



(51) International Patent Classification:

G06F 11/36 (2006.01) G06F 9/455 (2006.01)

(21) International Application Number:

PCT/JP2022/040552

(22) International Filing Date:

28 October 2022 (28.10.2022)

(25) Filing Language:

English

(26) Publication Language:

English

(71) Applicant: **RAKUTEN MOBILE, INC.** [JP/JP]; 1-14-1, Tamagawa, Setagaya-ku, Tokyo, 1580094 (JP).

(72) Inventor: **DAS Subhankar**; c/o Rakuten Mobile, Inc., 1-14-1, Tamagawa, Setagaya-ku, Tokyo, 1580094 (JP).

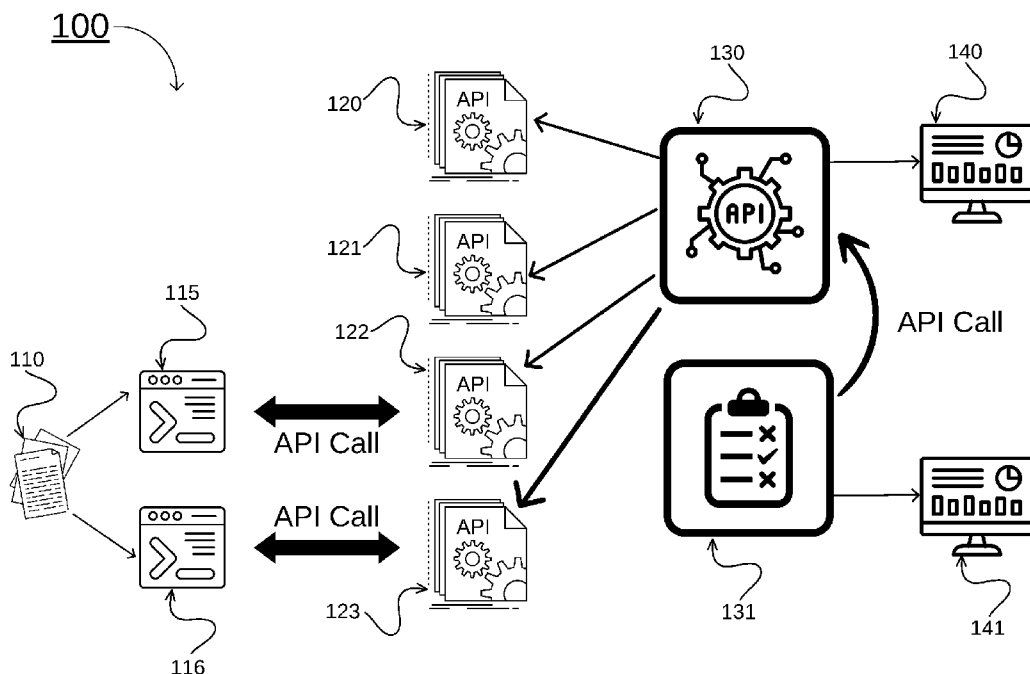
(74) Agent: **EINSEL Felix-Reinhard** et al.; Sonderhoff & Einsel Law and Patent Office, Shin-Marunouchi Center Building, 6-2, Marunouchi 1-chome, Chiyoda-ku, Tokyo, 1000005 (JP).

(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, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, 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, CV, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SC, 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, ME, MK, MT, NL, NO, PL, PT, RO, RS, SE,

(54) Title: SYSTEM, METHOD, AND MEDIUM FOR LIFECYCLE MANAGEMENT TESTING OF CONTAINERIZED APPLICATIONS

[Fig. 1]



(57) Abstract: Lifecycle Management Testing of a containerized application is conducted by executing a test script by a first containerized application thereby causing an Application Programming Interface (API) call to be issued to at least one automated testing system, running, by the at least one automated testing system, a testing sequence on a second containerized application, different from the first containerized application, based on the API call, and automatically displaying testing sequence results, where the results include an assessment of the health of the second containerized application.

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))*

Description

Title of Invention: SYSTEM, METHOD, AND MEDIUM FOR LIFECYCLE MANAGEMENT TESTING OF CONTAINERIZED APPLICATIONS

Background

[0001] Life Cycle Management (LCM) is performed to check the behavior and functionality, among other parameters, of containerized application pods. A containerized application is an application that runs in an isolated runtime environment called a container. The container encapsulates all the dependencies of its application, including binaries, system libraries, configuration files, and the like. Containers are portable, that is, they run consistently across multiple hosts. A pod is a collection of one or more containers encapsulating applications. LCM testing helps to determine how a pod will behave in a production environment, that is, how the pod will behave when made accessible to end-users. Testing a pod prior to deployment helps, for example, improve the health of a pod (e.g., reliability and performance) so that the pod behaves as predicted when moved to a production environment. LCM testing is used to check, for example, level of application availability, response to an infrastructure event, distribution of microservices across multiple servers, communication between microservices, resilience of applications after a network interruption, and to make sure applications continue to run after an instance or pod reboot, an application component crash, or central processing unit (CPU) starvation of an instance or a pod.

Brief Description of Drawings

- [0002] Aspects of the present disclosure are best understood from the following detailed description when read with the accompanying figures. It is noted that, in accordance with the standard practice in the industry, various features are not drawn to scale. In fact, the dimensions of the various features may be arbitrarily increased or reduced for clarity of discussion.
- [0003] [Fig.1]FIG. 1 is a diagram of an exemplary system suitable for LCM testing in accordance with some embodiments.
- [0004] [Fig.2]FIG. 2 is a flowchart of an exemplary method suitable for LCM testing in accordance with some embodiments.
- [0005] [Fig.3]FIG. 3 is a block diagram of an electronic device suitable for LCM testing in accordance with some embodiments.
- [0006] [Fig.4]FIG. 4 is a sample LCM test report generated by method 200 according to some embodiments.

DETAILED DESCRIPTION

- [0007] The following disclosure provides many different embodiments, or examples, for implementing different features of the provided subject matter. Specific examples of components, values, operations, materials, arrangements, or the like, are described below to simplify the present disclosure. These are, of course, merely examples and are not intended to be limiting. Other components, values, operations, materials, arrangements, or the like, are contemplated. For example, the connection of a first feature to a second feature in the description that follows may include embodiments in which the first and second features are directly connected, and may also include embodiments in which additional features may be connected and/or arranged between the first and second features, such that the first and second features may not be in direct connection or contact. In addition, the present disclosure may repeat reference numerals and/or letters in the various examples. This repetition is for the purpose of simplicity and clarity and does not in itself dictate a relationship between the various embodiments and/or configurations discussed.
- [0008] A pod is the smallest execution unit of a containerized application system. If a pod or the node a pod is running on fails, a new replica of the pod can be created and launched in order to keep that pod's one or more applications up and running. A node is a machine that runs the pod housing the containerized applications and can be a physical or virtual machine. Containers differ from virtual machines because containers do not include their own operating systems.
- [0009] When not running, a container exists as a container image, which is a packaged save file containing the application source code, binaries, and other files that exist within the container and allow it to function when the container is running. A container image can instantiate any number of container iterations needed to provide the desired functionality. The containers can be deployed to a cloud network in order to provide application services to network users.
- [0010] In order to run containerized applications at scale, for example, on networks that may have thousands or millions of users, an orchestration (i.e., container management) platform is generally used in order to manage the deploying and scaling of the containerized applications. Kubernetes is an example of one such containerized management / orchestration platform.
- [0011] Before a pod can move from the development stage to the production stage, LCM testing is conducted to test the behavior of the pod under different operating conditions. The lifecycle of a pod includes, but is not limited to: initialization of the pod, a pending state when the pod is ready to run and on standby, a create container state when the pod creates containers, a running state, where the pod is up and running, and an error state, which is entered upon the output of an error associated with the pod, for example, when one of the containers fails to run. The type of error output depends

on the particular malfunction of the pod. LCM testing of pods and their containerized applications allows for the gathering of data related to the performance of the pods and applications under production conditions, that is, when the pods and applications are deployed to be accessible to the intended users of the applications.

- [0012] In some approaches, testing is conducted through user interfaces such as a command line interface (CLI) and/or a graphical user interface (GUI). Depending on the scale of the deployment, using graphical user interfaces (GUIs) and command line interfaces (CLIs) may add significantly to LCM testing and deployment time. This contributes to application release delays. Further, attempts at avoiding application release delays by rushing application deployment contributes to the release of unreliable applications that may frustrate users.
- [0013] In contrast, an LCM testing regime that relies less on GUIs and CLIs can help increase the efficiency, speed, and reliability of LCM testing and contribute to shorter testing and deployment schedules, while also maintaining quality and reliability over GUI and CLI-dependent LCM testing.
- [0014] LCM testing helps determine the health of a pod prior to deployment. Pods have both a specification and a status. The specification indicates where the pod is in its lifecycle and the status indicates the specific status of a pod. LCM testing involves the analysis of certain parameters associated with a pod. Specifically, LCM testing of a pod can include verification of pod parameters such as liveness, readiness, affinity and anti-affinity, size of container images, highly available (HA) / node failover, replicas, multi-node, and Horizontal Pod Auto-scaling (HPA).
- [0015] A liveness probe indicates the health of a pod, that is whether the pod is running. If a pod is unhealthy (e.g., fails to run), the pod can be restarted. For example, if the liveness probe returns a "success" result, then the pod is determined to be healthy (i.e., running). If the liveness probe returns a "failure" result, the pod is unhealthy according to the liveness probe. A liveness probe may be configured with a timer which triggers periodic liveness checking of a pod. A readiness probe indicates whether a pod is ready to handle (e.g., respond to) a request. A "success" result indicates that the pod is ready to handle a request. A "failure" result indicates that a pod is not ready to handle a request. The readiness probe helps determine whether traffic to a pod should be sent or not, based on whether the pod is ready to receive traffic. Affinity and Antiaffinity defines the conditions or placement of the scheduling of a pod. Scheduling refers to the assignment of a pod to a node. Pods can be grouped together based on affinity. For example if two or more pods have affinity for one another, they can be scheduled on the same node. However, if two or more pods should not be scheduled on the same node, this is referred to as antiaffinity. Determining affinity and antiaffinity of pods allows for pods to be effectively spread across multiple

nodes, which provides enhanced availability in the case of node failure. Container image size refers to the data size of a container image. Smaller images can be downloaded more quickly than larger images. In general, images should be smaller than 25 megabytes (MB), however, image sizes up to 50 MB are acceptable. If an image is greater than 100 MB, the image will be flagged. Image size is not a required parameter in LCM testing and does not decide whether the LCM test returns a pass or fail result. High availability (HA) / node failover testing is associated with redundancy, which can provide both high availability and load balancing features. For example, multiple pods having the same roles that are spread across multiple nodes provide high availability in the case of node failure because traffic can be shifted from one pod to another pod on a different node without significant interruption in service. The Horizontal Pod Autoscaler (HPA) helps automatically scale the number of pods based on metrics such as CPU utilization or other appropriate metric. For example, the number of pods on a node may be reduced if CPU and or memory utilization is at or above a predetermined threshold. Minimum thresholds may also be set for memory and CPU utilization. The HPA is not a required parameter in LCM testing and does not decide whether the LCM test returns a pass or fail result.

[0016] In some embodiments of the present disclosure, a test script is executed by a first containerized application, thereby causing an Application Programming Interface (API) call to be issued to at least one automated testing system, at least one automated testing system runs a testing sequence on a second containerized application which is different from the first containerized application, based on the issued API call, and testing sequence results associated with the second containerized application are automatically displayed which include an assessment of the health of the second containerized application. In some embodiments, the testing sequence includes testing pod parameters such as liveness, readiness, affinity and anti-affinity, size of container images, highly available (HA) / node failover, replicas, multi-node, and Horizontal Pod Auto-scaling (HPA). In some embodiments, other components, such as binaries or other appropriate application components are verified. In some embodiments, a representational state transfer (REST) API call is used as the API call. In some embodiments, the at least one automated testing system is an automated test suite such as Jenkins(TM).

[0017] FIG. 1 is a diagram of exemplary system 100 suitable for LCM testing in accordance with some embodiments. A user executes one or more test scripts 110 which will interact with a target cluster to carry out LCM testing of one or more pods. Execution of the one or more test scripts 110 causes a first automated LCM testing application command to be executed via command line 116. In some embodiments, the LCM testing application command is executed via an API call between REST API Agent

130 and the LCM testing pod 131. In some embodiments, the command is a login command for logging in to the LCM testing application. The LCM testing application can be an application that automates the deployment, scaling and lifecycle management of containerized applications such as Robin.io(TM). Execution of the one or more test scripts 110 causes a REST API call (shown as "API Call" arrows between the command line interfaces 115 and 116 and API gateways 120-123) through an API gateway (any of API gateways 120-123) to execute a containerized software (e.g., kubernetes) command via a command line 115 command (e.g., kubectl) to check the status of a target pod and/or container. Execution of the one or more test scripts 110 causes a REST API call (shown as "API Call" arrows between the command line interfaces 115 and 116 and API gateways 120-123) to be issued between REST API Agent pod 130 and an artifactory application (e.g., JFrog(TM)) to check the size of a docker image stored by the artifactory application. Execution of the one or more test scripts 110 calls an automation pipeline (e.g., LCM TESTING PIPELINE 210, shown in FIG. 2) of an automated testing application via any of the API gateways 120-123 from the REST API Agent pod 130 via an API call (shown as "API Call" arrow between the API Agent pod 130 and LCM Testing pod 131) to execute a separate automation test script which causes the automated testing application to validate application features of the pod under test while the LCM testing is in progress. The LCM testing pod 131 is deployed on a cluster accessible to any of the clusters where an application to be LCM tested is deployed. Further, the LCM testing pod 131 is configured to communicate with the API Agent pod via REST API calls. Both the LCM testing pod and the API Agent pod report test progress and provide testing sequence results through a report display 140 and/or 141 (e.g., a graphical user interface (GUI)) as well as through the generation of testing sequence report documents (e.g., spreadsheets).

[0018] FIG. 2 is a flowchart of exemplary method 200 suitable for LCM testing in accordance with some embodiments. Each operation of method 200 is triggered with an API call (e.g., REST API call or other application-specific API call such as a kubectl API call). However, the sending of the test report (operation 218) can be triggered with a script. Method 200 is designed to get application information from a pod under test and analyze the information, fetch all objects in a namespace and analyze each fetched object, and to test on a Helm(TM) application release (e.g., decode helm release information that is stored as kubernetes secrets in a particular version of Helm and analyze the objects in the decoded secret). In operation 210 an LCM testing pipeline is initiated for causing an automated testing application to carry out specific testing tasks. In some embodiments, the LCM testing pipeline is generated via a testing script. In some embodiments, the LCM testing pipeline is downloaded from a stored library. The

LCM testing pipeline includes steps of: cleaning the workspace of the testing application, parsing application data to be used in the creation of an LCM configuration file, cloning LCM automation code (e.g., checking out an LCM testing automation script to be used for the automated testing), removing old configuration files (e.g., delete old LCM configuration files), generate new LCM testing configuration file to be used in the testing process, and run the LCM test script. Running the LCM test script executes the LCM automation script, which invokes test cases and provides testing results for a pod under test. Operation 210 also initiates an API call that causes the execution of operation 211. In operation 211, application bundles are verified. This includes verification of readiness probe, liveness probe, affinity and anti-affinity rules, replicas, CPU and memory requirements for running the pod under test, storage configurations, and auto healing bundles. In this application, auto healing refers to a pod's ability to recover from errors or a failure in operation. For example, if a pod begins to malfunction, it may auto heal by restarting to recover from the malfunction. In operation 212 application components are verified, including verification of the bundles mentioned in operation 211 in at least one application namespace. In operation 213 healing and restart time of the pod under test is validated by deleting application replicas, partially deleting pod components to trigger auto healing and checking the time it takes for auto healing, and also measuring pod restart times. In operation 214 node failover operations of a pod under test is checked by replacing and/or restarting nodes and verifying the respawning of the pod or pods under test. Optionally, application functionality is checked during the replacing and/or respawning of a pod or pods under test and/or performance scenarios to check auto scaling of a pod or pods under test can be conducted in operation 214. That is, how well an application hosted by a pod under test functions during pod replacement and/or respawn and spawning of new pods or deletion of unnecessary pods is carried out (i.e., auto scaling). In operation 220 the LCM testing of the pod under test is executed and a "pass" or "fail" result is determined. A "pass" result, for example, is when the pod under test operates normally. A "fail" result, for example, is when the pod fails to operate or partially fails to operate. The pass or fail results are recorded as a status update of the operation of the pod under test in operation 230. In operation 215 autoscaling of the pod under test is checked. That is, how the pod replicates or deletes copies of itself in response to increased or decreased demand for the pods application services. In operation 216 performance of the pod under test is checked in the event of deletion and relocation of the pod (e.g., relocation to a different node). In operation 221 the automated testing suite executes testing of the pod again. In some embodiments, operation 221 executes testing using at least a portion of the output from the test conducted in operation 220. In operation 221, the LCM testing of the pod under test is executed and a "pass" or "fail" result is de-

terminated. A "pass" result, for example, is when the pod under test operates normally. A "fail" result, for example, is when the pod fails to operate or partially fails to operate. The pass or fail results are recorded as a status update of the operation of the pod under test in operation 231. In operation 217 test data is consolidated and a test report is generated based on the previous test operations. In operation 218 the test report generated in operation 217 is sent to a predetermined recipient (e.g., email recipient). Operation 218 is optional.

[0019] FIG. 3 is a block diagram of a system 300 for implementing a method for connecting to a network, using an Internet-enabled radio node, in accordance with some embodiments. System 300 includes a hardware processor 310 and a non-transitory, computer readable storage medium 350 encoded with, (i.e., storing), the computer program code 370, (i.e., a set of executable instructions). Computer readable storage medium 350 is also encoded with instructions 351 for interfacing with different devices within a network. The processor 310 is electrically coupled to the computer readable storage medium 350 via a bus 360. The processor 310 is also electrically coupled to an I/O interface 320 by bus 360. A network interface 330 is also electrically connected to the processor 310 via bus 360. Network interface 330 is connected to a network 340, so that processor 310 and computer readable storage medium 350 are capable of connecting to external elements via network 340. The processor 310 is configured to execute the computer program code 370 encoded in the computer readable storage medium 350 in order to cause system 300 to be usable for performing a portion or all of the operations as described in method 200.

[0020] In some embodiments, the processor 310 is a central processing unit (CPU), a multi-processor, a distributed processing system, an application specific integrated circuit (ASIC), and/or a suitable processing unit.

[0021] In some embodiments, the computer readable storage medium 350 is an electronic, magnetic, optical, electromagnetic, infrared, and/or a semiconductor system (or apparatus or device). For example, the computer readable storage medium 350 includes a semiconductor or solid-state memory, a magnetic tape, a removable computer diskette, a random-access memory (RAM), a read-only memory (ROM), a rigid magnetic disk, and/or an optical disk. In some embodiments using optical disks, the computer readable storage medium 350 includes a compact disk-read only memory (CD-ROM), a compact disk-read/write (CD-R/W), and/or a digital video disc (DVD).

[0022] In some embodiments, the storage medium 350 stores the computer program code 370 configured to cause system 300 to perform method 200. In some embodiments, the storage medium 350 also stores information needed for performing a method 200 as well as information generated and/or used during performing the method 200, such as test script data 352, first containerized application 353, and second containerized ap-

plication 354 and/or a set of executable instructions to perform the operation of method 200.

[0023] In some embodiments, the storage medium 350 stores instructions 351 for interfacing with external components within the network. The instructions 351 enable processor 310 to generate instructions readable by the external components to effectively implement method 200.

[0024] System 300 includes I/O interface 320. I/O interface 320 is coupled to external circuitry. In some embodiments, I/O interface 320 includes a keyboard, keypad, mouse, trackball, trackpad, and/or cursor direction keys for communicating information and commands to processor 310.

[0025] System 300 also includes network interface 330 coupled to the processor 310. Network interface 330 allows system 300 to communicate with network 340, to which one or more other computer systems are connected. Network interface 330 includes wireless network interfaces such as BLUETOOTH, WIFI, WIMAX, GPRS, or WCDMA; or wired network interface such as ETHERNET, USB, or IEEE-1394. In some embodiments, method 200 is implemented in two or more systems 300, and information such as memory type, memory array layout, I/O voltage, and I/O pin location are exchanged between different systems 300 via network 340.

[0026] FIG. 4 is a sample LCM test report generated by method 200. The report includes the name of the application in a pod under test (APP Name), which is "Process Hub" in this example. The "Components" column lists the components tested, and other parameters such as readiness, liveness, and the like which are each explained herein, are also shown. The report may be displayed and/or sent to a recipient (e.g., email recipient).

[0027] An aspect of this disclosure relates to a system for lifecycle management testing of containerized applications, the system that includes a memory that stores instructions, and at least one processor configured by the instructions to perform operations that include executing a test script by a first containerized application thereby causing an Application Programming Interface (API) call to be issued to at least one automated testing system, running, by the at least one automated testing system, a testing sequence on a second containerized application, different from the first containerized application, based on the API call, and automatically displaying testing sequence results, wherein the results comprise an assessment of the health of the second containerized application. In some embodiments, the first containerized application runs in a separate container from the first containerized application. In some embodiments, the operations further include running, by a second automated testing system, a second test sequence on the second containerized application, using the testing sequence results as an input parameter. In some embodiments, the operations further include running, by a

third automated testing system, a third test sequence on the second containerized application, using the testing sequence results as an input parameter. In some embodiments, the first and second containerized applications are in different pods on a same cluster of a network. In some embodiments, the operations further include executing a second testing script, and the execution of the testing script causes the at least one automated testing system to generate a testing pipeline to be used as the testing sequence. In some embodiments, the assessment of the health of the second containerized application includes at least one of: checking whether the application is up and running, checking whether the application is ready to accept traffic, checking node scheduling of the application, checking how many copies of the application should be scheduled, checking a container image size of the application, or checking node failover information associated with the application.

[0028] An aspect of this disclosure relates to a computer-implemented method for lifecycle management testing of containerized applications. The method includes executing a test script by a first containerized application thereby causing an Application Programming Interface (API) call to be issued to at least one automated testing system, running, by the at least one automated testing system, a testing sequence on a second containerized application, different from the first containerized application, based on the API call, and automatically displaying testing sequence results, wherein the results comprise an assessment of the health of the second containerized application. In some embodiments, the first containerized application exists in a separate container from the first containerized application. In some embodiments, the method further includes running, by a second automated testing system, a second test sequence on the second containerized application, using the testing sequence results as an input parameter. In some embodiments, the method further includes running, by a third automated testing system, a third test sequence on the second containerized application, using the testing sequence results as an input parameter. In some embodiments, the first and second containerized applications are in different pods on a same cluster of a network. In some embodiments, the method further includes executing a second testing script, wherein the execution of the testing script causes the at least one automated testing system to generate a testing pipeline to be used as the testing sequence. In some embodiments, the assessment of the health of the second containerized application includes at least one of: checking whether the application is up and running, checking whether the application is ready to accept traffic, checking node scheduling of the application, checking how many copies of the application should be scheduled, checking a container image size of the application, or checking node failover information associated with the application.

[0029] An aspect of this disclosure relates to a non-transitory computer readable medium for

lifecycle management testing of containerized applications, storing instructions that, when executed by at least one processor, cause the at least one processor to perform operations including executing a test script by a first containerized application thereby causing an Application Programming Interface (API) call to be issued to at least one automated testing system, running, by the at least one automated testing system, a testing sequence on a second containerized application, different from the first containerized application, based on the API call, and automatically displaying testing sequence results, wherein the results include an assessment of the health of the second containerized application. In some embodiments, the first containerized application exists in a separate container from the first containerized application. In some embodiments, the method further includes running, by a second automated testing system, a second test sequence on the second containerized application, using data output by the first test sequence. In some embodiments, the first and second containerized applications are in different pods on a same cluster of a network. In some embodiments, the method further comprises executing a second testing script, and the execution of the testing script causes the at least one automated testing system to generate a testing pipeline to be used as the testing sequence. In some embodiments, the assessment of the health of the second containerized application includes at least one of: checking whether the application is up and running, checking whether the application is ready to accept traffic, checking node scheduling of the application, checking how many copies of the application should be scheduled, checking a container image size of the application, or checking node failover information associated with the application.

[0030] The foregoing outlines features of several embodiments so that those skilled in the art may better understand the aspects of the present disclosure. Those skilled in the art should appreciate that they may readily use the present disclosure as a basis for designing or modifying other processes and structures for carrying out the same purposes and/or achieving the same advantages of the embodiments introduced herein. Those skilled in the art should also realize that such equivalent constructions do not depart from the spirit and scope of the present disclosure, and that they may make various changes, substitutions, and alterations herein without departing from the spirit and scope of the present disclosure.

Claims

- [Claim 1] A system for lifecycle management testing of containerized applications, the system comprising:
a memory that stores instructions; and
at least one processor configured by the instructions to perform operations comprising:
executing a test script by a first containerized application thereby causing an Application Programming Interface (API) call to be issued to at least one automated testing system;
running, by the at least one automated testing system, a testing sequence on a second containerized application, different from the first containerized application, based on the API call; and
automatically displaying testing sequence results, wherein the results comprise an assessment of the health of the second containerized application.
- [Claim 2] The system of claim 1, wherein the first containerized application runs in a separate container from the first containerized application.
- [Claim 3] The system of claim 1, wherein the operations further comprise:
running, by a second automated testing system, a second test sequence on the second containerized application, using the testing sequence results as an input parameter.
- [Claim 4] The system of claim 3, wherein the operations further comprise:
running, by a third automated testing system, a third test sequence on the second containerized application, using the testing sequence results as an input parameter.
- [Claim 5] The system of claim 1, wherein the first and second containerized applications are in different pods on a same cluster of a network.
- [Claim 6] The system of claim 1, wherein the operations further comprise:
executing a second testing script, wherein
the execution of the testing script causes the at least one automated testing system to generate a testing pipeline to be used as the testing sequence.
- [Claim 7] The system of claim 1, wherein
the assessment of the health of the second containerized application comprises at least one of: checking whether the application is up and running, checking whether the application is ready to accept traffic, checking node scheduling of the application, checking how many

copies of the application should be scheduled, checking a container image size of the application, or checking node failover information associated with the application.

[Claim 8] A computer-implemented method for lifecycle management testing of containerized applications, the method comprising:
executing a test script by a first containerized application thereby causing an Application Programming Interface (API) call to be issued to at least one automated testing system;
running, by the at least one automated testing system, a testing sequence on a second containerized application, different from the first containerized application, based on the API call; and
automatically displaying testing sequence results, wherein the results comprise an assessment of the health of the second containerized application.

[Claim 9] The method of claim 8, wherein
the first containerized application exists in a separate container from the first containerized application.

[Claim 10] The method of claim 8, further comprising:
running, by a second automated testing system, a second test sequence on the second containerized application, using the testing sequence results as an input parameter.

[Claim 11] The method of claim 10, further comprising:
running, by a third automated testing system, a third test sequence on the second containerized application, using the testing sequence results as an input parameter.

[Claim 12] The method of claim 8, wherein the first and second containerized applications are in different pods on a same cluster of a network.

[Claim 13] The method of claim 8, further comprising:
executing a second testing script, wherein the execution of the testing script causes the at least one automated testing system to generate a testing pipeline to be used as the testing sequence.

[Claim 14] The method of claim 8, wherein
the assessment of the health of the second containerized application comprises at least one of: checking whether the application is up and running, checking whether the application is ready to accept traffic, checking node scheduling of the application, checking how many copies of the application should be scheduled, checking a container image size of the application, or checking node failover information associated with the application.

sociated with the application.

[Claim 15]

A non-transitory computer-readable medium for lifecycle management testing of containerized applications, storing instructions that, when executed by at least one processor, cause the at least one processor to perform operations comprising:
executing a test script by a first containerized application thereby causing an Application Programming Interface (API) call to be issued to at least one automated testing system;
running, by the at least one automated testing system, a testing sequence on a second containerized application, different from the first containerized application, based on the API call; and
automatically displaying testing sequence results, wherein the results comprise an assessment of the health of the second containerized application.

[Claim 16]

The medium of claim 15, wherein
the first containerized application exists in a separate container from the first containerized application.

[Claim 17]

The medium of claim 15, further comprising
running, by a second automated testing system, a second test sequence on the second containerized application, using data output by the first test sequence.

[Claim 18]

The medium of claim 16, wherein the first and second containerized applications are in different pods on a same cluster of a network.

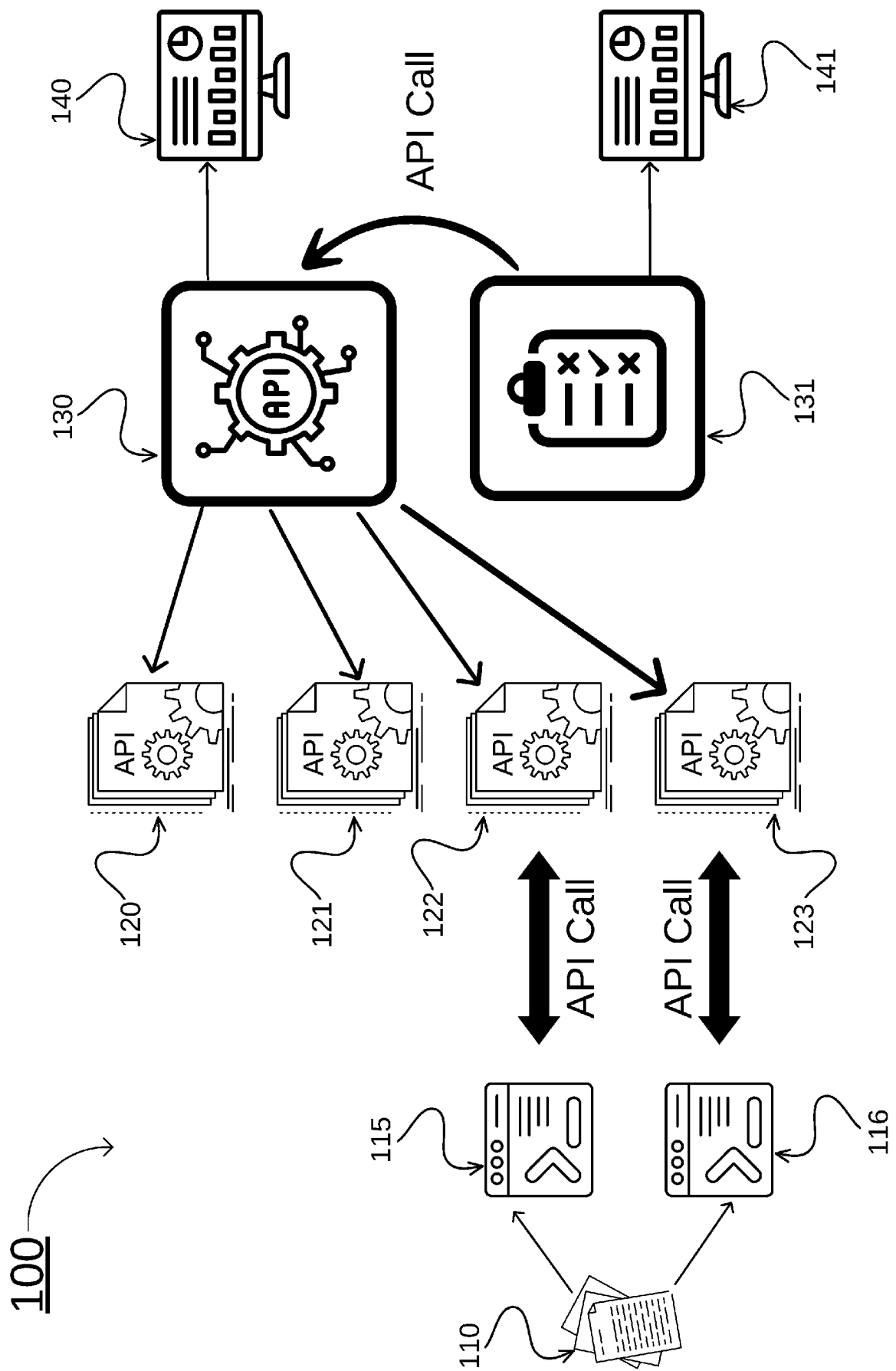
[Claim 19]

The medium of claim 15, further comprising:
executing a second testing script, wherein
the execution of the testing script causes the at least one automated testing system to generate a testing pipeline to be used as the testing sequence.

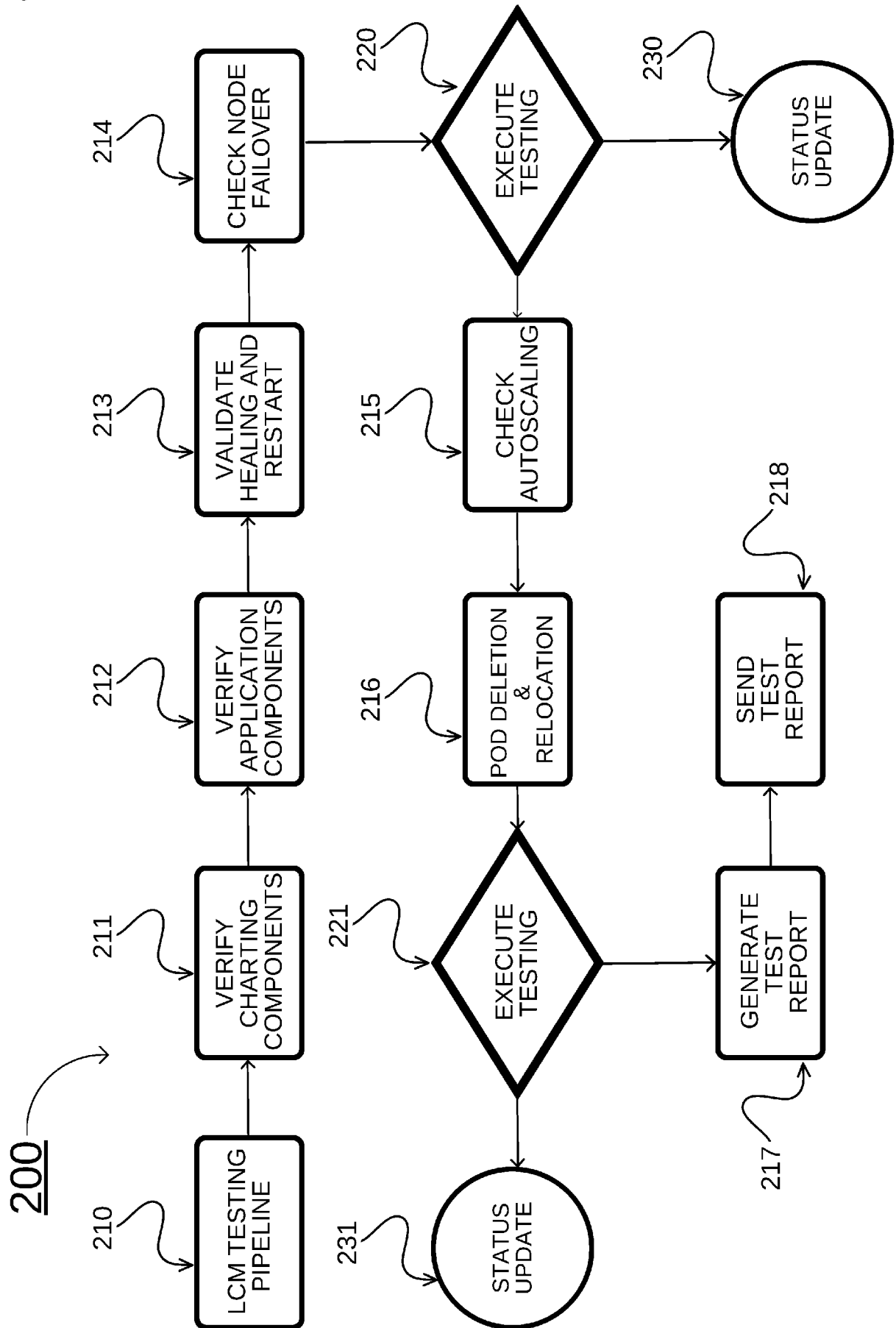
[Claim 20]

The medium of claim 15, wherein
the assessment of the health of the second containerized application comprises at least one of: checking whether the application is up and running, checking whether the application is ready to accept traffic, checking node scheduling of the application, checking how many copies of the application should be scheduled, checking a container image size of the application, or checking node failover information associated with the application.

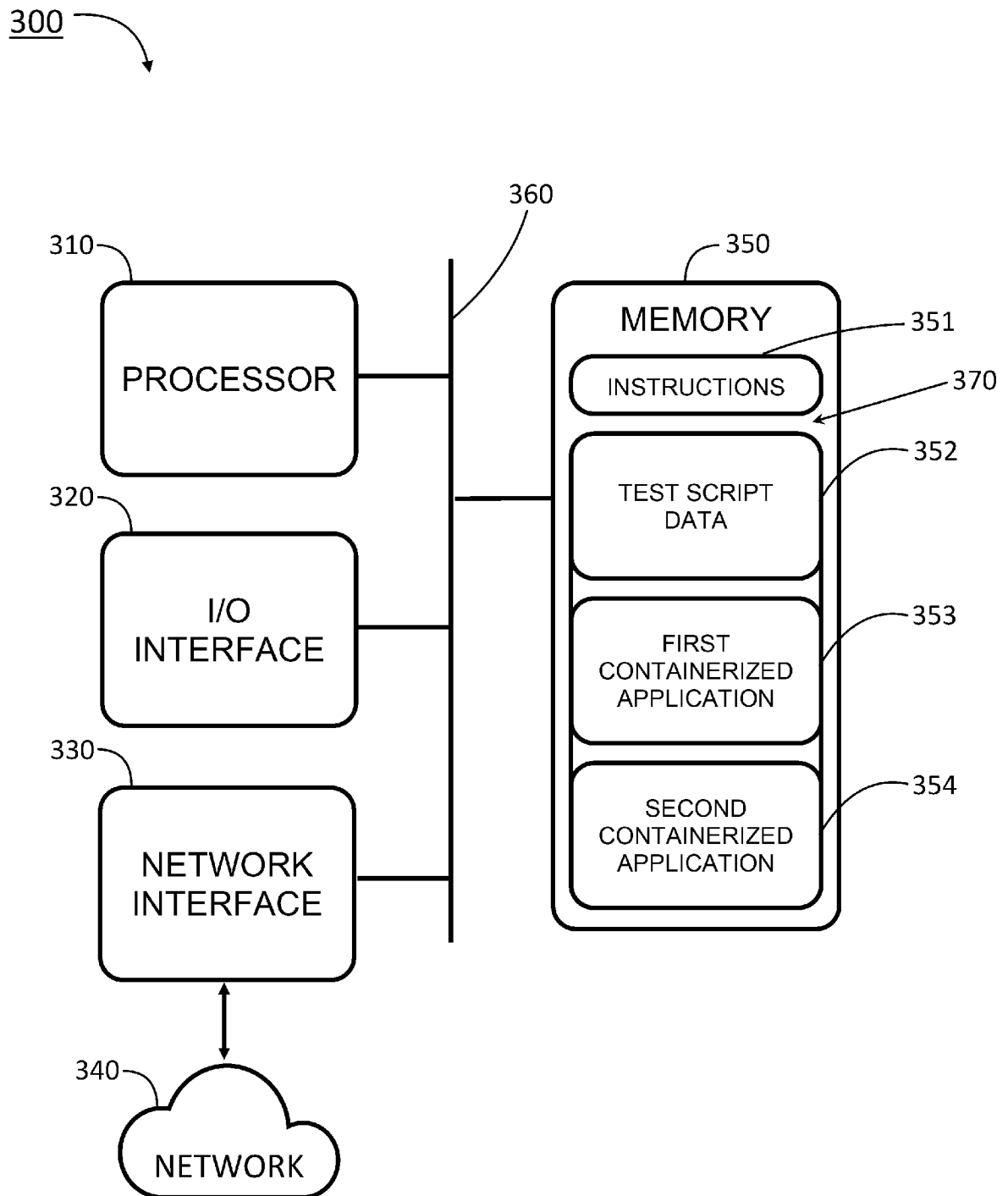
[Fig. 1]



[Fig. 2]



[Fig. 3]



[Fig. 4]

Sample LCM Test Report

APP Name	Components	Readiness probe (health url / process)	Liveness probe (health url / process)	Resource req & limit	Replicas	HA	Affinity & Anti-affinity Rules	Image Size	Auto Healing	Node failover	App snapshot setup	Restart time	Comments
Process Hub	rcpphubui-rcpphubui	Yes	Yes	RAM-1G,CPU-1 Core	NA	Yes	Yes	78.1 MB	Yes	NA	NA	4m 55s	
	phub-db-mysql-node	No	No	RAM-8G,CPU-4 Cores	NA	Yes	Yes	131.7 MB	Yes	NA	NA	4m 35s	Missing liveness & readiness probes
	phub-db-mysql-router	Yes	Yes	RAM-1G,CPU-2 Core	NA	Yes	Yes	126.5 MB	Yes	NA	NA	2m 45s	
	process-hub	Yes	Yes	RAM-8G,CPU-6 Cores	NA	Yes	Yes	25.8 MB	Yes	NA	NA	3m 35s	

INTERNATIONAL SEARCH REPORT

International application No.

PCT/JP2022/040552

A. CLASSIFICATION OF SUBJECT MATTER**G06F 11/36**(2006.01)i; **G06F 9/455**(2006.01)i

FI: G06F11/36 168; G06F9/455 150

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)

G06F11/36; G06F9/455

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

Published examined utility model applications of Japan 1922-1996
 Published unexamined utility model applications of Japan 1971-2023
 Registered utility model specifications of Japan 1996-2023
 Published registered utility model applications of Japan 1994-2023

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

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	EP 1876532 A1 (TELEFONAKTIEBOLAGET LM ERICSSON (PUBL.)) 09 January 2008 (2008-01-09) Paras. [0007], [0015]-[0029], Figs. 1-2	1-20
Y	US 2004/0078719 A1 (SUN MICROSYSTEMS, INC.) 22 April 2004 (2004-04-22) Paras. [0009], [0026]-[0027], Fig. 1	1-20
Y	US 2021/0157716 A1 (INTERNATIONAL BUSINESS MACHINES CORPORATION) 27 May 2021 (2021-05-27) Para. [0004]	6,13,19
A	US 2018/0336123 A1 (RED HAT, INC.) 22 November 2018 (2018-11-22) entire 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

11 January 2023

Date of mailing of the international search report

24 January 2023

Name and mailing address of the ISA/JP

Japan Patent Office
 3-4-3, Kasumigaseki, Chiyoda-ku, Tokyo
 100-8915, Japan

Authorized officer

KANEDA, Takayuki 5B 3144

Telephone No. +81-3-3581-1101 Ext. 3545

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.

PCT/JP2022/040552

Patent document cited in search report			Publication date (day/month/year)	Patent family member(s)			Publication date (day/month/year)
EP	1876532	A1	09 January 2008	US	2009/0204851	A1	
				WO	2008/003764	A2	
				CN	101484881	A	
US	2004/0078719	A1	22 April 2004	(Family: none)			
US	2021/0157716	A1	27 May 2021	(Family: none)			
US	2018/0336123	A1	22 November 2018	(Family: none)			