November 2022

CONTAINER PATCHING AUTOMATION

BALASAHEB RAOSAHEB DENGALE
VISA

DILSHAD T
VISA

CHARAN RAMIREDDY
VISA

PANNEER PERUMAL
VISA

CONTAINER PATCHING AUTOMATION

VISA

INVENTORS:

- **BALASAHEB RAOSAHEB DENGALE**
- **DILSHAD T**
- **CHARAN RAMIREDDY**
- **PANNEER PERUMAL**

TECHNICAL FIELD

[0001] The present subject matter is, in general, related to Kubernetes patching systems, and particularly, to a method and a system for automating patching of a container.

BACKGROUND

[0002] A container is a unit of software that packages code and its dependencies so the application runs quickly and reliably across computing environments. The containers virtualize an operating system and run anywhere, from a private data center to the public cloud or even on a developer's personal user devices. The container patching is a process of repairing a vulnerability or a flaw in a software and an application after the release. In other words, the container patching is a process of updating the software and the operating system and subsequently, addressing the security vulnerabilities within a program or a product.

[0003] In the conventional methods, the patching is performed manually by the users or the workers who continuously monitor Virtual Machines (VMs) and coordinate with Asknow change to operation team. Subsequently, the operation team coordinates with multiple application teams and patch all clusters with agreed maintenance window. This process exhibits downtime for the cluster and the application in some of the cases, which is a drawback of the conventional container patching methods.

[0004] In view of the above, there is a need for an automated container patching system that reduces the downtime for the cluster and the application in the container patching.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] The accompanying drawings, which are incorporated in and constitute a part of this disclosure, illustrate exemplary embodiments and, together with the description, explain the disclosed principles. In the figures, the left-most digit(s) of a reference number identifies the figure in which the reference number first appears. The same numbers are used throughout the figures to reference like features and components. Some embodiments of device or system and/or methods in accordance with embodiments of the present subject matter are now described, by way of example only, and with reference to the accompanying figures, in which:

[0006] **Fig. 1** shows a flowchart illustrating a method of automatically patching the container platform in accordance with some embodiments consistent with the present disclosure.

[0007] **Fig. 2** shows a flowchart illustrating a method for updating a database (VMS) in accordance with some embodiments consistent with the present disclosure.

[0008] **Fig. 3** shows a flowchart illustrating a method for validating an application or a software in accordance with some embodiments consistent with the present disclosure.

[0009] **Fig. 4** illustrates a block diagram of an exemplary computer system for implementing embodiments consistent with the present disclosure.

[0010] The figures depict embodiments of the disclosure for purposes of illustration only. One skilled in the art will readily recognize from the following description that alternative embodiments of the structures and methods illustrated herein may be employed without departing from the principles of the disclosure described herein.

DESCRIPTION OF THE DISCLOSURE

[0011] In the present document, the word "exemplary" is used herein to mean "serving as an example, instance, or illustration." Any embodiment or implementation of the present subject matter described herein as "exemplary" is not necessarily to be construed as preferred or advantageous over other embodiments.

[0012] While the disclosure is susceptible to various modifications and alternative forms, specific embodiment thereof has been shown by way of example in the drawings and will be described in detail below. It should be understood, however that it is not intended to limit the disclosure to the particular forms disclosed, but on the contrary, the disclosure is to cover all modifications, equivalents, and alternative falling within the spirit and the scope of the disclosure.

[0013] The terms “comprises”, “comprising”, or any other variations thereof, are intended to cover a non-exclusive inclusion, such that a setup, device, or method that comprises a list of components or steps does not include only those components or steps but may include other components or steps not expressly listed or inherent to such setup or device or method. In other words, one or more elements in a device or system or apparatus proceeded by “comprises… a” does not, without more constraints, preclude the existence of other elements or additional elements in the device or system or apparatus.

[0014] The terms "an embodiment", "embodiment", "embodiments", "the embodiment", "the embodiments", "one or more embodiments", "some embodiments", and "one embodiment" mean "one or more (but not all) embodiments of the invention(s)" unless expressly specified otherwise.

[0015] The terms "including", "comprising", “having” and variations thereof mean "including but not limited to", unless expressly specified otherwise.

[0016] The present disclosure relates to a method and an automation system for automatically patching the container. The automation system in the present disclosure increases security by keeping the platform patched. Further, the automation system reduces the downtime in the container patching process. As a result, the present disclosure provides with complete end-to-end hands of patching with control in a place. Furthermore, the present disclosure performs pre-validations and post-validations to make sure that there is no impact before or after patching the nodes to the cluster. The automation system stops the patching process through the automatic control if suspicious failures crosses predefined limitations. Finally, the automation system checks in place with graceful drain and API health checks.

[0017] **Fig. 1** shows a flowchart illustrating a method of automatically patching the container platform in accordance with some embodiments consistent with the present disclosure.

[0018] As illustrated in **Fig. 1**, the method **100** includes one or more blocks illustrating a method for automatic patching of the container The order in which the method **100** is described is not intended to be construed as a limitation, and any number of the described method blocks can be combined in any order to implement the method. Additionally, individual blocks may be deleted from the methods without departing from the scope of the subject matter described herein. Furthermore, the method can be implemented in any suitable hardware, software, firmware, or combination thereof.

[0019] At block **102**, the method **100** includes performing pre-validations from Operating System (OS) perspective. In an implementation, the pre-validations may comprise, without limiting to, operations such as gathering facts, validating pre-request information such as cluster, type master etc., validating connection and elevation and clustering.

[0020] At block **104**, the method **100** includes pre-validating the cluster/node from Kubernetes perspective and drain node. In an implementation, the pre-validating may include, without limitation, validating application or cluster before patching, performing ready checks, and performing backup and the like. The pre-validation is performed based on the Kubernetes and the drain node.

[0021] At block **106**, the method **100** includes patching the node and rebooting the node. In an embodiment, the nodes may be updated before rebooted.

[0022] At block **108**, the method **100** includes performing the post-validation operations from the OS perspective. In an implementation, the post-validating may include, without limitation, validating application or cluster after patching. The post-validation is performed based on the cardon node.

[0023] At block **110**, the method **100** includes post-validating the cluster or the node and re-establishing the cluster. In an implementation, the post-validation of the cluster or node may include, without limitation, cleaning up the nodes and performing health checks for the cluster.

[0024] **Fig. 2** shows a flowchart illustrating a method for updating a database (VMS) in accordance with some embodiments consistent with the present disclosure.

[0025] At step **201**, a user may login to Vulnerability Management Self-service (VMS) using predefined login credentials through a user device **105**. As an example, the user may be, without limitation, an operation team member, a worker in an organization, an engineer and the like. As an example, the user credentials may include a username and a password authentication token that is bound to a particular user for login. In an implementation, the user device may include, without limitation, a smartphone, a tablet, a laptop, a desktop or a smart device associated with the user.

[0026] At step **203**, the VMS may schedule a remediation. It will create Asknow change to follow ITIL process and schedule it after 24 hours. There will be sufficient controls in place to trigger this change and send proper notifications to the support teams defined. This step will trigger underline automations which is specific to container platform patching.

[0027] At step **205**, the VMS may select the servers. As an example, the VMS may select a Virtual Machine (VM) server to host or run the VMs, which runs various operating systems and acts as computing platforms on their own through emulation and virtualization.

[0028] At step **207**, the VMS may select container packages. As an example, the container packages may include a software that packages code and its dependencies to run application fast and reliably across all the computing platforms.

[0029] At step **209**, the VMS may submit a request and coordinate with an Asknow change as shown in step **211**.

[0030] At step **213**, the servers move to a maintenance mode, and the VMS may release one server at a time as shown in step **215**.

[0031] At step **217**, the released server may trigger the Ansible playbook. As an example, the Ansible playbook is a list of tasks that automatically execute the tasks against the hosts. Each module within the Ansible playbook performs a specific task, and each module contains metadata which determines when and where a task is executed, as well as which user executes it.

[0032] At step **219**, a status of execution of the tasks is checked. If the status of the execution of the tasks is completed, then the VMS (database) is updated as shown in step **221**. If the status of the execution of the tasks is still in progress, then the process of verifying the status is continued.

[0033] **Fig. 3** shows a flowchart illustrating a method for validating application or software in accordance with some embodiments consistent with the present disclosure.

[0034] At step **301**, a basic pre-request validation is performed. In an implementation, the basic pre-request validation process comprises gathering facts, validating pre-request information, validating connection and elevation and clustering. As an example, the pre-requisite information may include, without limitation, a cluster, a type of cluster, a master node information and the like.

[0035] At step **303**, patching of prescripts is performed. In an implementation, the prescripts are runbooks which can be attached to update deployment. The prescripts are run before the

patching occurs. For example, this includes sending planned maintenance notifications by an email.

[0036] At step **305**, it is verified whether the patching prescripts is successful or not. If the patching prescripts are not successful, then the validation process is stopped. Alternatively, if the patching prescripts are successfully performed, then an application is pre-validated as shown in step **307**.

[0037] At step **309**, it is verified whether the pre-validation of the application is successful or not. If the pre-validation of the application is not successful (or a failure), then the validation process is stopped. Alternatively, if the pre-validation of the application is successful, then the server is patched as shown in step **311**. As an example, the patch server or server patching is a process of updating the server's software to fix errors, updating software versions or enhancing performance and security on the server.

[0038] At step **313**, patching of postscripts is performed. In an implementation, the postscripts are runbooks which run after the deployment of update.

[0039] At step **315**, it is verified whether the patching postscripts process is successful or not. If the process is not successful, then the validation process is stopped. Whereas, if the patching postscripts is successful, then the application is post-validated as shown in step **317**.

[0040] At step **319**, it is verified whether the post-validation of the application is successful or not. If the post-validation process is not successful, then the validation process is stopped. Alternatively, if the post-validation process is successful, then the basic post-validation is performed on the application as shown in step **321**.

General computer system:

[0041] **Fig. 4** illustrates a block diagram of an exemplary computer system for implementing embodiments consistent with the present disclosure.

[0042] In an embodiment, the computer system **400** may be used to implement the system. The computer system **400** may include a central processing unit ("CPU" or "processor") **402**. The processor **402** may include at least one data processor connected to a network interface **403** to communicate with a communication network **409**. The processor **402** may include specialized

processing units such as integrated system (bus) controllers, memory management control units, floating point units, graphics processing units, digital signal processing units, etc.

[0043] The processor **402** may be disposed in communication with one or more Input/Output (I/O) devices (**410** and **411**) via I/O interface **401**. The I/O interface **401** employ communication protocols/methods such as, without limitation, audio, analog, digital, monoaural, Radio Corporation of America (RCA) connector, stereo, IEEE-1394 high speed serial bus, serial bus, Universal Serial Bus (USB), infrared, Personal System/2 (PS/2) port, Bbayonet Neill-Concelman (BNC) connector, coaxial, component, composite, Digital Visual Interface (DVI), High-Definition Multimedia Interface (HDMI), Radio Frequency (RF) antennas, S-Video, Video Graphics Array (VGA), IEEE 802.11b/g/n/x, Bluetooth, cellular e.g., Code-Division Multiple Access (CDMA), High-Speed Packet Access (HSPA+), Global System for Mobile communications (GSM), Long-Term Evolution (LTE), Worldwide Interoperability for Microwave access (WiMax), or the like, etc.

[0044] Using the I/O interface **401**, the computer system **400** may communicate with one or more I/O devices such as input devices **410** and output devices **411**. For example, the input devices 410 may be an antenna, keyboard, mouse, joystick, (infrared) remote control, camera, card reader, fax machine, dongle, biometric reader, microphone, touch screen, touchpad, trackball, stylus, scanner, storage device, transceiver, video device/source, etc. The output devices **413** may be a printer, fax machine, video display (e.g., Cathode Ray Tube (CRT), Liquid Crystal Display (LCD), Light-Emitting Diode (LED), plasma, Plasma Display Panel (PDP), Organic Light-Emitting Diode display (OLED) or the like), audio speaker, etc.

[0045] In some embodiments, the processor **402** may be disposed in communication with a communication network **409** via a network interface **403**. The network interface **403** may communicate with the communication network **409**. The network interface **403** may employ connection protocols including, without limitation, direct connect, ethernet (e.g., twisted pair 10/100/1000 Base T), Transmission Control Protocol/Internet Protocol (TCP/IP), token ring, IEEE 802.11a/b/g/n/x, etc. The communication network 409 may include, without limitation, a direct interconnection, Local Area Network (LAN), Wide Area Network (WAN), wireless network (e.g., using Wireless Application Protocol), the Internet, etc. A user **101** may communicate with the communication system **400** through the communication network **409** using a user device **105**. The network interface **403** may employ connection protocols include, but not limited to, direct connect, ethernet (e.g., twisted pair 10/100/1000 Base T),

Transmission Control Protocol/Internet Protocol (TCP/IP), token ring, IEEE 802.11a/b/g/n/x, etc.

[0046] The communication network **409** includes, but is not limited to, a direct interconnection, a Peer-to-Peer (P2P) network, Local Area Network (LAN), Wide Area Network (WAN), wireless network (e.g., using Wireless Application Protocol), the Internet, Wi-Fi and such. The communication network 409 may either be a dedicated network or a shared network, which represents an association of the different types of networks that use a variety of protocols, for example, Hypertext Transfer Protocol (HTTP), Transmission Control Protocol/Internet Protocol (TCP/IP), Wireless Application Protocol (WAP), etc., to communicate with each other. Further, the communication network 409 may include a variety of network devices, including routers, bridges, servers, computing devices, storage devices, etc.

[0047] In some embodiments, the processor **402** may be disposed in communication with a memory **405** (e.g., RAM, ROM, etc. not shown in Fig. 3) via a storage interface **404**. The storage interface 404 may connect to memory **405** including, without limitation, memory drives, removable disc drives, etc., employing connection protocols such as, Serial Advanced Technology Attachment (SATA), Integrated Drive Electronics (IDE), IEEE-1394, Universal Serial Bus (USB), fiber channel, Small Computer Systems Interface (SCSI), etc. The memory drives may further include a drum, magnetic disc drive, magneto-optical drive, optical drive, Redundant Array of Independent Discs (RAID), solid-state memory devices, solid-state drives, etc.

[0048] The memory **405** may store a collection of program or database components, including, without limitation, user interface **406**, an operating system **407**, etc. In some embodiments, computer system **400** may store user/application data, such as, the data, variables, records, etc., as described in this disclosure. Such databases may be implemented as fault-tolerant, relational, scalable, secure databases such as Oracle or Sybase.

[0049] The operating system **407** may facilitate resource management and operation of the computer system **400**. Examples of operating systems include, without limitation, Apple™ Macintosh ™ OS X™, UNIX™, Unix-like system distributions (e.g., Berkeley Software Distribution (BSD), FreeBSD™, Net BSD™, Open BSD™, etc.), Linux distributions (e.g., Red Hat™, Ubuntu™, K-Ubuntu™, etc.), International Business Machines (IBM™) OS/2™,

Microsoft Windows™ (XP™, Vista/7/8, etc.), Apple iOS™, Google Android™, Blackberry™ operating system (OS), or the like.

[0050] In some embodiments, the computer system **400** may implement web browser 308 stored program components. Web browser 408 may be a hypertext viewing application, such as Microsoft™ Internet Explorer™, Google Chrome™, Mozilla Firefox™, Apple™ Safari™, etc. Secure web browsing may be provided using secure hypertext transport protocol (HTTPS), Secure Sockets Layer (SSL), Transport Layer Security (TLS), etc. Web browsers 408 may utilize facilities such as AJAX, DHTML, Adobe™ Flash, Javascript, Application Programming Interfaces (APIs), etc. In some embodiments, the computer system 300 may implement a mail server stored program component. The mail server may be an Internet mail server such as Microsoft Exchange, or the like. The mail server may utilize facilities such as ASP, ActiveX, ANSI C++/C#, Microsoft .NET, Common Gateway Interface (CGI) scripts, Java, JavaScript, PERL, PHP, Python, WebObjects, etc. The mail server may utilize communication protocols such as Internet Message Access Protocol (IMAP), Messaging Application Programming Interface (MAPI), Microsoft Exchange, Post Office Protocol (POP), Simple Mail Transfer Protocol (SMTP), or the like.

[0051] In some embodiments, the computer system **400** may implement a mail client stored program component. The mail client may be a mail viewing application, such as Apple Mail, Microsoft Entourage, Microsoft Outlook, Mozilla Thunderbird, etc.

[0052] Furthermore, one or more computer-readable storage media may be utilized in implementing embodiments consistent with the present disclosure. A computer-readable storage medium refers to any type of physical memory on which information or data readable by a processor may be stored. Thus, a computer-readable storage medium may store instructions for execution by one or more processors, including instructions for causing the processor(s) to perform steps or stages consistent with the embodiments described herein. The term “computer-readable medium” should be understood to include tangible items and exclude carrier waves and transient signals, i.e., be non-transitory. Examples include Random Access Memory (RAM), Read-Only Memory (ROM), volatile memory, non-volatile memory, hard drives, Compact Disc (CD) ROMs, DVDs, flash drives, disks, and any other known physical storage media.

[0053] The described operations may be implemented as a method, system or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The described operations may be implemented as code maintained in a "non-transitory computer readable medium", where a processor may read and execute the code from the computer readable medium. The processor is at least one of a microprocessor and a processor capable of processing and executing the queries. A non-transitory computer readable medium may include media such as magnetic storage medium (e.g., hard disk drives, floppy disks, tape, etc.), optical storage (CD-ROMs, DVDs, optical disks, etc.), volatile and non-volatile memory devices (e.g., EEPROMs, ROMs, PROMs, RAMs, DRAMs, SRAMs, Flash Memory, firmware, programmable logic, etc.), etc. Further, non-transitory computer-readable media may include all computer-readable media except for a transitory. The code implementing the described operations may further be implemented in hardware logic (e.g., an integrated circuit chip, Programmable Gate Array (PGA), Application Specific Integrated Circuit (ASIC), etc.).

[0054] The illustrated steps are set out to explain the exemplary embodiments shown, and it should be anticipated that ongoing technological development will change the manner in which particular functions are performed. These examples are presented herein for purposes of illustration, and not limitation. Further, the boundaries of the functional building blocks have been arbitrarily defined herein for the convenience of the description. Alternative boundaries can be defined so long as the specified functions and relationships thereof are appropriately performed. Alternatives (including equivalents, extensions, variations, deviations, etc., of those described herein) will be apparent to persons skilled in the relevant art(s) based on the teachings contained herein. Such alternatives fall within the scope and spirit of the disclosed embodiments. It must also be noted that as used herein, the singular forms "a," "an," and "the" include plural references unless the context clearly dictates otherwise.

[0055] Furthermore, one or more computer-readable storage media may be utilized in implementing embodiments consistent with the present disclosure. A computer readable storage medium refers to any type of physical memory on which information or data readable by a processor may be stored. Thus, a computer readable storage medium may store instructions for execution by one or more processors, including instructions for causing the processor(s) to perform steps or stages consistent with the embodiments described herein. The term "computer readable medium" should be understood to include tangible items and exclude

carrier waves and transient signals, i.e., are non-transitory. Examples include Random Access Memory (RAM), Read-Only Memory (ROM), volatile memory, non-volatile memory, hard drives, CD ROMs, DVDs, flash drives, disks, and any other known physical storage media.

[0056] Finally, the language used in the specification has been principally selected for readability and instructional purposes, and it may not have been selected to delineate or circumscribe the inventive subject matter. Accordingly, the disclosure of the embodiments of the disclosure is intended to be illustrative, but not limiting, of the scope of the disclosure.

[0057] With respect to the use of substantially any plural and/or singular terms herein, those having skill in the art can translate from the plural to the singular and/or from the singular to the plural as is appropriate to the context and/or application. The various singular/plural permutations may be expressly set forth herein for sake of clarity.

CONTAINER PATCHING AUTOMATION

ABSTRACT

[0058] The present disclosure relates to a method and an automation system for automatically patching a software container. In an embodiment, the present disclosure discloses the aspect of performing pre-validations from Operating System (OS) perspective, and pre-validating cluster/node from Kubernetes perspective and a drain node. Further, the present disclosure discloses patching the node and rebooting the node and performing post-validation from the OS perspective. Additionally, the present disclosure discloses the aspect of post-validating the cluster/node and re-establishing the cluster.

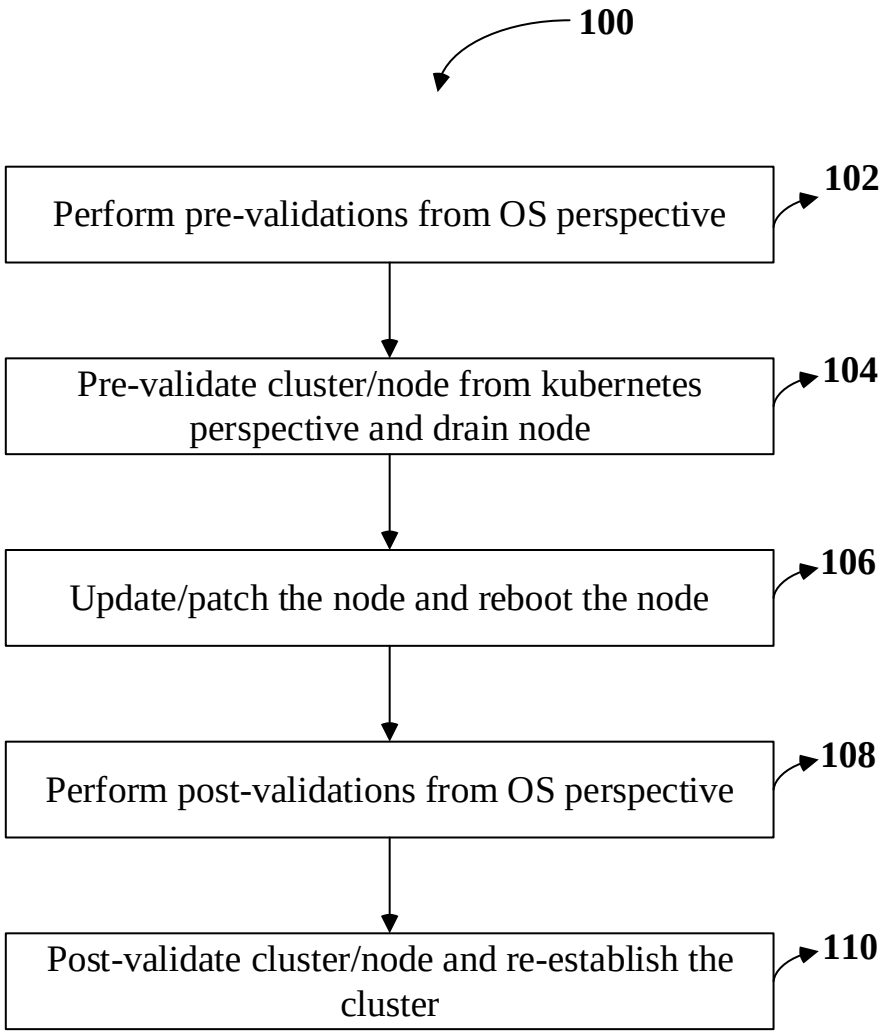
100
Perform pre-validations from OS perspective
102
Pre-validate cluster/node from kubernetes perspective and drain node
104
Update/patch the node and reboot the node
106
Perform post-validations from OS perspective
108
Post-validate cluster/node and re-establish the cluster
110

Fig. 1

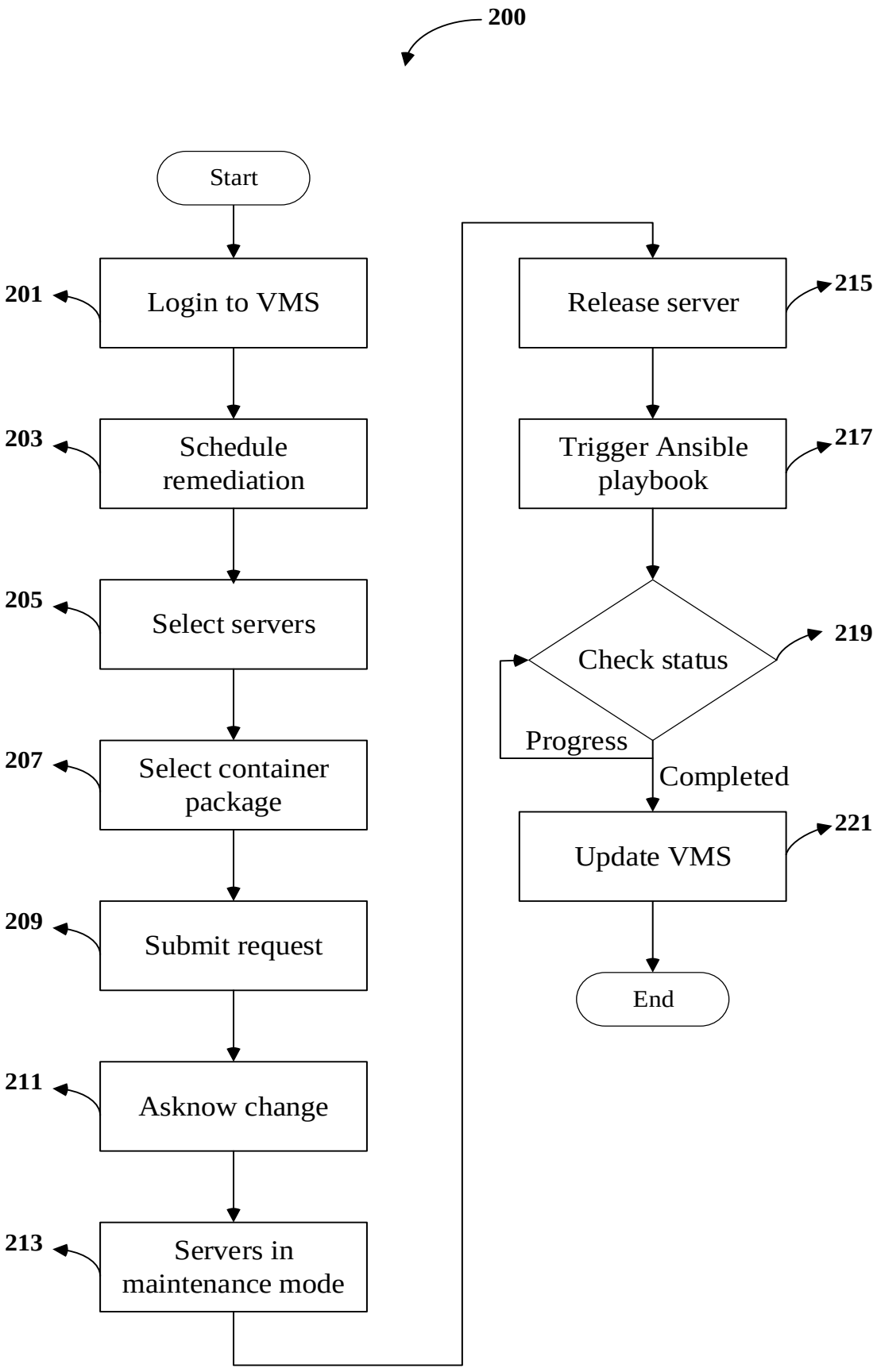
200
Start
201
Login to VMS
203
Schedule remediation
205
Select servers
207
Select container package
209
Submit request
211
Asknow change
213
Servers in maintenance mode
215
Release server
217
Trigger Ansible playbook
219
Check status
Progress
Completed
221
Update VMS
End

Fig. 2

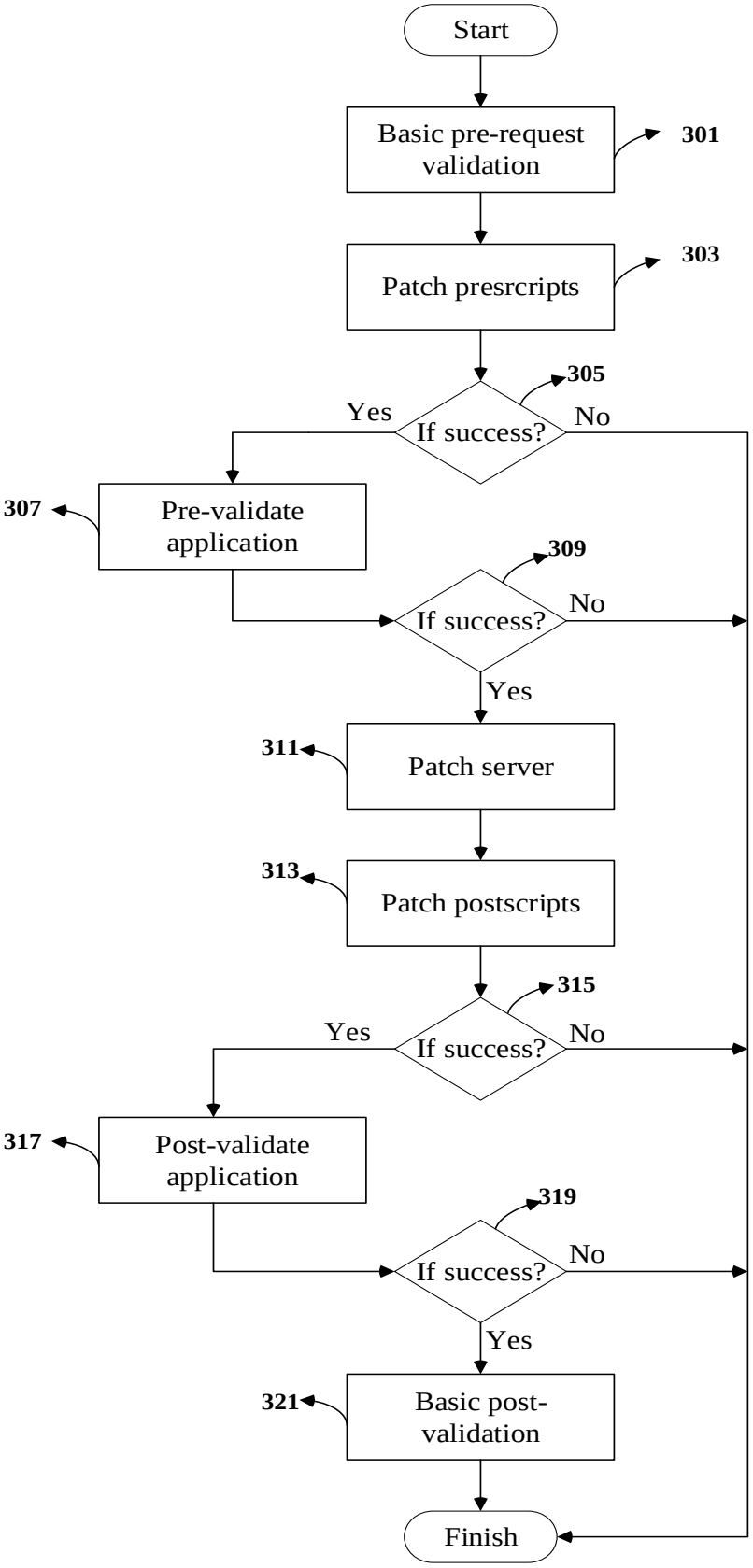
Start
Basic pre-request validation
301
Patch presrcripts
303
If success?
305
Yes
No
Pre-validate application
307
If success?
309
No
Yes
Patch server
311
Patch postscripts
313
If success?
315
Yes
No
Post-validate application
317
If success?
319
No
Yes
Basic post-validation
321
Finish

Fig. 3

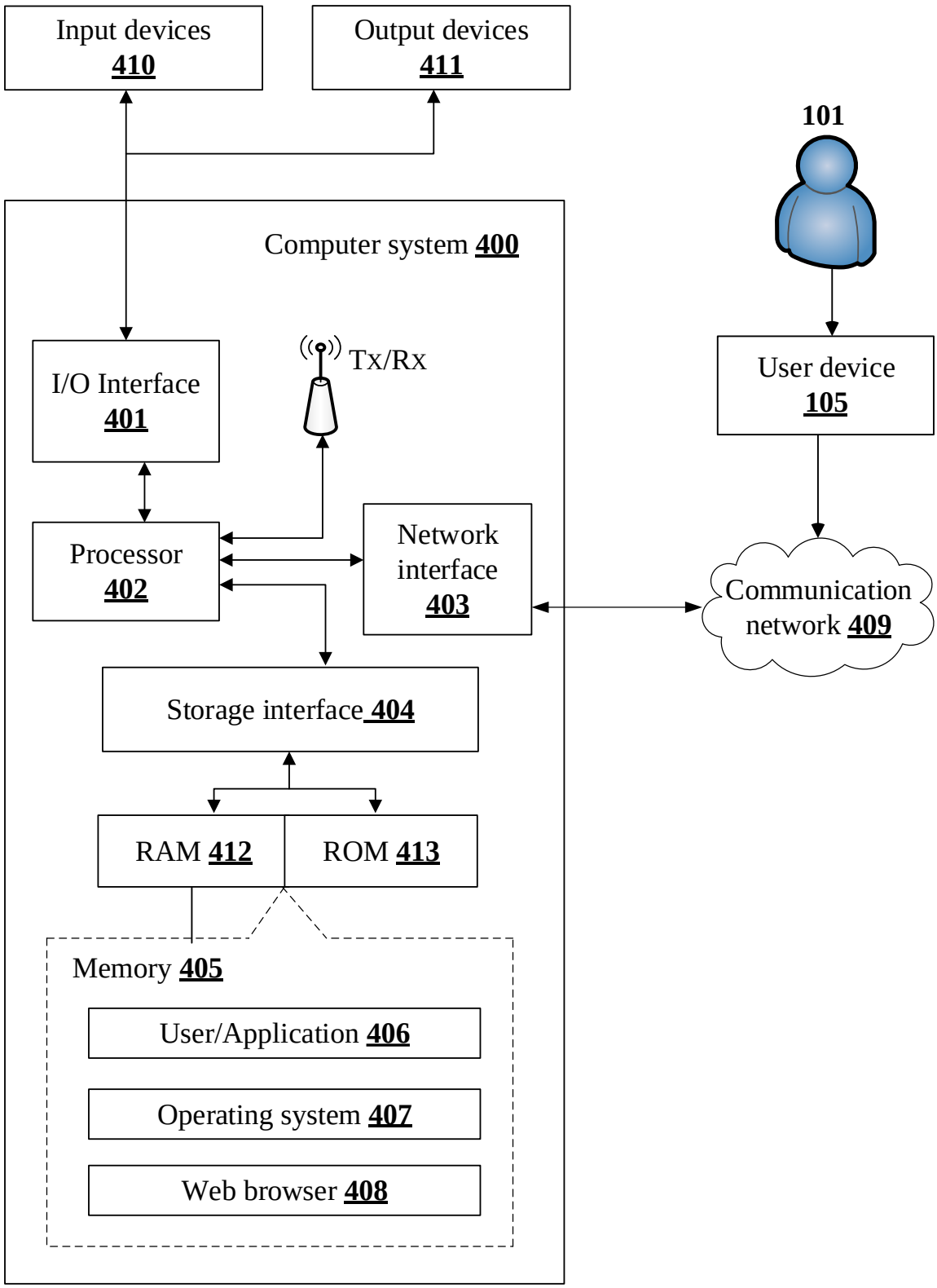
Input devices
410
Output devices
411
101
Computer system 400
I/O Interface
401
Tx/Rx
User device
105
Processor
402
Network
interface
403
Communication
network 409
Storage interface 404
RAM 412
ROM 413
Memory 405
User/Application 406
Operating system 407
Web browser 408

Fig. 4