(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
11 August 2022 (11.08.2022)

WIPO | PCT

(10) International Publication Number
WO 2022/166549 A1

(51) International Patent Classification:
H04L 9/32 (2006.01)

(21) International Application Number:
PCT/CN2022/071600

(22) International Filing Date:
12 January 2022 (12.01.2022)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
17/167,398 04 February 2021 (04.02.2021) US

(71) Applicant: **INTERNATIONAL BUSINESS MACHINES CORPORATION** [US/US]; New Orchard Road, Armonk, New York 10504 (US).

(71) Applicant *(for MG only)*: **IBM (CHINA) CO., LIMITED** [CN/CN]; 7F, Bldg. 10, Zhang Jiang Innovation Park, 399 Keyuan Road, Zhang Jiang High-Tech Campus, Pudong New Area, Shanghai 201203 (CN).

(72) Inventors: **KITAHARA, Hirokuni**; 19-21, Nohinbashi-Hakozakicho, Chuo-ku, Tokyo 103-8510 (JP). **KUDO, Ruriko**; 19-21, Nohinbashi-Hakozakicho, Chuo-ku, Tokyo 103-8510 (JP). **GAJANANAN, Kugamoorthy**; 19-21, Nohinbashi-Hakozakicho, Chuo-ku, Tokyo 103-8510 (JP). **WATANABE, Yuji**; 19-21, Nohinbashi-Hakozakicho, Chuo-ku, Tokyo 103-8510 (JP).

(74) Agent: **LIU, SHEN & ASSOCIATES**; 10th Floor, Building 1, 10 Caihefang Road, Haidian District, Beijing 100080 (CN).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, IT, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW,

WO 2022/166549 A1

(54) Title: CLUSTER RESOURCE SIGNATURE VERIFICATION

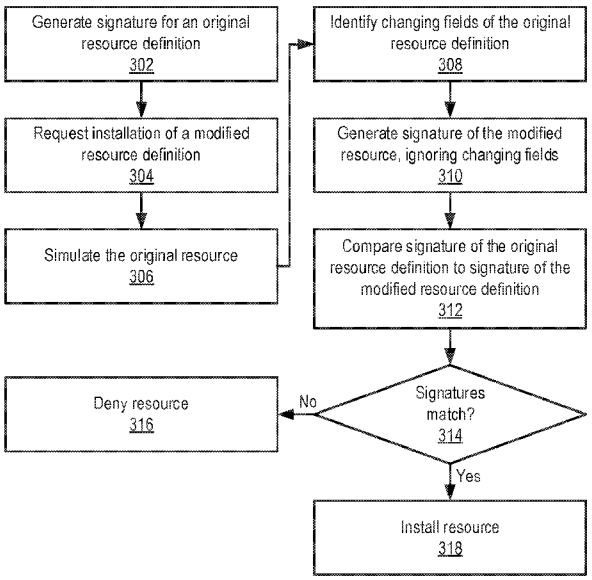


FIG. 3

(57) Abstract: Methods and systems for verifying a resource definition include simulating an original resource definition to identify at least one change that is made to the original resource definition by a management service. A signature of a received resource definition is generated, omitting portions of the received resource definition that correspond to the at least one identified change. The signature of the received resource definition is compared to a signature of the original resource definition to find a match and to verify the received resource definition. The received resource definition is implemented, responsive to finding the match.

[Continued on next page]

SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

CLUSTER RESOURCE SIGNATURE VERIFICATION

BACKGROUND

[0001] The present invention generally relates to cluster management and security, and, more particularly, to signing static parts of a resource definition.

[0002] Cloud computing systems now run a wide variety of mission-critical applications, including financial and governmental institutions. Such systems may be implemented in a variety of ways, including any of several container orchestration systems. The sensitive nature of such applications brings with it a need for strong security, for example to prevent resources from being tampered with prior to implementation.

SUMMARY

[0003] A method for verifying a resource definition includes simulating an original resource definition to identify at least one change that is made to the original resource definition by a management service. A signature of a received resource definition is generated, omitting portions of the received resource definition that correspond to the at least one identified change. The signature of the received resource definition is compared to a signature of the original resource definition to find a match and to verify the received resource definition. The received resource definition is implemented, responsive to finding the match.

[0004] A system for verifying a resource definition includes a hardware processor and a memory that stores computer program code. When the computer program code is executed by the hardware processor, it causes the hardware processor to simulate an original resource definition to identify at least one change that is made to the original resource definition by a management service, to generate a signature of a received resource definition, omitting portions of the received resource definition that correspond to the at least one identified change, to compare the signature of the received resource definition to a signature of the original resource definition to find a match and to verify the received resource definition, and to implement the received resource definition, responsive to finding the match.

[0005] These and other features and advantages will become apparent from the following detailed description of illustrative embodiments thereof, which is to be read in connection with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] The following description will provide details of preferred embodiments with reference to the following figures wherein:

[0007] FIG. 1 is a block diagram of a Platform as a Service (PaaS) system that includes resource signature verification, in accordance with an embodiment of the present invention;

[0008] FIG. 2 is a block diagram of an individual processing node in a PaaS system that performs signature verification on received resource definitions, in accordance with an embodiment of the present invention;

[0009] FIG. 3 is a block/flow diagram of a method for verifying a received resource definition against a signature of a corresponding original resource definition, in accordance with an embodiment of the present invention;

[0010] FIG. 4 is a block/flow diagram of a method for installing and using requested resources in a PaaS system that includes signature verification, in accordance with an embodiment of the present invention;

[0011] FIG. 5 is a block diagram of a signature verification system that simulates an original resource definition to identify a whitelist of changes that occur during handling of the resource definition, and that verifies a received resource definition using that whitelist, in accordance with an embodiment of the present invention;

[0012] FIG. 6 is a diagram that illustrates different kinds of resource definitions and the changes that may occur within them, in accordance with an embodiment of the present invention;

[0013] FIG. 7 is a diagram of a cloud computing environment according to the present principles; and

[0014] FIG. 8 is a diagram of abstraction model layers according to the present principles.

DETAILED DESCRIPTION

[0015] A given cluster resource may change during use. For example, an original resource may be different from the resource that is actually implemented. This presents a challenge, when an application requires that the resource be digitally signed before installation. In such a case, the installation may fail, because the signed resource is not identical to the resource that is being installed.

[0016] To address this problem, a resource whitelist may be generated to identify the parts of the resource that will not change. This may be accomplished by simulating the resource multiple times to identify those attributes within the resource that remain constant, and those which change. A signature file can then be generated that is based on the static portions of the resource. At the time of installation, the signature can be verified by looking only to those attributes within the resource that are static.

[0017] Referring now to FIG. 1, a diagram of a platform as a system (PaaS) system 100 is shown. A user 102 has a workload for execution on a distribution computing platform. The user 102 communicates with a manager system 104 and provides information regarding the workload, including the number and type of processing nodes 106 that will be needed to execute the workload. The information provided to the manager system 104 includes, for example, a number of processing nodes 106, a processor type, an operating system, an execution environment, storage capacity, random access memory capacity, network bandwidth, and any other points that may be needed for the workload. The user 102 can furthermore provide particular images or containers to the manager system 104 for storage in a registry there. It should be understood that the PaaS system 100 can include many thousands of processing nodes 106, each of which can be idle or busy in accordance with the workloads being executed by the PaaS system 100 at any given time. It should also be understood that, although a single manager system 104 is shown, there can be multiple such manager systems 104, with multiple registries distributed across the PaaS system 100. While this can reduce the problem of limited bandwidth at the registry by providing alternative image sources, very large deployments will still cause delays. Furthermore, although the manger system 104 is shown as being a single device, it should be understood that scheduling can be performed at a separate location from the image registry.

[0018] Before and during execution of the workload, the manager system 104 determines which processing nodes 106 will use which images. The manager system 104 thus distributes images to the processing nodes 106 according to its own internal logic, for example based on node and resource availability at the time of provisioning. The manager system 104 and the PaaS system 100 can handle multiple different workloads from multiple different users 102, such that the availability of particular resources will depend on what is happening in the PaaS system 100 generally.

[0019] Provisioning, as the term is used herein, refers to the process by which resources in a PaaS system 100 are allocated to a user 102 and are prepared for execution. Thus, provisioning includes the determinations made by the manager system 104 as to which processing elements 106 will be used for the workload as well as the transmission of images and any configuration steps that are needed to

prepare the processing nodes 106 for execution of the workload. The processing nodes 106 selected by the manager system 104 then install the images and perform any configuration needed for the workload. Execution can then proceed, triggered by the manager system 104, by the user 102, or by any other appropriate source.

[0020] It is to be understood in advance that, although this disclosure includes a detailed description of cloud computing, implementation of the teachings recited herein are not limited to a cloud computing environment. Rather, embodiments of the present invention are capable of being implemented in conjunction with any other type of computing environment now known or later developed.

[0021] Referring now to FIG. 2, additional detail on a processing node 106 is shown. The processing node 106 includes a hardware processor 202, a memory 204, and a network interface 206. The network interface 206 is configured to communicate with the manager system 104 and with other processing nodes 106 as needed, using any appropriate wired or wireless communications medium and protocol. The processing node 106 also includes one or more functional modules that may, in some embodiments, be implemented as software that is stored in the memory 204 and that is executed by the hardware processor 202. In other embodiments, one or more of the functional modules may be implemented as one or more discrete hardware components in the form of, e.g., application-specific integrated chips or field programmable gate arrays.

[0022] The processing node 106 may include one or more containers 208, for example that have been distributed to it by the manager system 104. It is specifically contemplated that each container 208 represents a distinct operating environment. The containers 208 each include a set of software applications, configuration files, workload datasets, and any other information or software needed to execute a specific workload. The containers 208 are stored in memory 204 and are instantiated and decommissioned by the container orchestration engine 210 as needed. It should be understood that, as a general matter, an operating system of the processing node 106 exists outside the containers 208. Thus, each container 208 interfaces with the same operating system kernel, reducing the overhead needed to execute multiple containers simultaneously. The containers 208 meanwhile generally have no communication with one another outside of specifically agreed-to interfaces, reducing security concerns.

[0023] A container 208 may access a distributed filesystem that is shared between multiple processing nodes 106. The container 208 may communicate with the container orchestration engine

210 to mount a persistent volume 218. Upon receipt of a request from a container 208, the container orchestration engine 210 communicates with the storage provision engine 212. When the storage provision engine 212 receives a directory request, it provisions the distributed filesystem with a local directory mount point, informs the quota manager 214 and sync engine 216 to complete quota/isolation and information syncing processes, and provides a directory back to the container orchestration engine 210.

[0024] The quota manager 214 interacts with the storage provision engine 202 to set quotas and isolation for the persistent volume directories 218. The quota manager 214 gets information from the storage provision engine 212 regarding the successful mount of the distributed filesystem and can be implemented as, for example, a kernel module, where system calls monitor the directory size and isolation of the persistent volumes 218. The quota manager 214 interacts with the storage provision engine 212 to address volumes overrunning their quotas. For example, the quota manager 214 can perform actions based on policy, such as halting writing operations, when the directory's size exceeds a predetermined quota. The quota amount for an individual directory is determined when the persistent volume 218 is created. Isolation is used to ensure that the persistent volume 218 cannot be used by other processes.

[0025] A sync engine 216 communicates with sync engines 216 from other processing nodes 106. The sync engine 216 communicates information relating to how the persistent volumes 218 are being used, including quota information and isolation information. Quota and isolation is set on a per-host basis, so the sync engine 216 communicates that information to other hosts to guarantee consistence across different nodes.

[0026] The persistent volumes 218 are represented herein as directories in the local filesystem of the processing node 106. These directories can then be mount points 220 for the distributed filesystem 106. As the containers 208 access the persistent volumes 218, filesystem operations are transmitted to and from the distributed filesystem 106, making the data in the distributed filesystem 106 accessible to the containers. In particular, it should be noted that multiple containers 208 can share access to the distributed filesystem in this manner.

[0027] In some cases, where signature verification is needed before a resource can be installed, the container orchestration engine 210 communicates with a signature verifier 222. The signature verifier compares the resource that is to be installed (e.g., a description of a container 208) with a signature for the resource. If the signature matches, then the resource may be installed and executed. If not, then

the signature verifier 222 denies the request, and the resource is prevented from being installed or executed.

[0028] Referring now to FIG. 3, a method for verifying a resource definition is shown. Block 302 generates a signature for an original resource definition. In one specific, and non-limiting example, a KUBERNETES® system may be used for container management. In such an embodiment, resources may be defined using text-based files in the "YAML" format. The following is an exemplary YAML resource definition that may be used in a KUBERNETES system:

[0029] abiVersion: v1

[0030] kind: ConfigMap

[0031] data:

[0032] sample-data: abc

[0033] Security-enabled: true

[0034] It should be understood that this resource definition is purely exemplary, and that any appropriate resource may be used instead. The signature may be generated by any appropriate signing algorithm, for example using a signing key. Such algorithms may include, for example, RSA, DSA, and ECDSA. Additionally, the use of other container management systems is also contemplated.

[0035] Block 304 receives a request for installing the resource, for example using a resource definition file that has been modified by a KUBERNETES system. For example, the modified resource definition may be changed as follows, with differences being indicated:

[0036] abiVersion: v1

[0037] kind: ConfigMap

[0038] metadata: # This line has been added

[0039] uid: a2b4e4f2-3ca8-a996 # This line has been added

[0040] data:

[0041] sample-data: abc

[0042] Security-enabled: true

[0043] default-config: def # This line has been added

[0044] If a signature was formed for the original resource definition, and the signature verifier 222 later attempts to verify the signature for the entirety of the modified resource definition, then the verification will fail, because the modified resource definition is not identical. To address this problem, the signature verifier 222 ignores those lines of the modified resource definition which are expected

to be altered by the system. Excluding such lines, the signature verifier 222 generates a signature of the remaining lines and compares it to the signature of the original resource definition. If there is a match, then the modified resource definition may be verified.

[0045] While these changeable lines and fields may be predetermined by analysis of the KUBERNETES system software, software updates may cause changes to how the resource definitions are handled, and may add new types of information that were not previously contemplated. To address this, the changes may be analyzed on a per-resource basis, at runtime, to build a whitelist of current changes.

[0046] Thus, block 306 may simulate the original resource definition, for example using the KUBERNETES "DryRun" function, to generate one or more simulated resource definitions, which may be changed in a manner that is similar to the changes that would be made by the KUBERNETES system for an actual implementation. By simulating the resource, rather than relying on a static whitelist, changes to the underlying software can be accommodated immediately. By comparing the simulated resource definitions to the original resource definitions, block 308 can find lines that have been altered in, or added to, the simulated resource definitions. At least two simulations may be run, but greater numbers may provide better results.

[0047] Block 310 generates a signature for the modified resource definition, ignoring any lines that have were indicated in block 308 as being changed. Block 312 compares the signature of the original resource definition to the signature of the modified resource definition. If block 314 finds that the two signatures do not match, then block 316 denies the resource. In this case, the signature verifier 222 instructs the container orchestration engine 210 to deny the request to install the resource. If the two signatures do match, then block 318 causes the signature verifier 222 to instruct the container orchestration engine 210 to install the resource.

[0048] Referring now to FIG. 4, a method for performing a task using a PaaS system, such as the system 100 described above. Block 402 generates a workload. For example, the workload may include any appropriate software to perform any purpose, such as processing financial transactions or performing statistical computations on large datasets. Block 404 determines what resources will be needed to perform the workload. For example, block 404 may determine how many instantiations may be needed, what operating systems and supporting software may be used, and what containers may be used. Block 406 sends the request for these resources to the manager system 104.

[0049] Block 408 verifies the resource signatures as described above. If the verification is successful, block 410 installs the requested resources, and block 412 performs the workload using the installed resources. In the event that block 408 fails to verify the resource signatures, then a user 102 may go back to audit their resource requests, to identify and address discrepancies. A failure to verify the signature may indicate an attempt to tamper with the resources by an attacker.

[0050] The present invention may be a system, a method, and/or a computer program product at any possible technical detail level of integration. The computer program product may include a computer readable storage medium (or media) having computer readable program instructions thereon for causing a processor to carry out aspects of the present invention.

[0051] The computer readable storage medium can be a tangible device that can retain and store instructions for use by an instruction execution device. The computer readable storage medium may be, for example, but is not limited to, an electronic storage device, a magnetic storage device, an optical storage device, an electromagnetic storage device, a semiconductor storage device, or any suitable combination of the foregoing. A non-exhaustive list of more specific examples of the computer readable storage medium includes the following: a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), a static random access memory (SRAM), a portable compact disc read-only memory (CD-ROM), a digital versatile disk (DVD), a memory stick, a floppy disk, a mechanically encoded device such as punch-cards or raised structures in a groove having instructions recorded thereon, and any suitable combination of the foregoing. A computer readable storage medium, as used herein, is not to be construed as being transitory signals *per se*, such as radio waves or other freely propagating electromagnetic waves, electromagnetic waves propagating through a waveguide or other transmission media (e.g., light pulses passing through a fiber-optic cable), or electrical signals transmitted through a wire.

[0052] Computer readable program instructions described herein can be downloaded to respective computing/processing devices from a computer readable storage medium or to an external computer or external storage device via a network, for example, the Internet, a local area network, a wide area network and/or a wireless network. The network may comprise copper transmission cables, optical transmission fibers, wireless transmission, routers, firewalls, switches, gateway computers and/or edge servers. A network adapter card or network interface in each computing/processing device receives computer readable program instructions from the network and forwards the computer readable

program instructions for storage in a computer readable storage medium within the respective computing/processing device.

[0053] Computer readable program instructions for carrying out operations of the present invention may be assembler instructions, instruction-set-architecture (ISA) instructions, machine instructions, machine dependent instructions, microcode, firmware instructions, state-setting data, configuration data for integrated circuitry, or either source code or object code written in any combination of one or more programming languages, including an object oriented programming language such as Smalltalk, C++, or the like, and procedural programming languages, such as the "C" programming language or similar programming languages. The computer readable program instructions may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider). In some embodiments, electronic circuitry including, for example, programmable logic circuitry, field-programmable gate arrays (FPGA), or programmable logic arrays (PLA) may execute the computer readable program instructions by utilizing state information of the computer readable program instructions to personalize the electronic circuitry, in order to perform aspects of the present invention.

[0054] Aspects of the present invention are described herein with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems), and computer program products according to embodiments of the invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer readable program instructions.

[0055] These computer readable program instructions may be provided to a processor of a computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks. These computer readable program instructions may also be stored in a computer readable storage medium that can direct a computer, a programmable data processing apparatus, and/or other devices to function in a particular manner, such that the computer readable storage medium

having instructions stored therein comprises an article of manufacture including instructions which implement aspects of the function/act specified in the flowchart and/or block diagram block or blocks.

[0056] The computer readable program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other device to cause a series of operational steps to be performed on the computer, other programmable apparatus or other device to produce a computer implemented process, such that the instructions which execute on the computer, other programmable apparatus, or other device implement the functions/acts specified in the flowchart and/or block diagram block or blocks.

[0057] Reference in the specification to "one embodiment" or "an embodiment" of the present invention, as well as other variations thereof, means that a particular feature, structure, characteristic, and so forth described in connection with the embodiment is included in at least one embodiment of the present invention. Thus, the appearances of the phrase "in one embodiment" or "in an embodiment", as well any other variations, appearing in various places throughout the specification are not necessarily all referring to the same embodiment.

[0058] It is to be appreciated that the use of any of the following "/", "and/or", and "at least one of", for example, in the cases of "A/B", "A and/or B" and "at least one of A and B", is intended to encompass the selection of the first listed option (A) only, or the selection of the second listed option (B) only, or the selection of both options (A and B). As a further example, in the cases of "A, B, and/or C" and "at least one of A, B, and C", such phrasing is intended to encompass the selection of the first listed option (A) only, or the selection of the second listed option (B) only, or the selection of the third listed option (C) only, or the selection of the first and the second listed options (A and B) only, or the selection of the first and third listed options (A and C) only, or the selection of the second and third listed options (B and C) only, or the selection of all three options (A and B and C). This may be extended, as readily apparent by one of ordinary skill in this and related arts, for as many items listed.

[0059] The flowchart and block diagrams in the Figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods, and computer program products according to various embodiments of the present invention. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of instructions, which comprises one or more executable instructions for implementing the specified logical function(s). In some alternative implementations, the functions noted in the blocks may occur out of the order noted in the Figures.

For example, two blocks shown in succession may, in fact, be accomplished as one step, executed concurrently, substantially concurrently, in a partially or wholly temporally overlapping manner, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts or carry out combinations of special purpose hardware and computer instructions.

[0060] As employed herein, the term "hardware processor subsystem" or "hardware processor" can refer to a processor, memory, software or combinations thereof that cooperate to perform one or more specific tasks. In useful embodiments, the hardware processor subsystem can include one or more data processing elements (e.g., logic circuits, processing circuits, instruction execution devices, etc.) The one or more data processing elements can be included in a central processing unit, a graphics processing unit, and/or a separate processor- or computing element-based controller (e.g., logic gates, etc.) The hardware processor subsystem can include one or more on-board memories (e.g., caches, dedicated memory arrays, read only memory, etc.) In some embodiments, the hardware processor subsystem can include one or more memories that can be on or off board or that can be dedicated for use by the hardware processor subsystem (e.g., ROM, RAM, basic input/output system (BIOS), etc.)

[0061] In some embodiments, the hardware processor subsystem can include and execute one or more software elements. The one or more software elements can include an operating system and/or one or more applications and/or specific code to achieve a specified result.

[0062] In other embodiments, the hardware processor subsystem can include dedicated, specialized circuitry that performs one or more electronic processing functions to achieve a specified result. Such circuitry can include one or more application-specific integrated circuits (ASICs), FPGAs, and/or PLAs.

[0063] These and other variations of a hardware processor subsystem are also contemplated in accordance with embodiments of the present invention.

[0064] Referring now to FIG. 5, additional detail on the signature verifier 222 is shown. It should be understood that the signature verifier 222 may be implemented as part of the processing node 106 as described above, may be implemented as part of the manager system 104, may be an independent system running on separate hardware, or may be implemented in any other appropriate fashion. In the event that the signature verifier 222 is running on separate hardware, then it may include a hardware

processor, a memory, and a communications interface (not shown). If the signature verifier 222 is collocated with the manager system 104 or a processing node 106, then it may share the hardware resources of those systems.

[0065] The signature verifier 222 receives and stores a copy of the original resource definition 502. This original resource definition 502 may differ from the original resource definition, due to discrepancies introduced by the PaaS system, for example by KUBERNETES software. A resource simulator 504 (e.g., the KUBERNETES "DryRun" function) runs one or more simulations of the original resource definition 502 to generate simulated resources 506. A whitelist generator 508 analyzes the simulated resources 506 to identify parts of the definition that change when it is handled by the PaaS system. The whitelist generator 508 thereby generates a signature whitelist 510 that may identify lines and fields in the resource definition that change during normal operation.

[0066] The requested resource definition 511 is processed by a signature generator 512, for example using the same function as was used to generate the original resource signature 516, using the signature whitelist 510 to omit portions of the requested resource definition 511 that change during normal operation. The signature generator 512 thereby produces a requested resource signature 514. The requested resource signature 514 is compared with the original resource signature 516 at signature comparison 518. A verifier 520 makes a determination, based on any discrepancies between the two signatures, as to whether the requested resource definition is legitimate and may be installed.

[0067] Referring now to FIG. 6, a comparison of various resource definitions is shown. The original resource definition is shown as block 502, which shows the information that specifies the resource. Blocks 506 shows a set of simulated resource definitions that are based on the original resource definition 502. The portions that change during the simulation are shown as underlined; these additions are introduced by the PaaS system for its own purposes, and do not generally affect the functioning of the ultimate resource installation.

[0068] A requested resource definition is shown in block 511. During verification, those additions shown in the simulated definitions 506 are located within the requested resource definition 511 and are omitted during signature verification. Excluding these additions, the remaining contents of the requested resource definition 511 match the contents of the original resource definition 502, so the verification succeeds.

[0069] In contrast, a malicious resource definition 602 is shown. As with the requested resource definition 511, any identified changeable lines are omitted during signature verification. However,

another line has been introduced by a malicious actor, shown in bold. If this malicious resource definition 602 is submitted for signature verification, the mismatch that the introduced line causes will cause verification to fail, preventing installation of the malicious resource.

[0070] Cloud computing is a model of service delivery for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, network bandwidth, servers, processing, memory, storage, applications, virtual machines, and services) that can be rapidly provisioned and released with minimal management effort or interaction with a provider of the service. This cloud model may include at least five characteristics, at least three service models, and at least four deployment models.

[0071] Characteristics are as follows:

[0072] On-demand self-service: a cloud consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with the service's provider.

[0073] Broad network access: capabilities are available over a network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, laptops, and PDAs).

[0074] Resource pooling: the provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to demand. There is a sense of location independence in that the consumer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter).

[0075] Rapid elasticity: capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to quickly scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time.

[0076] Measured service: cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

[0077] Service Models are as follows:

[0078] Software as a Service (SaaS): the capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based e-mail). The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

[0079] Platform as a Service (PaaS): the capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including networks, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations.

[0080] Infrastructure as a Service (IaaS): the capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components (e.g., host firewalls).

[0081] Deployment Models are as follows:

[0082] Private cloud: the cloud infrastructure is operated solely for an organization. It may be managed by the organization or a third party and may exist on-premises or off-premises.

[0083] Community cloud: the cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on-premises or off-premises.

[0084] Public cloud: the cloud infrastructure is made available to the general public or a large industry group and is owned by an organization selling cloud services.

[0085] Hybrid cloud: the cloud infrastructure is a composition of two or more clouds (private, community, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load-balancing between clouds).

[0086] A cloud computing environment is service oriented with a focus on statelessness, low coupling, modularity, and semantic interoperability. At the heart of cloud computing is an infrastructure that includes a network of interconnected nodes.

[0087] Referring now to FIG. 7, illustrative cloud computing environment 50 is depicted. As shown, cloud computing environment 50 includes one or more cloud computing nodes 10 with which local computing devices used by cloud consumers, such as, for example, personal digital assistant (PDA) or cellular telephone 54A, desktop computer 54B, laptop computer 54C, and/or automobile computer system 54N may communicate. Nodes 10 may communicate with one another. They may be grouped (not shown) physically or virtually, in one or more networks, such as Private, Community, Public, or Hybrid clouds as described hereinabove, or a combination thereof. This allows cloud computing environment 50 to offer infrastructure, platforms and/or software as services for which a cloud consumer does not need to maintain resources on a local computing device. It is understood that the types of computing devices 54A-N shown in FIG. 7 are intended to be illustrative only and that computing nodes 10 and cloud computing environment 50 can communicate with any type of computerized device over any type of network and/or network addressable connection (e.g., using a web browser).

[0088] Referring now to FIG. 8, a set of functional abstraction layers provided by cloud computing environment 50 (FIG. 7) is shown. It should be understood in advance that the components, layers, and functions shown in FIG. 8 are intended to be illustrative only and embodiments of the invention are not limited thereto. As depicted, the following layers and corresponding functions are provided:

[0089] Hardware and software layer 60 includes hardware and software components. Examples of hardware components include: mainframes 61; RISC (Reduced Instruction Set Computer) architecture based servers 62; servers 63; blade servers 64; storage devices 65; and networks and networking components 66. In some embodiments, software components include network application server software 67 and database software 68.

[0090] Virtualization layer 70 provides an abstraction layer from which the following examples of virtual entities may be provided: virtual servers 71; virtual storage 72; virtual networks 73, including virtual private networks; virtual applications and operating systems 74; and virtual clients 75.

[0091] In one example, management layer 80 may provide the functions described below. Resource provisioning 81 provides dynamic procurement of computing resources and other resources that are utilized to perform tasks within the cloud computing environment. Metering and Pricing 82 provide

cost tracking as resources are utilized within the cloud computing environment, and billing or invoicing for consumption of these resources. In one example, these resources may include application software licenses. Security provides identity verification for cloud consumers and tasks, as well as protection for data and other resources. User portal 83 provides access to the cloud computing environment for consumers and system administrators. Service level management 84 provides cloud computing resource allocation and management such that required service levels are met. Service Level Agreement (SLA) planning and fulfillment 85 provide pre-arrangement for, and procurement of, cloud computing resources for which a future requirement is anticipated in accordance with an SLA.

[0092] Workloads layer 90 provides examples of functionality for which the cloud computing environment may be utilized. Examples of workloads and functions which may be provided from this layer include: mapping and navigation 91; software development and lifecycle management 92; virtual classroom education delivery 93; data analytics processing 94; transaction processing 95; and signature verification 96.

[0093] Having described preferred embodiments of cluster resource signature verification (which are intended to be illustrative and not limiting), it is noted that modifications and variations can be made by persons skilled in the art in light of the above teachings. It is therefore to be understood that changes may be made in the particular embodiments disclosed which are within the scope of the invention as outlined by the appended claims. Having thus described aspects of the invention, with the details and particularity required by the patent laws, what is claimed and desired protected by Letters Patent is set forth in the appended claims.

WHAT IS CLAIMED IS:

1. A computer-implemented method for verifying a resource definition, comprising:

simulating an original resource definition to identify at least one change that is made to the original resource definition by a management service;

generating a signature of a received resource definition, omitting portions of the received resource definition that correspond to the at least one identified change;

comparing the signature of the received resource definition to a signature of the original resource definition to find a match and to verify the received resource definition; and

implementing the received resource definition, responsive to finding the match.

2. The method of claim 1, wherein the original resource definition and the received resource definition are defined for a container manager.

3. The method of claim 2, wherein the at least one change is introduced by the container manager during handling of any resource definition.

4. The method of claim 2, wherein simulating the original resource definition includes running the original resource definition using a DryRun function of the container manager.

5. The method of claim 1, wherein simulating the original resource definition includes running a simulation of the original resource definition at least twice.

6. The method of claim 1, wherein generating the signature of the received resource definition includes removing the portions of the received resource definition that correspond to the at least one identified change from the received resource definition before generating a signature from remaining portions.

7. The method of claim 1, further comprising receiving the received resource definition from a user, wherein simulating the simulating an original resource definition is performed after receiving the received resource definition.

8. The method of claim 7, wherein further comprising generating the signature of the original resource definition, wherein generating the signature of the original resource definition is performed before receiving the received resource definition.

9. The method of claim 1, wherein the original resource definition and the received resource definition are text-based YAML files.

10. The method of claim 1, wherein implementing the received resource definition includes installing a corresponding resource on a local processing node.

11. A computer program product for verifying a resource definition, the computer program product comprising a computer readable storage medium having program instructions embodied therewith, the program instructions executable by a hardware processor to cause the hardware processor to:

simulate an original resource definition to identify at least one change that is made to the original resource definition by a management service;

generate a signature of a received resource definition, omitting portions of the received resource definition that correspond to the at least one identified change;

compare the signature of the received resource definition to a signature of the original resource definition to find a match and to verify the received resource definition; and

implement the received resource definition, responsive to finding the match.

12. The computer program product of claim 11, wherein the original resource definition and the received resource definition are defined for a container manager.

13. The computer program product of claim 12, wherein the at least one change is introduced by the container manager during handling of any resource definition.

14. The computer program product of claim 12, wherein simulating the original resource definition includes running the original resource definition using a DryRun function of the container manager.

15. The computer program product of claim 11, wherein simulating the original resource definition includes running a simulation of the original resource definition at least twice.

16. The computer program product of claim 11, wherein generating the signature of the received resource definition includes removing the portions of the received resource definition that correspond to the at least one identified change from the received resource definition before generating a signature from remaining portions.

17. The computer program product of claim 11, wherein the program instructions are further executable by a hardware processor to cause the hardware processor to receive the received resource definition from a user, wherein simulating the simulating an original resource definition is performed after receiving the received resource definition.

18. The computer program product of claim 17, wherein the program instructions are further executable by a hardware processor to cause the hardware processor to generate the signature of the original resource definition, wherein generating the signature of the original resource definition is performed before receiving the received resource definition.

19. The computer program product of claim 11, wherein implementing the received resource definition includes installing a corresponding resource on a local processing node.

20. A system for verifying a resource definition, comprising:

a hardware processor; and

a memory that stores computer program code, which, when executed by the hardware processor, causes the hardware processor to:

simulate an original resource definition to identify at least one change that is made to the original resource definition by a management service;

generate a signature of a received resource definition, omitting portions of the received resource definition that correspond to the at least one identified change;

compare the signature of the received resource definition to a signature of the original resource definition to find a match and to verify the received resource definition; and

implement the received resource definition, responsive to finding the match.

5

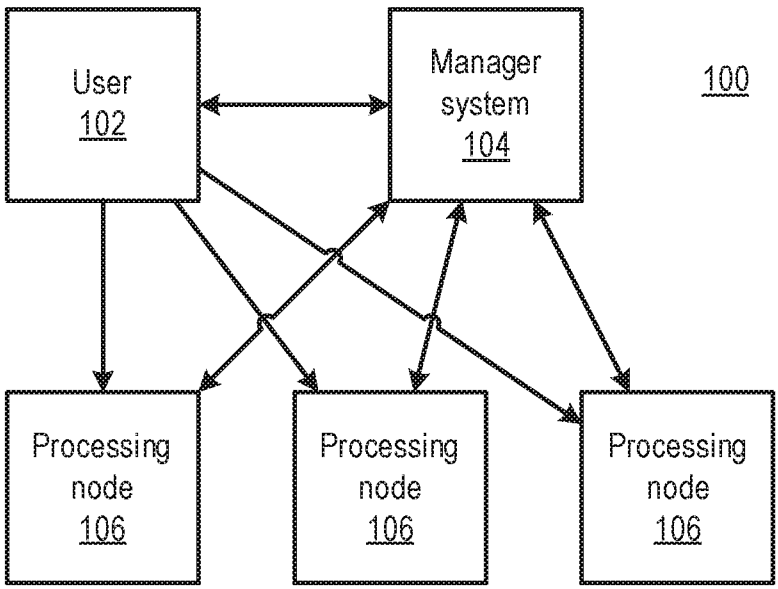

User
102
Manager
system
104
100
Processing
node
106
Processing
node
106
Processing
node
106

FIG. 1

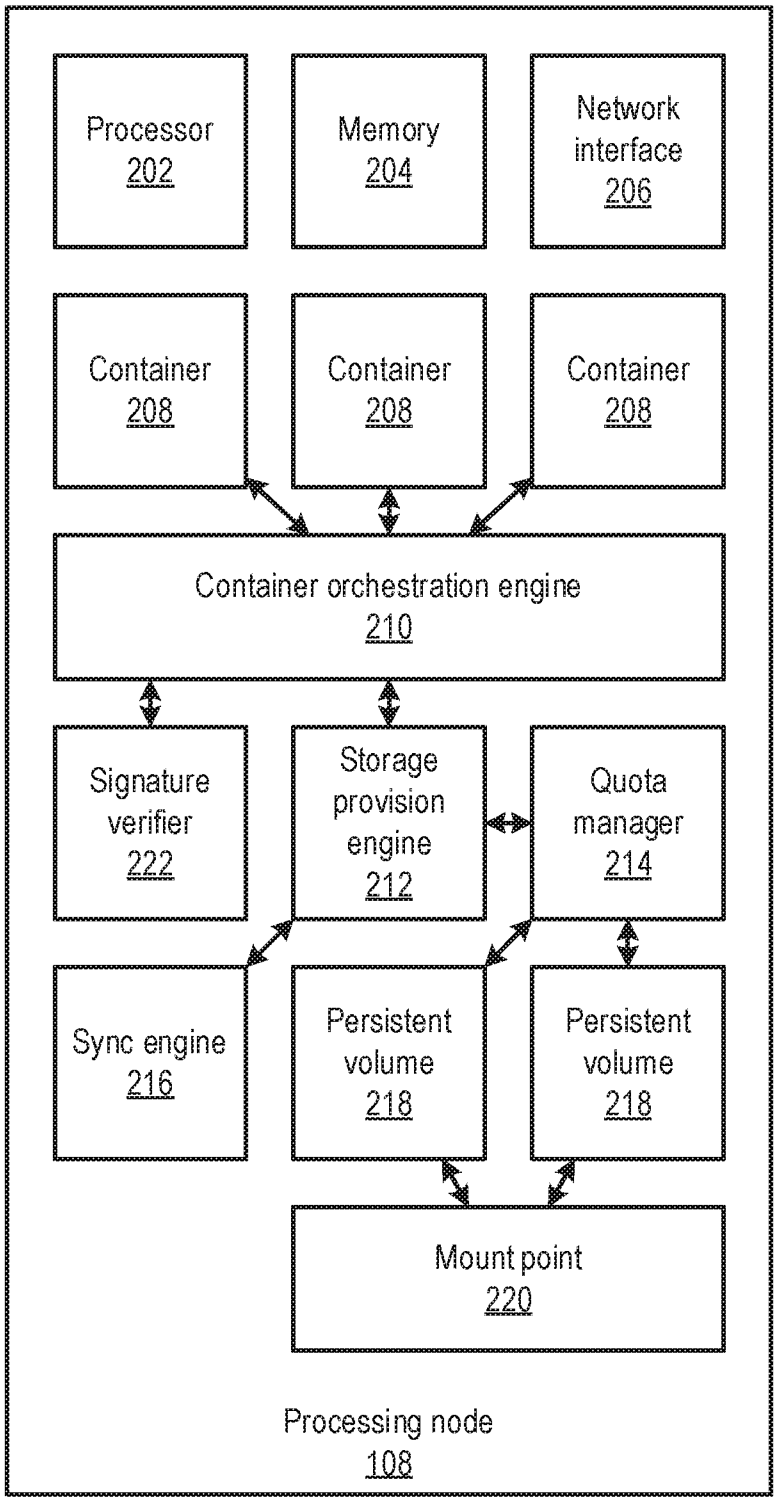

Processor
202
Memory
204
Network interface
206
Container
208
Container
208
Container
208
Container orchestration engine
210
Signature verifier
222
Storage provision engine
212
Quota manager
214
Sync engine
216
Persistent volume
218
Persistent volume
218
Mount point
220
Processing node
108

FIG. 2

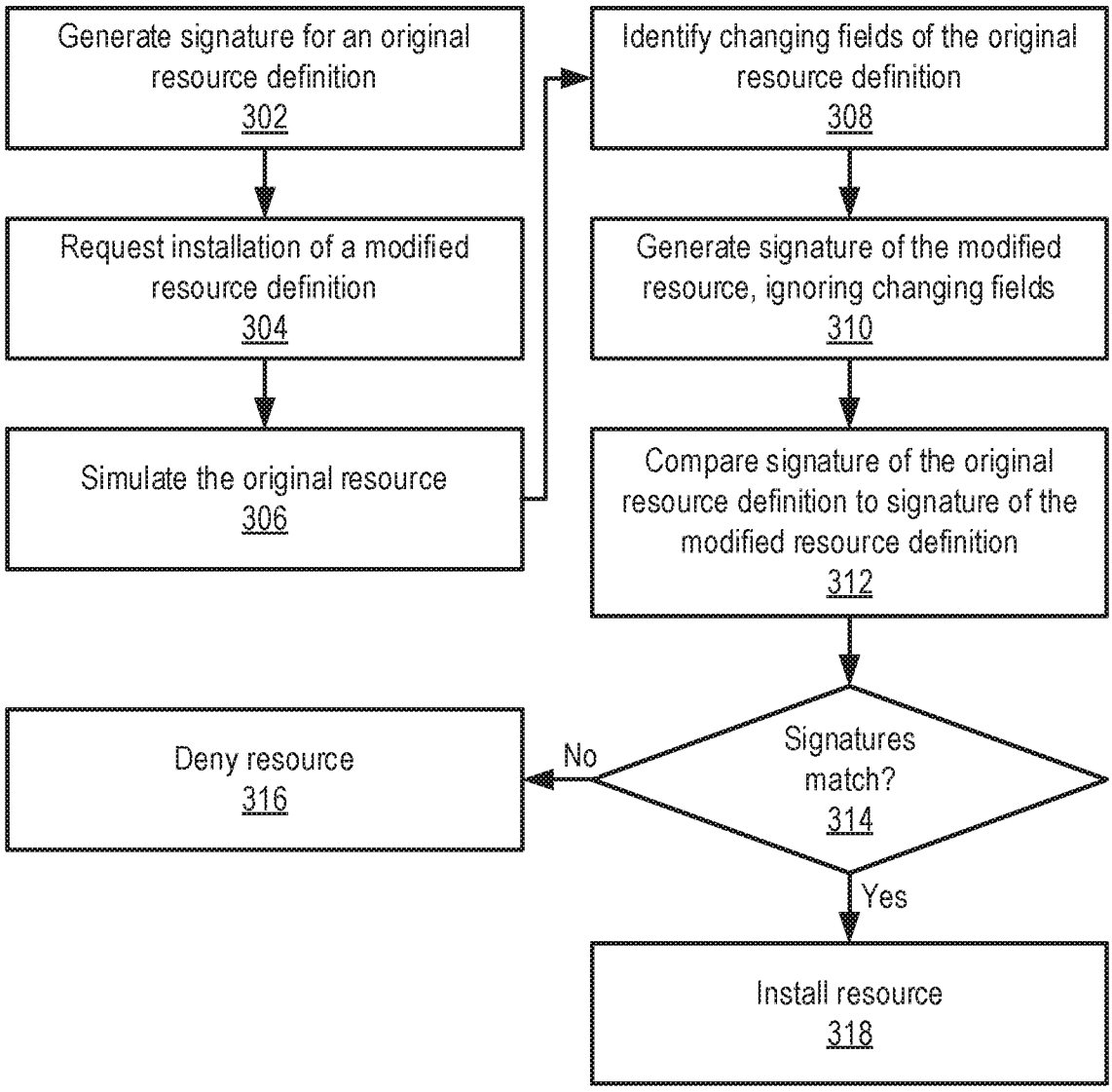

Generate signature for an original resource definition
302
Request installation of a modified resource definition
304
Simulate the original resource
306
Identify changing fields of the original resource definition
308
Generate signature of the modified resource, ignoring changing fields
310
Compare signature of the original resource definition to signature of the modified resource definition
312
Signatures match?
314
No
Deny resource
316
Yes
Install resource
318

FIG. 3

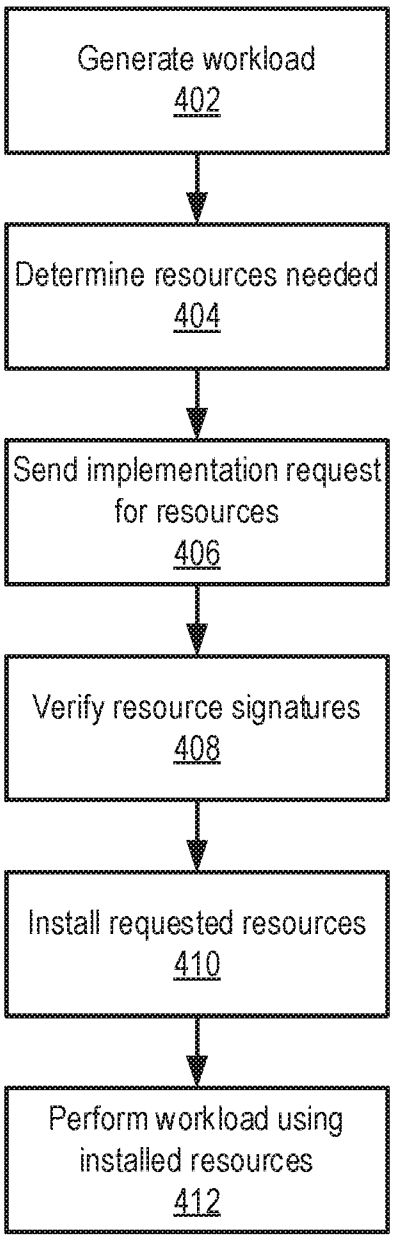

Generate workload
402
Determine resources needed
404
Send implementation request for resources
406
Verify resource signatures
408
Install requested resources
410
Perform workload using installed resources
412

FIG. 4

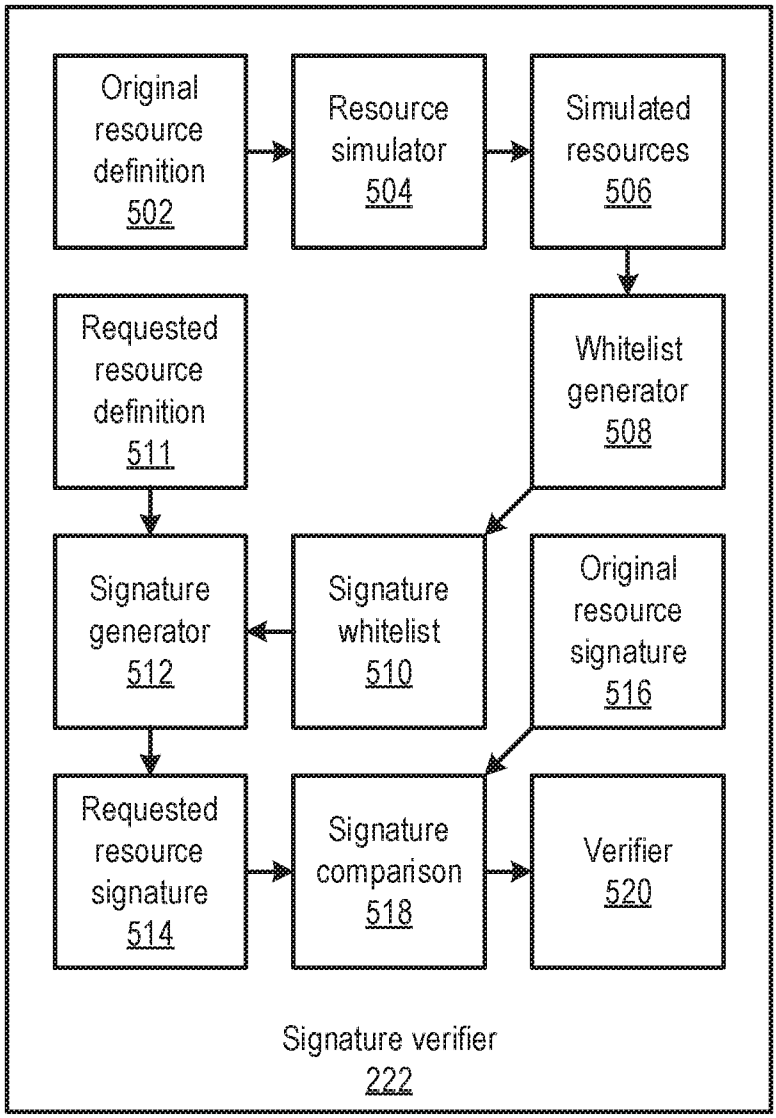

Original resource definition 502
Resource simulator 504
Simulated resources 506
Requested resource definition 511
Whitelist generator 508
Signature generator 512
Signature whitelist 510
Original resource signature 516
Requested resource signature 514
Signature comparison 518
Verifier 520
Signature verifier 222

FIG. 5

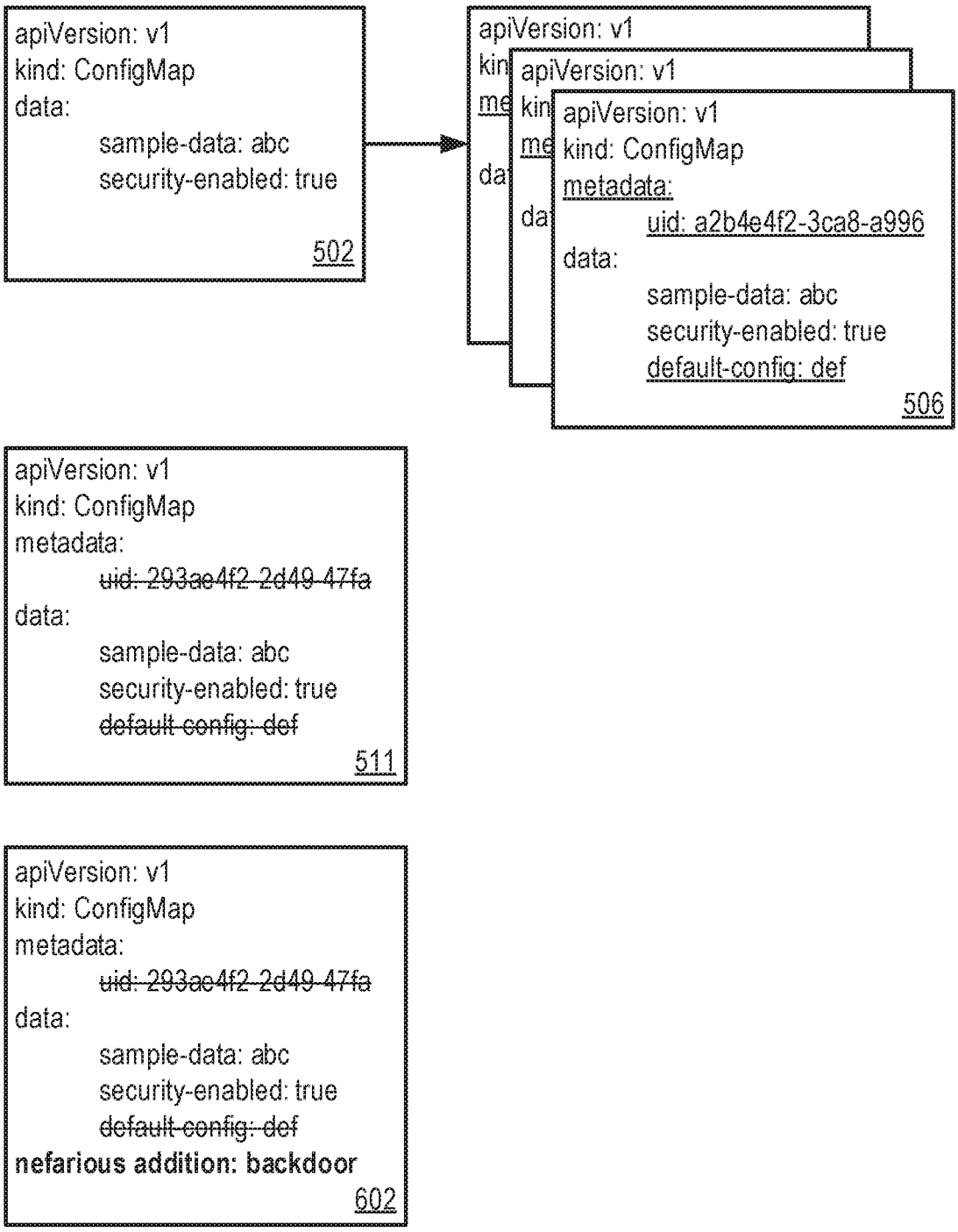

apiVersion: v1
kind: ConfigMap
data:
sample-data: abc
security-enabled: true
502
apiVersion: v1
apiVersion: v1
apiVersion: v1
kind: ConfigMap
metadata:
uid: a2b4e4f2-3ca8-a996
data:
sample-data: abc
security-enabled: true
default-config: def
506
apiVersion: v1
kind: ConfigMap
metadata:
uid: 293ae4f2-2d49-47fa
data:
sample-data: abc
security-enabled: true
default-config: def
511
apiVersion: v1
kind: ConfigMap
metadata:
uid: 293ae4f2-2d49-47fa
data:
sample-data: abc
security-enabled: true
default-config: def
nefarious addition: backdoor
602

FIG. 6

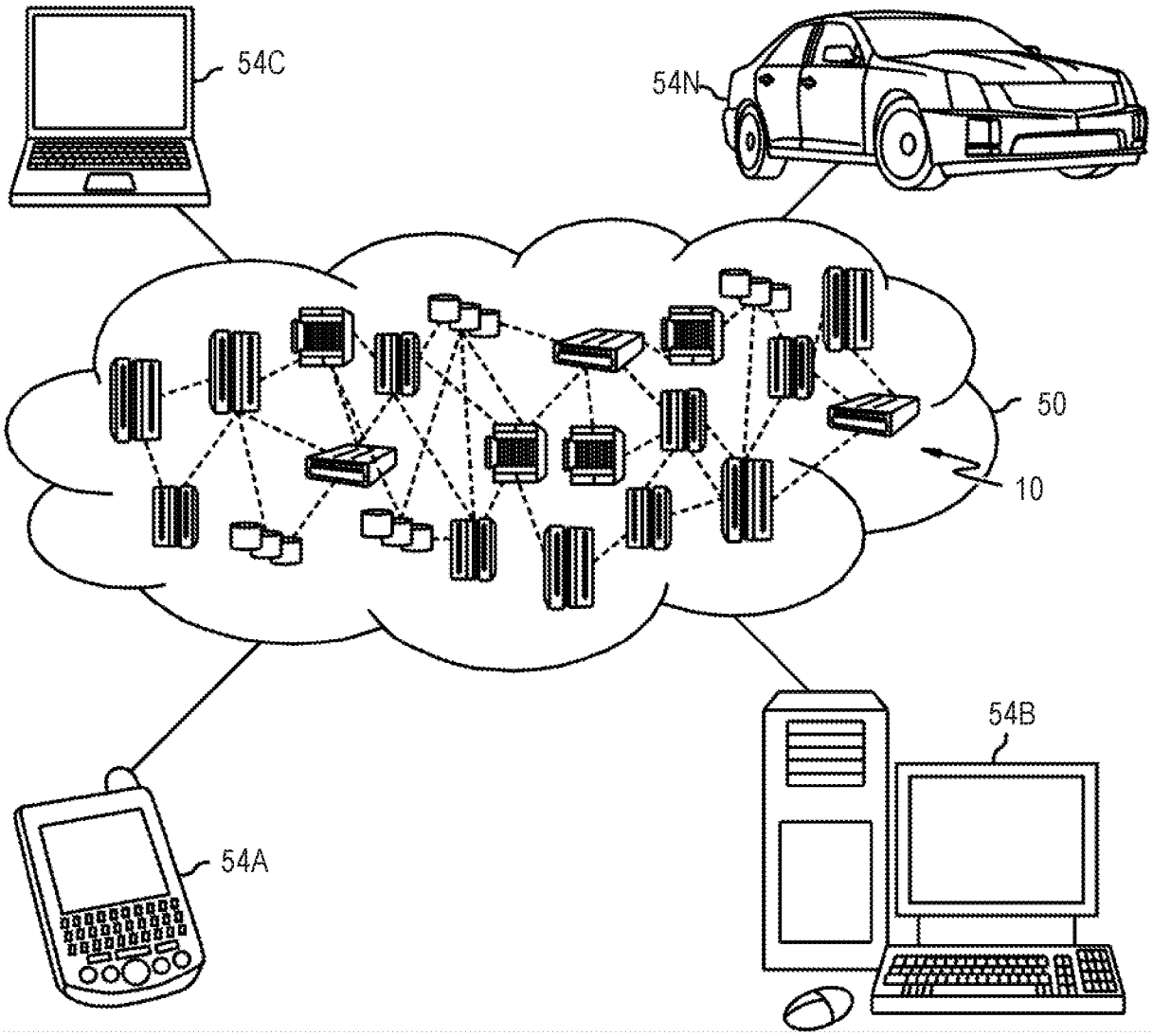


FIG. 7

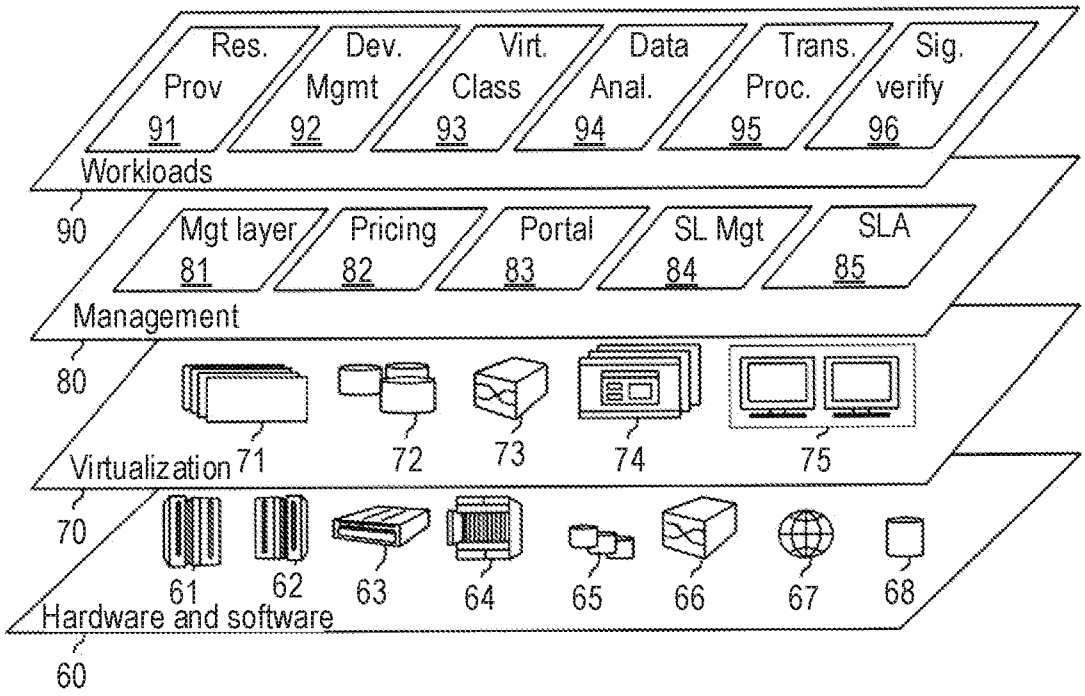

Res. Prov 91
Dev. Mgmt 92
Virt. Class 93
Data Anal. 94
Trans. Proc. 95
Sig. verify 96
Workloads
90
Mgt layer 81
Pricing 82
Portal 83
SL Mgt 84
SLA 85
Management
80
71
72
73
74
75
Virtualization
70
61
62
63
64
65
66
67
68
Hardware and software
60

FIG. 8

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CN2022/071600

A. CLASSIFICATION OF SUBJECT MATTER

H04L 9/32(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPODOC, WPI, CNPAT, CNKI, IEEE, GOOGLE: verify, resource, simulate, original, signature, compare, container manager, remove, node

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	CN 108777625 A (TENCENT TECHNOLOGY SHENZHEN CO., LTD. et al.) 09 November 2018 (2018-11-09) description, paragraphs [0027]-[0039], [0109]-[0121]	1-20
A	CN 106066812 A (INTERNATIONAL BUSINESS MACHINES CORPORATION) 02 November 2016 (2016-11-02) the whole document	1-20
A	CN 111144859 A (TENCENT TECHNOLOGY SHENZHEN CO., LTD.) 12 May 2020 (2020-05-12) the whole document	1-20
A	US 6263434 B1 (SUN MICROSYSTEMS, INC.) 17 July 2001 (2001-07-17) the whole document	1-20
A	US 9679122 B1 (RED HAT, INC.) 13 June 2017 (2017-06-13) the whole document	1-20

☐ Further documents are listed in the continuation of Box C. ☑ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
17 March 2022	**29 March 2022**
Name and mailing address of the ISA/CN **National Intellectual Property Administration, PRC** **6, Xitucheng Rd., Jimen Bridge, Haidian District, Beijing** **100088, China** Facsimile No. **(86-10)62019451**	Authorized officer **WU,Qing** Telephone No. **86-(10)-53961436**

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/CN2022/071600

Patent document cited in search report	Publication date (day/month/year)	Patent family member(s)	Publication date (day/month/year)
CN 108777625 A	09 November 2018	CN 110417558 A	05 November 2019
CN 106066812 A	02 November 2016	US 2016306655 A1	20 October 2016
CN 111144859 A	12 May 2020	None	
US 6263434 B1	17 July 2001	WO 0122655 A1	29 March 2001
		AU 7098000 A	24 April 2001
US 9679122 B1	13 June 2017	US 2017270287 A1	21 September 2017