after a specified number of retries, then the monitoring component can determine that there is a problem and can store information in the Admin data store **222** or another such job queue to perform an action for the instance, such as to verify the problem and re-provision the instance if necessary. The sweeper can access this information and kick off a recovery workflow for the instance to attempt to automatically recover from the failure. The host manager **228** can act as a proxy for the monitoring and other components of the control plane, performing tasks for the instances on behalf of the control plane components. Occasionally, a problem will occur with one of the instances, such as the corresponding host, instance, or volume crashing, rebooting, restarting, etc., which cannot be solved automatically. In one embodiment, there is a logging component (not shown) that can log these and other customer visibility events. The logging component can include an API or other such interface such that if an instance is unavailable for a period of time, a customer can call an appropriate "events" or similar API to get the information regarding the event. In some cases, a request may be left pending when an instance fails. Since the control plane in this embodiment is separate from the data plane, the control plane never receives the data request and thus cannot queue the request for subsequent submission (although in some embodiments this information could be forwarded to the control plane). Thus, the control plane in this embodiment provides information to the user regarding the failure so the user can handle the request as necessary.

[0033] As discussed, once an instance is provisioned and a user is provided with a DNS address or other address or location, the user can send requests "directly" to the data plane **210** through the network using a Java Database Connectivity (JDBC) or other such client to directly interact with that instance **234**. In one embodiment, the data plane takes the form of (or at least includes or is part of) a computing cloud environment, or a set of Web services and resources that provides data storage and access across a "cloud" or dynamic network of hardware and/or software components. A DNS address is beneficial in such a dynamic cloud environment, as instance or availability failures, for example, can be masked by programmatically remapping a DNS address to any appropriate replacement instance for a use. A request received from a user **202** or application **204**, for example, can be directed to a network address translation (NAT) router **224**, or other appropriate component, which can direct the request to the actual instance **234** or host corresponding to the DNS of the request. As discussed, such an approach allows for instances to be dynamically moved, updated, replicated, etc., without requiring the user or application to change the DNS or other address used to access the instance. As discussed, each instance **234** can include a host manager **228** and a data store **226**, and can have at least one backup instance or copy in persistent storage **230**. Using such an approach, once the instance has been configured through the control plane, a user, application, service, or component can interact with the instance directly through requests to the data plane, without having to access the control plane **232**. For example, the user can directly issue structured query language (SQL) or other such commands relating to the data in the instance through the DNS address. The user would only have to access the control plane if the user wants to perform a task such as expanding the storage capacity of an instance. In at least one embodiment, the functionality of the control plane **208** can be offered as at least one service by a provider that may or may not be related to a provider of the data plane **210**, but may simply be a third-party service that can be used to provision and manage data instances in the data plane, and can also monitor and ensure availability of those instances in a separate data plane **210**.

[0034] As discussed, one advantage to providing the functionality of a control plane as a Web service or other such service is that the control plane functions as a virtual database administrator (DBA) and avoids the need for a human DBA to perform tasks such as provisioning data. Provisioning data is presently a tedious manual procedure, requiring a DBA to receive the necessary configuration information, determine whether the configuration is valid, optimize and tune the instance, and perform other such tasks, which take a significant amount of time and effort. Further, such an approach provides many opportunities for error, which might not be discovered until after data is lost. Using a control plane or service as described herein, a user or customer can instead submit a call including information such as a type of hardware and a version of a database product. The control plane or service can then perform the necessary tasks to create, delete, modify, expand, or otherwise modify a data store or data storage instance. The control plane also can support several different database engines in a consistent fashion, without requiring a DBA to be an expert in each of the engines. Once provisioned, the user has native access to the data instance(s), and can simply point existing applications (such as MySQL applications) to the DNS address or other location information for the particular instance. There is no restriction or modification of query models or other such functionality, as a user can continue to use applications built on MySQL, Oracle, or other database technology.

[0035] Using components such as those discussed above, FIG. **3** illustrates an example process **300** by which a customer can request the performance of a control-related task with respect to at least one data instance in a data environment, here the data plane, using the control plane or a similar data control service. While the term "customer" is used herein to refer to the "owner" of data, or a data store or instance hosted by the RDS system, it should be understood that the term customer is merely an example, and that any appropriate user or developer can be allowed to access the control plane and data plane in the various embodiments. A request, such as a Web services call, is received through a customer-facing control plane interface component **302**. The request is analyzed to determine at least one action needed to process the request **304**. As discussed, this can take the form of a component of a Web services layer parsing the request to determine the action(s) being requested. In this embodiment, information for the action, such as the type of action and parameters to be used to perform the action, is written to a job queue **306**, such as may be located in an Admin data store or other such storage location. The job queue can be monitored, such as by a sweeper component, to determine the presence of job information **308** and, when job information is detected, a request can be sent to initiate a workflow for the requested action **310**. This can include a request sent by the sweeper component to a workflow component and/or service to instantiate a workflow. In other embodiments, a workflow component might monitor the job queue for jobs, or a component of the Web services layer may send the job information directly to a workflow component.

[0036] Upon receiving the job information, the information is analyzed to determine and/or assemble an appropriate workflow for the requested action **312**. As discussed, different tasks can be selected for the workflow based upon factors such as the type of action requested and the type of database engine being used. Beginning with the first task of the workflow, state information is sent to a host manager in the data environment operable to use the state information to determine a task to be performed, perform the task with respect to a data repository and/or data instance, and return a response upon completion of the task **314**. Upon receiving the response, the workflow component determines whether there is another task to be performed **316**. If so, state information for the next task is sent to the host manager, and upon completion of that task the host manager sends a response to the workflow component. After the final task has been completed, a message is sent to the requesting customer (or another appropriate user, application, or location) that the requested action has been completed **318**. After the action has been performed, the customer is able to directly access the data instance upon which the action was performed using a data interface of the data environment, without accessing or passing through the control plane **320**. As mentioned, the user can provided with a DNS address and port number, for example, such that if the action resulted in movement of data or another similar action, the customer or an application can continue to use the same DNS address, which will be directed to the appropriate location in the data plane.

[0037] Similarly, FIG. **4** illustrates an example process **400** by which a control plane or control service can monitor the performance of a data instance (or data store, repository, etc.) in a data environment, here the data plane. A request for status is sent to a host manager component for a data instance **402**. A determination is made as to whether a response is received within a specified amount of time **404**. If no response is received, it is determined whether a threshold number of requests have been sent **406**. If a threshold number of requests have not been send, another request can be sent **408**. If a response message is received, the response is analyzed to determine whether the message includes any errors or tasks to be addressed **410**. If not, and the instance is determined to be healthy, the process can continue with another request for status being sent at a later time. If the response message indicates that an action needs to be performed with respect to the data instance, information for the action, such as the type of action and parameters to be used to perform the action, is written to a job queue **412**, such as may be located in an Admin data store or other such storage location. The job queue can be monitored to determine the presence of job information **414**, such as by a sweeper component, and when job information is detected, a request can be sent to initiate a workflow for the requested action **416**. In other embodiments, a workflow component might monitor the job queue for jobs, or a component of the Web services layer may send the job information directly to a workflow component.

[0038] Upon receiving the job information, the information is analyzed to determine and/or assemble an appropriate workflow for the requested action **418**. Beginning with the first task of the workflow, state information is sent to a host manager in the data environment to perform the task and execute the workflow **420**, such as may be accomplished using a process described with respect to steps **314** to **316** of the process of FIG. **3**. If the final task is completed successfully, the data instance can simply continue to process requests sent from customers or applications via a data plane interface. If any task cannot be completed successfully, a message can be sent to a customer (or another appropriate user, application, or location) indicating a potential problem with the data instance. Various other notification actions can occur, such as generating an action notification and/or adding information to an error log.

[0039] Specific Interface Examples

[0040] As discussed above, users of the control plane can perform various tasks relating to data repositories and data instances using a set of APIs or other such interfaces. While the selection and names of the example APIs are used for purposes of explanation, it should be apparent that other selections, combinations, names, and other aspects can vary between the various embodiments. As discussed in one of the examples above, customers can create a data store using a "CreateDatabase" or similar API. The user can call a Web service to specify any desired values for an instance type (which describes the CPU and memory capacity), storage size, repository name, port, and other such values. The customer could also utilize a "DescribeDatabase" or similar API to poll on the status of the repository to determine the state of the repository, such as whether the repository state is provisioned. When the status of database is "AVAILABLE," for example, the customer can retrieve an endpoint which is returned as part of a response to the DescribeDatabase call. Customers can delete a repository or instance using a "DeleteDatabase" or similar API. Customers also can have the ability to hibernate a repository or instance, placing an instance in a "sleep" state, for example, using a "HibernateDatabase" or similar API. During such a "sleep" state, the data typically will not be accessible but the data will be backed up durably. Customers can wake a hibernated data repository or instance using a "ResumeDatabase" or similar API.

[0041] As mentioned earlier, a control plane or service can handle the complexity of not just database provisioning, but also tasks such as upgrades, patch management, backups, and failover. A customer can control the times for backups and maintenance activities by enabling customers to specify (or modify) the backup window and maintenance window times while invoking a "CreateDatabase" (or "ModifyDatabase" or similar) API. Using a "ModifyDatabase" API, customers can increase the storage size, change the instance type, or modify various other fields.

[0042] Customers also can be provided with at least one "Database Access Control" or similar API. When a data repository is created, users can specify one or more existing security groups to restrict network access to the repository. Customers can authorize access to the repository by adding permission rules to the security group that is applied to the repository using an API such as an AuthorizeDBSecurityGroupIngress API. Customers can also add or remove security groups from a repository at any time using an API such as a "ModifyDatabase" API. Customers can create (or delete) security groups using similar APIs, such as a "CreateDBSecurityGroup" (or "DeleteDBSecurityGroups") API.

[0043] A control plane also can provide at least one "Database User Management" or similar API. As part of a CreateDatabase API, for example, customers in one embodiment can be expected to supply the username and password for a special repository user, such as may be called a "Repository Owner." A Repository Owner is a special type of user who owns the repository schema objects. After creation of a

repository, a customer can perform tasks such as to add more users to using a "CreateDatabaseUser" API, remove users using a "DeleteDatabaseUser" API, and list customers using a "DescribeDatabaseUsers" API. Customers also can obtain the history of events (such as outages due to maintenance or backup related events) related to the repositories and instances using a "DescribeEvents" or similar API.

[0044] Customer Example

[0045] In this example a customer wants to provision a new data instance, instead of maintaining and managing an existing MySQL database. In this example, the existing CUSTOMER database is 60 GB, and storage growth estimates are around 10% per month. Based on these initial capacity requirements, the customer selects an instance to be provisioned with an initial capacity of 80 GB. The customer chooses a master user and master user password, and based on firewall requirements chooses an appropriate port number (e.g., 4030) on which the data instance will be listening.

[0046] The customer, if not already signed up or subscribed to the control service, can sign up for the service. In some embodiments, the user will receive software or will access an interface page through the Internet, for example, that will allow a user to submit requests to the control plane or service. For example, FIG. **5** illustrates an example of a display **500**, here a page rendered in a browser application, that can allow a user to make calls into the control plane. As illustrated, the interface can include options that allow the user to enter information needed to perform a control action on the data plane. For example, the interface page can include options to specify a repository for the action **502**, select an action to be performed **504**, and specify options for the action **506**, such as a database engine to be use or capacity to be requested, as well as version information **508** or other such options. In other embodiments, a user can manually (or otherwise) create and submit Web service calls to the control plane. In the following example, the customer generates a request to create a new data repository using a command line tool. A request can take the form of, for example:

```
rds-create-database --identifier customerprod --dbname customer -
-size 80 -class small --engine mysqI5.1 --master master_username
--password master_password --port 4030
```

The customer can have the ability to check on the provisioning status, and can request a connect string using the command line tool to describe the repository, such as by submitting:

[0047] describe-repositories customerprod

The customer can grant access to the default security group, such as from the address range 205.192.0.0/16 by:

[0048] authorize default -s 205.192.0.0/16

The customer also can check on the status of security changes, such as by submitting:

[0049] describe-group default

[0050] Once the request has been submitted, the control plane can asynchronously execute the request to provision the repository. A "DescribeDatabases" or similar API can be used to determine the status of the request. While provisioning is still in progress, the status will show as "Pending Creation," for example, and can be changed to a state such as "Created" once the provisioning has been completed. At this point, the customer can have all the information necessary for connecting to the repository.

[0051] Once the repository has been provisioned and is available, the customer can perform various actions on the data plane. For example, the customer can populate the CUSTOMER repository, such as by using a MySQL dump utility or similar data transfer process. The customer in this example runs a command (i.e., on the source MySQL database server using a compatible client utility) such as the following:

```
$ mysqldump -opt customer | mysql--host end_point_hostname --port
4030 -C customer
```

The customer also can verify that the required tables are created, such as by submitting:

```
$ mysql -u master_usernname -h end_point_hostname --port 4030 -p
master_password

Mysql> show tables;

+-------------------------+
| Tables in customer      |
+-------------------------+
| Table1                  |
| Table2                  |
| Table3                  |
+-------------------------+
```

The repository is ready for use by one of the customer's applications. The customer changes the connect string (or other pointer) for the application to point to new instance database instead of the original self-managed database.

[0052] In this example, the customer also wants to implement data security through role-based access control. Before turning on the provisioned and loaded data instance and making the instance available, the customer wants to implement role based access control such that a development team will have read/write access to the repository but business analysts will only obtain read access. The client also wants "master user" access limited to handful of senior members, so the remaining developers need a different database user role.

[0053] With respect to the control plane, the customer can submit a request to create the new database user using the command line tool, for example, such as by submitting the following:

```
create-user --identifier customerprod --username develop1 --
  password develop1
create-user --identifier customerprod --username analyst1--
  password analyst1
```

The customer can also check on provisioning status for the request, such as by submitting:

[0054] describe-users customerprod

While provisioning is still in progress, the status can show a state such as "Pending Creation," and the status will be changed to a state such as "Created" once the provisioning has been completed. The customer can now perform necessary tasks for securing the users in the data plane.

[0055] The customer then can, with respect to the data plane, grant read/write privileges to a develop1 user for all tables owned by master username, such as by submitting:

```
$ mysql-u master_username -h end_point_hostname --port 4030 -p
  master1
Mysql>grant select, insert, update, delete on master_username.*
  to 'develop1'@'%';
```

The customer can also grant read privileges to an analyst1 user for all tables owned by master username:

```
$ mysql-u master_username -h end_point_hostname --port 4030 -p
  master1
Mysql>grant select on master_username.*to'analyst1'@'%';
```

[0056] After the instance has been running for a while, the customer may decide to scale up the size of the instance, such as to 150 GB of storage. The customer in this example thus can submit a request to modify the database capacity using the command line tool, such as by submitting:

[0057] modify-database - - identifier customerprod - - size 150

If the system also allows to adjust the size of an instance for compute or processing needs, then that adjustment can be made in the same or a similar command by specifying an additional parameter value. The customer also can check on a provisioning status for the adjustment by submitting a command such as:

[0058] describe-databases customerprod

The requested repository modifications in one embodiment take place during the maintenance window specified by the customer as discussed above. While the changes are in progress, the status be, for example, "Pending Modification," which can be changed to a value such as "Active" once the provisioning has been completed. As discussed, the customer does not have to take any actions on the data plane side during the execution of this request. On the control plane side, the customer can subscribe to a service such as an auto-scaling plan. Once subscribed, the customer does not have to take any action, even on the control plane, as the auto-scaling can be configured to manage capacity for the customer and scale up or down as needed.

[0059] At some point, the customer may wish to implement improved or updated processes for various development needs and may wish to set up a test instance of a particular data store. The customer may also want to take a snapshot of the production instance so that the test instance is fully populated and comparable with the production date. The customer decides that, for the particular needs of the testing procedure, the customer can utilize a SMALL instance for purposes of processing capacity, and can provision the same storage capacity as is used for production. The customer thus can submit a request to clone the database using a command line tool, such as by submitting:

```
create-database --identifier customertest --dbname tcustomer --
  size 150 --class small--engine mysql5.1--master master_username
--password master_password--port 4030
```

The customer can also check on the provisioning status by submitting a command such as:

[0060] describe-databases customertest

The requested modifications can take place during the maintenance window previously specified by the customer. While the changes are in progress, the status will show as "Pending Modification," for example, and can be changed to a state such as "Active" once the provisioning has been completed. The customer does not have to take any actions on the data plane side during the execution of this request. On the control plane side, the customer can subscribe to a service such as auto-scaling, as mentioned previously, such that once subscribed the customer does not have to take any action even on the control plan as the auto-scaling service will manage the scaling for the customer.

[0061] As discussed previously, the use of a control plane or service in accordance with various embodiments does not restrict the type of SQL queries that a customer can run, and does not impose any restrictions relating to construction of a schema, such as to be partition ready and not allow queries spanning partitions. Instead, a repository such as a relational database can be provisioned in a computing "cloud" without restricting the users' schema or queries. As commonly known, even though there is a theoretical SQL standard, the SQL quirks, syntaxes and their behaviors (e.g., NULL handling) vary across different relational database engines (e.g., MySQL, Oracle, or Postgres). For at least these reasons, users may wish to choose a relational database engine that is familiar for purposes of programming and operations. Such an approach allows customers to use the same set of database tools that the customers have used previously for tasks such as data modeling, development, and debugging, even when the customers migrate their data stores to the cloud (or elsewhere) via the control plane. Using such an approach, customers are not required to rewrite their application or any operational tools, which lowers the barrier of entry significantly for customers to move data to the cloud.

[0062] A customer's data repositories can be moved to the cloud in one embodiment by running the repositories on compute nodes of a cloud computing environment. Block level storage volumes, such as off-instance storage volumes that persist independently from the life of an instance, can be used with these instances for storing the repository binary, logs and volumes, for example. Such an approach can be advantageous, as the virtualization provides flexibility to quickly and easily scale a compute and storage resources for a repository. Further, such an approach can provide for persistent storage in the cloud.

[0063] As known in the art, relational databases can be run in different modes, such as may include: stand-alone (non-replicated), replicated, or replicated and partitioned. A customer typically makes the choice of which mode to run for a repository based on the availability and scalability needs of the repository and the incurred total cost of ownership (TCO). Some applications and services to not require a repository to be highly available and durable, and may instead utilize a stand-alone repository that is able to tolerate outages on the order of minutes. Other applications and servers can require a repository to be always available, and require the repository to never lose data even in the event of a failure. In this case, the applications and services typically require a replicated database offering. Some users, applications, or services require a massively scalable repository that can partition data across multiple repositories, such that scaling can occur beyond the compute and storage capacity of a single database. To address these different use cases, an approach in accordance with one embodiment offers at least two modes, such as stand-alone and high availability, for each database engine. Some

embodiments also allow customers build their own partitioning layer on top of either stand-alone or high availability repositories.

[0064] As mentioned, the control plane layer can take advantage, or "sit on top," of various basic software frameworks for performing tasks such as: implementing workflows, establishing secure communication channels between the host managers of the data plane and the components of the control plane, installing software on the instances of the data plane, and performing various database backup and recovery procedures.

[0065] For example, a control plane layer can take advantage of a work flow service to manage workflows. As commonly known, a key characteristic of any workflow engine is that the engine enables asynchronous and resumable processing. As discussed above, a workflow can be thought of as a state machine that starts with an initial state and goes through a series of intermediate state transitions by executing different steps of the workflow before reaching the end goal. This end goal can be thought of as the terminal state of a state machine. A workflow service offers the ability to create workflows, and provides hooks to determine the current state of a given workflow and the step(s) to next be executed. The service can store the current state of the state machine, keeping track of the steps which executed successfully and the steps that must be executed to keep the workflow moving. The service does not, in general, actually execute the state transitions for us. The precise tasks of executing the tasks for a workflow will in many embodiments be performed by the "client" components of the workflow.

[0066] Since a control plane can have multiple workflows running in parallel at any given time, and these workflows can be for performing different tasks, the control plane can take advantage of an architecture that is able to schedule multiple workflows and execute multiple activities in parallel. In one embodiment, the control plane includes various worker fleets that are programmed to execute the various workflow tasks. The interaction between these worker fleets and the workflow service will be described with respect to the configuration **600** of FIG. **6**. Each worker host runs three components in this example, a poller component **604**, a decider component **602**, and a dispatcher component **606**. Each host runs a single poller thread, which polls the decider queue for each workflow type. The decider queues are polled in one embodiment based on the priority order of different workflow types. For instance, the decider queues can be polled for a recovery workflow ahead of a repository creation workflow. If, for example, a pollDeciderQueue API returns a non-empty list of decisions, the poller can forward the decisions to the decider component. The decider component then can make a decision regarding the next task to execute in a given workflow, and can call a "startActivity" or similar API to add the task to the activity queue for a given workflow. During the polling, the poller can move to the next task when the decider queue returns empty results, and can poll the activity queue using a "PollActivityQueue" or similar API. If the pollActivityQueue API returns a non-empty list, this list can be handed over to a dispatcher threadpool, which can be tasked with executing the work flow activity. Upon successful completion of the workflow activity, an "ActivityCompleted" or similar API can be called, which will call the workflow service to enqueue in the decision queue.

[0067] Each workflow host in the workflow fleet runs a poller threader, decider, and dispatcher threadpool in this example. The workflows can be defined in one embodiment using an annotations framework, and a workflow application can build these definitions at startup by reading from the appropriate classpath. The workflow service host first registers the list of registered workflows and activity types using APIs of the workflow service such as a "registerWorkflowType" and "registerActivityType" API. Since these APIs are idempotent in this example, each API can be called multiple times from multiple fleets.

[0068] New workflow instances of different workflow types may need to be created for tasks such as repository creation, repository deletion, repository modification, repository recovery, repository backups, user creation, user deletion, password reset, security management, and other such tasks. Each of these workflow instances can be created using a sweeper that constantly sweeps the Admin repository for any changes to be executed, as discussed above with respect to FIG. **2**. For instance, if a user wants to create a new repository then the Web service layer can store the required configuration in Admin DB with a status column such as "PENDING_CHANGES." Each workflow host can runs a sweeper thread that sweeps for any database or security group records with status set to PENDING_CHANGES, and can start a workflow accordingly.

[0069] While in some embodiments the workflow service's "createWorkflow" API can be called directly from the Web service layer as soon as the required configuration is stored in the Admin repository, calling the workflow service directly can result in a two-phase commit style problem. If the workflow is not available from the workflow service, then the Admin repository updates have to be rolled back and CreateDatabaseAPI call not accepted. To avoid such two-phase commit style issues, various embodiments utilize the sweeper architecture that is operable to sweep for new workflow activities to be started by looking at changes to the Admin repository records with a status set to be PENDING_CHANGES, for example.

[0070] Another architecture that can be utilized advantageously relates to providing secure communications to the host managers of the data plane from the components of the host plane. In one embodiment, the workflow and monitoring components of the control plane are constantly communicating with the host managers to perform various tasks (e.g., database maintenance and software installation), as well as to check the status of the various instances and/or repositories. It is important in at least some embodiments that all communications between the control plane and the host managers occur over a secure network that prevents anyone from eavesdropping or issuing unauthorized commands to the host managers.

[0071] In one embodiment, all communication channels to the host managers are secure using a hypertext transfer protocol over a secure socket layer (SSL). Each application server hosting a host manager application can be started using scripts at bootup of an instance. Before starting the application server engine, a script can be executed that generates a self-signed certificate and installs the certificate to enable the SSL communication channel(s). SSL communication is used in one embodiment for encrypting the communication channel and not for client authentication. Client authentication is instead achieved with a public/private key signature embedded in each request, such that in one embodiment all clients sign query string parameters using a private key. This signature can be validated by a custom interceptor, which can be

deployed with the application server for the host manager. Further, a security group (i.e., firewall rules) can be established for each monitored instance in the data plane such that only hosts sitting in a given network or secure group can communicate using the host manager port. Secure information and credentials (such as private keys) can be stored in an appropriate keystore, which can provide for functionality such as key management and rotation.

[0072] Another architecture can be used to assist with software installation and maintenance. Software will generally need to be installed on instances in the data plane instances during various stages of the repository lifecycle. To create a repository, various binaries and/or seeds may need to be installed. After the repository is created, various patches may need to be applied to the database, as well as critical security patches that may need to be installed to the operating system. It thus can be desirable in some embodiments to build upon a flexible software installation architecture or framework that enables the installation of different types of software on the various instances. One of the key requirements of such a framework may be to not only to install new software, but also to provide information on the current set of installed software and the respective versions. It also can be desirable for such a framework to provide functionality to resolve conflicts during installation, verify the success of installation, and provide APIs or other mechanisms to query the list of installed software.

[0073] An installation framework in one embodiment takes advantage of a packet manager such as RPM (Red Hat Package Manager), which enables software to be distributed that is already compiled, such that the software can be installed with a single command. The software can be stored in a "bucket," such that the software can be installed from pre-defined URLs. An RPM or similar installer command can take the package's manifest file (which can be another URL) and the RPM URL as two different parameters. The installed RPMs will be signed by the control plane and/or by Red Hat, and both keys can be installed and maintained for the instances. Software installation in such a situation can be executed by a host manager, which can provide an "installSoftware" or similar API. Such an API can take into account parameters such as Package URL, Manifest URL, RetryCount, ForceInstall flag, and RPM root location). Upon invoking this API, the manifest file will be downloaded, and each item will be compared to the currently installed list of applications. If the item is already installed, then an attempt to reinstall will not be done unless a "force install" or similar flag is specified. To check whether an individual package has been installed, each host manager can provide a "getStatusofSoftware" or similar API. The "installSoftware" host manager API can be idempotent (as workflow may die and retry the step again) and asynchronous (as software installation takes a while to complete). These two aspects can be achieved by using a static object for synchronization, which can prevent a second "installSoftware" call from interfering with the first call.

[0074] An "installSoftware" API in accordance with one embodiment runs in a loop for a maximum of "RetryCount" times. The API also removes any installation files which might exist from a previous failed attempt The API can download a manifest file and determine which items need to be installed, then download the file(s) or package from an appropriate repository or other source. The appropriate RPM files are then processed and installed. A final cleanup step then can be executed, regardless of the presence of any errors in the installation process.

[0075] The installation of the host manager application and any updates also can be managed for each instance, in a way that does not require taking down all the other instances. In one embodiment, an instance starts the application server engine at bootup, and the host manager is installed by calling the application server manager framework to deploy the new host application. As with other communications, the communication can be intercepted and the client authenticated before installing software or pushing updates to a host manager, which can be accomplished without affecting the availability of existing repositories.

[0076] Another aspect that can rely upon an underlying framework relates to repository and data backup. It can be desirable for the control plane to backup customer repositories and instance for various reasons, such as user-initiated backups (which can be performed during the backup time windows) and system-initiated backups during database restore, etc. A single framework can be implemented to handle both instances. To backup a repository, a framework can handle backing up both the data files and any associated log files. While various steps and processes will be described, it should be understood that various steps and approaches can differ from various database engines, such as MySQL and others.

[0077] An approach for backing up data in accordance with one embodiment suspends data operations until shapshots are taken of the appropriate data volumes, and the log files a similarly copied into an appropriate location. For example, an Admin tier can wait for the backup window before initiating a backup procedure. Once inside the backup window, the Admin tier can recreate a workflow that will create a workflow instance for repository backups. In one example, the workflow invokes a "supendDatabaseForBackup" or similar API for the host manager. This API can manage tasks to, for example, flush and lock the tables, suspend I/O to the data volume, create and mount an LVM snapshot for the log volume, create a log position file with the last log position, and start a timer to resume the database. This timer can be used to resume the repository in case the Admin tier hangs up while performing a task, such as taking snapshots, preventing the repository from being accidentally suspended for indefinite period of time. The workflow can poll the host manager for completion of these and/or other such tasks. Once the workflow has confirmed that the host manager has suspended the repository, the workflow can will attempt to backup the data volumes using a set of ordered tasks. For example, the workflow can indicate to create snapshots of each data volume, and verify that the snapshots have been successfully created. A row can be inserted for each snapshot volume in a location such as a backup_data_volumes table. Subsequently, the workflow can invoke a host manager's "resumeDatabaseFromBackup" or similar API. This process can copy the repository logs and log position information to an appropriate storage location, can unmount the log snapshot, remove log snapshot log volume, and unlock all tables. The Admin tier then can create a customer event that indicates the backup has been completed and the repository is again available.

[0078] As discussed, the log files also can be backed up in a similar fashion. The logs can be used to perform tasks such as replaying various transactions in case the data files have to be restored. The engine logs can be copied to an appropriate

storage location, such that previously backed-up log files can be obtained using a simple list command. A host manager will use this result to determine whether there are logs that need to be copied. For example, the host manager can request a bucket list to obtain the list of log files written such that the last sequence can be backed up. If new logs have been created, it can first be determined that the logs are not actively being written to by a database engine, and then the logs can be copied and the copying verified to have been performed successfully.

[0079] Another aspect that can be handled by taking advantage of various frameworks includes the management of various security aspects, such as secure keys and user credentials. Secure information such as secure keys and passwords can be stored using a secure key management system or service, such as is described in co-pending U.S. patent application Ser. No. 12/372,597, Feb. 17, 2009, and entitled "Encryption Key Management," which is hereby incorporated herein by reference. Such a service can contain at least two versions for each credential, an 'OLD' version and the current version. A key can be rotated, for example, by uploading the new value for the key to the service, such as by using the base name for the key, and launching a workflow to propagate that key value to host managers as needed. Once that workflow is terminated successfully, such that each appropriate host has the new credential, the old version of the key can be effectively replaced with the new value. If, for any credential, the old key does not match the new key, that is an indication that a key rotation process is currently underway. A new key rotation is not started if the old key does not match the current key, as such an approach can risk losing credentials that may still be in use. A command line utility or similar interface can be used to push keys to the key management service, which can enforce this check.

[0080] A separate workflow can be defined for updating and/or rotating credentials such as host manager credentials on all host manager instances. Such an approach can utilize the same inputs as a "SendCredentials" or similar API on the each host manager, such as credential type, public key, and optional private key. In place of the credential value, however, the workflow can will accept the name of the key used to store that value in the key management service. The workflow can verify that the current value is different from the new value, and if the values are the same the workflow can terminate with an appropriate error condition. For each active host managed by the control plane, a sub-workflow can be launched that will send the new credential(s) to the host managers on each host. Once all the sub-workflows are complete, the new credential value can replace the old value. Any host that is created or reactivated while this workflow is in progress typically will need to be given the new version of the credential instead of the original.

[0081] A sub-workflow for sending the credentials to a host can utilize require the same inputs as the original workflow, as well as the host name and port for the specific host manager. The sub-workflow can call an "UpdateCredentials" or similar API on the host manager for each specified credential, and can call a "GetCredentials" or similar API on the host manager to verify that the update has completed. The host manager in at least one embodiment will not report the new value for the credential until everything has been done to put the credential in place. If all host managers are not updated within an appropriate period of time, such as two hours (where two hours is configurable and easy to update as required), the workflow can time out and generate an error ticket or other such indication of failure. All the root/admin credentials used by a host manager to communicate with a repository can be stored in the Admin repository in an encrypted form. When rotating the keys to encrypt passwords in the Admin repository, the new keys can be uploaded to the management service and a workflow launched to re-encrypt all appropriate user passwords using the new key. Once that workflow completes successfully, the new encryption key can be used. In addition to changing the encryption keys, this work flow can also change the root password for each database. A workflow for rotating password encryption keys can verify that the new encryption key is different from the old encryption key, encrypt any user passwords for in-flight workflows with the new key, and encrypt the root passwords for any inactive repositories with the new key. Since the repository is inactive, the passwords may not be changed but can be re-encrypted with the new keys. For each active repository, a new root password can be generated and stored in a pending changes field (encrypted with the new key) and a sub-workflow can be launched for updating host manager credentials with the new password. When the sub-workflow completes, the new root password can be written back to the data repository, using the new encryption key. The root database password will not be changed when a repository is inactive, but the workflow that reactivates the repository can change the root password once the repository is active.

[0082] An approach in accordance with one embodiment utilizes a command line utility that wraps a remote command and enforces restrictions on how credentials are rotated. This can guarantee that public and private keys are only rotated in tandem, and that no key is rotated if a previous rotation is still in progress. The utility can verify that the keys were successfully deployed to all hosts and then launch the appropriate workflow in the appropriate control plane environment. A command line utility can use a syntax such as:

```
rotate-rds-key \
    --stage One of Devo, Integ, QA, or Prod\
    --type credential_type\
    --publicKey value for public key \
    --privateKey optional for some types; value for private key
```

Such a utility can fail if the current key is different from the old key on any host in the fleet, there is an error copying the new key to any host in the fleet, or the workflow step could not be started. In the case where the keys are already different, there may be no changes to roll back. The utility can roll back any changes in the other cases and alert the user of cases where the roll back was unsuccessful.

[0083] In a case where a host manager instance dies during an "UpdateCredentials" or similar workflow, allowing the workflow to retry can handle many scenarios with no special logic on the workflow side. Workflow steps other than the step that is updating credentials may receive "MissingCredential" or similar exceptions from which the workflows will need to recover. In such cases, it can be acceptable to send the new credential to the host manager. In the case of a database administration password, the password change may not have taken effect. The workflow step that tries to re-send the root password (as well as any other call to Host Manager that requires the root password from other workflows) can fail with a "MissingCredentials" or similar exception. Workflow steps other than the change password workflow can attempt to

set the credential to the new password and handle any failures. The workflow that is actively trying to change the password can first try sending the new password. If that succeeds, the workflow is done; otherwise, the workflow can retry with the old password followed by the new password. If the host manager receives an "UpdateCredentials" or similar call for the root password where the host manager does not currently have a password in memory, the host manager can attempt to connect to the repository using that password and fail if the connection cannot be established.

[0084] The rotations of various credentials and secure objects are accomplished in many instances without any noticeable impact on the customers. As the web service layer in many embodiments does not use any of these credentials as part or processing customer requests, the customer API calls can continue to proceed as normal. The impact of rotating a credential can vary somewhat depending on the type of credential being rotated. For example, when a new pair of Web services keys is generated, requests signed with the original pair may start to fail. This will only affect workflow steps, in general, which the workflow system can retry for a period of time. The new Web services credentials can be uploaded quickly to the management service in order to minimize the disruption to ongoing workflows. While workflows for generating and propagating new passwords are in progress, workflow boxes can have access to both old and new encryption keys, such that connections can be made to individual repositories and instances while each workflow is in progress. For host manager authentication keys, production hosts can have retry logic in place to retry connections with the old key if requests are being rejected. For RPM signing keys, host managers may be unable to install software for some time if a key is rotated.

[0085] As discussed above, the various embodiments can be implemented in a wide variety of operating environments, which in some cases can include one or more user computers, computing devices, or processing devices which can be used to operate any of a number of applications. User or client devices can include any of a number of general purpose personal computers, such as desktop or laptop computers running a standard operating system, as well as cellular, wireless, and handheld devices running mobile software and capable of supporting a number of networking and messaging protocols. Such a system also can include a number of workstations running any of a variety of commercially-available operating systems and other known applications for purposes such as development and database management. These devices also can include other electronic devices, such as dummy terminals, thin-clients, gaming systems, and other devices capable of communicating via a network.

[0086] Various aspects also can be implemented as part of at least one service or Web service, such as may be part of a service-oriented architecture. Services such as Web services can communicate using any appropriate type of messaging, such as by using messages in extensible markup language (XML) format and exchanged using an appropriate protocol such as SOAP (derived from the "Simple Object Access Protocol"). Processes provided or executed by such services can be written in any appropriate language, such as the Web Services Description Language (WSDL). Using a language such as WSDL allows for functionality such as the automated generation of client-side code in various SOAP frameworks.

[0087] Most embodiments utilize at least one network that would be familiar to those skilled in the art for supporting communications using any of a variety of commercially-available protocols, such as TCP/IP, OSI, FTP, UPnP, NFS, CIFS, and AppleTalk. The network can be, for example, a local area network, a wide-area network, a virtual private network, the Internet, an intranet, an extranet, a public switched telephone network, an infrared network, a wireless network, and any combination thereof

[0088] In embodiments utilizing a Web server, the Web server can run any of a variety of server or mid-tier applications, including HTTP servers, FTP servers, CGI servers, data servers, Java servers, and business application servers. The server(s) also may be capable of executing programs or scripts in response requests from user devices, such as by executing one or more Web applications that may be implemented as one or more scripts or programs written in any programming language, such as Java®, C, C# or C++, or any scripting language, such as Perl, Python, or TCL, as well as combinations thereof. The server(s) may also include database servers, including without limitation those commercially available from Oracle®, Microsoft®, Sybase®, and IBM®.

[0089] The environment can include a variety of data stores and other memory and storage media as discussed above. These can reside in a variety of locations, such as on a storage medium local to (and/or resident in) one or more of the computers or remote from any or all of the computers across the network. In a particular set of embodiments, the information may reside in a storage-area network ("SAN") familiar to those skilled in the art. Similarly, any necessary files for performing the functions attributed to the computers, servers, or other network devices may be stored locally and/or remotely, as appropriate. Where a system includes computerized devices, each such device can include hardware elements that may be electrically coupled via a bus, the elements including, for example, at least one central processing unit (CPU), at least one input device (e.g., a mouse, keyboard, controller, touch screen, or keypad), and at least one output device (e.g., a display device, printer, or speaker). Such a system may also include one or more storage devices, such as disk drives, optical storage devices, and solid-state storage devices such as random access memory ("RAM") or read-only memory ("ROM"), as well as removable media devices, memory cards, flash cards, etc.

[0090] Such devices also can include a computer-readable storage media reader, a communications device (e.g., a modem, a network card (wireless or wired), an infrared communication device, etc.), and working memory as described above. The computer-readable storage media reader can be connected with, or configured to receive, a computer-readable storage medium, representing remote, local, fixed, and/or removable storage devices as well as storage media for temporarily and/or more permanently containing, storing, transmitting, and retrieving computer-readable information. The system and various devices also typically will include a number of software applications, modules, services, or other elements located within at least one working memory device, including an operating system and application programs, such as a client application or Web browser. It should be appreciated that alternate embodiments may have numerous variations from that described above. For example, customized hardware might also be used and/or particular elements might be implemented in hardware, software (including por-

table software, such as applets), or both. Further, connection to other computing devices such as network input/output devices may be employed.

[0091] Storage media and computer readable media for containing code, or portions of code, can include any appropriate media known or used in the art, including storage media and communication media, such as but not limited to volatile and non-volatile, removable and non-removable media implemented in any method or technology for storage and/or transmission of information such as computer readable instructions, data structures, program modules, or other data, including RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disk (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 a system device. Based on the disclosure and teachings provided herein, a person of ordinary skill in the art will appreciate other ways and/or methods to implement the various embodiments.

[0092] The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense. It will, however, be evident that various modifications and changes may be made thereunto without departing from the broader spirit and scope of the invention as set forth in the claims.

What is claimed is:

1. A data management system, comprising:

one or more processors;

memory including instructions that, upon being executed by the one or more processors, cause the one or more processors to:

send a first request, via a control interface of a data store provisioning service, for access to a provisioned data store instance;

receive information for accessing the provisioned data store instance via a data interface of the data store provisioning service in response to the first request;

migrate data from a non-provisioned data store instance of the data management system to the provisioned data store instance via the data interface; and

configure the provisioned data store instance as at least one of a replica or a partition of the non-provisioned data store instance,

wherein the provisioned data store instance automatically scales according to a specified compute and storage capacity.

2. The data management system of claim **1**, wherein the instructions, upon being executed, further cause the one or more processors to:

send a second request, via the control interface, for at least one of upgrading a data store engine of the provisioned data store instance, downgrading the data store engine, converting the provisioned data store instance to a replicated instance, backing up data of the provisioned data store instance, restoring the provisioned data store instance, rebooting the provisioned data store instance, hibernating the provisioned data store instance, resuming the provisioned data store instance from hibernation, or deleting the provisioned data store instance.

3. The data management system of claim **1**, wherein the information for accessing the provisioned data instance includes at least one of a Domain Name System (DNS) name or network address information.

4. A computer-implemented method for integrating a provisioned data store instance with a data management system, comprising:

under control of one or more computer systems configured with executable instructions,

sending a first request, via a control interface of a data store provisioning service, for access to the provisioned data store instance;

receive information for accessing the provisioned data store instance via a data interface of the data store provisioning service in response to the first request; and

configuring the provisioned data store instance as at least one of a replica or a partition of a non-provisioned data store instance,

wherein the provisioned data store instance automatically scales according to a specified compute and storage capacity.

5. The computer-implemented method of claim **4**, wherein the control interface corresponds to a control environment, the provisioned data store instance corresponds to a data environment, and the control environment is separate from the data environment.

6. The computer-implemented method of claim **4**, further comprising:

migrating data from the non-provisioned data store instance of the data management system to the provisioned data store instance via the data interface.

7. The computer-implemented method of claim **4**, further comprising:

sending a second request, via the control interface, for at least one of upgrading a data store engine of the provisioned data store instance, downgrading the data store engine, converting the provisioned data store instance to a replicated instance, backing up data of the provisioned data store instance, restoring the provisioned data store instance, rebooting the provisioned data store instance, hibernating the provisioned data store instance, resuming the provisioned data store instance from hibernation, or deleting the provisioned data store instance.

8. The computer-implemented method of claim **4**, wherein a user sets, via the control interface, the specified compute and storage capacity.

9. The computer-implemented method of claim **4**, wherein the data store provisioning service sets the specified compute and storage capacity.

10. The computer-implemented method of claim **4**, wherein the information for accessing the provisioned data store instance includes at least one of a DNS name or network address information.

11. The computer-implemented method of claim **4**, further comprising:

selecting a data store engine for the provisioned data store instance via the control interface.

12. The computer-implemented method of claim **4**, further comprising:

managing user credentials via the control interface.

13. The computer-implemented method of claim **4**, further comprising:

pointing an application server to the provisioned data store instance.

14. A non-transitory computer-readable storage medium for integrating a provisioned data store instance with a data store management system including instructions that, upon being executed by a processor, cause the processor to:

send a first request, via a control interface of a data store provisioning service, for access to the provisioned data store instance;

receive information for accessing the provisioned data store instance via a data interface of the data store provisioning service in response to the first request;

migrate data from a non-provisioned data store instance of the data management system to the provisioned data store instance via the data interface; and

configure the provisioned data store instance as at least one of a replica or a partition of the non-provisioned data store instance,

wherein the provisioned data store instance automatically scales according to a specified compute and storage capacity.

15. The non-transitory computer-readable storage medium of claim **14**, wherein the control interface corresponds to a control environment, the provisioned data store instance corresponds to a data environment, and the control environment is separate from the data environment.

16. The non-transitory computer-readable storage medium of claim **14**, wherein a user sets, via the control interface, the specified compute and storage capacity.

17. The non-transitory computer-readable storage medium of claim **14**, wherein the data store provisioning service sets the specified compute and storage capacity.

18. The non-transitory computer-readable storage medium of claim **14**, wherein the instructions, upon being executed, further cause the processor to:

select a data store engine for the provisioned data store instance via the control interface.

19. The non-transitory computer-readable storage medium of claim **14**, wherein the instructions, upon being executed, further cause the processor to:

manage user credentials via the control interface.

20. The non-transitory computer-readable storage medium of claim **14**, wherein the instructions, upon being executed, further cause the processor to:

point an application server to the provisioned data store instance.

* * * * *