UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
MIDLAND-ODESSA DIVISION

VIRTAMOVE, CORP., Plaintiff, v. ORACLE CORPORATION, Defendant.	Case No. 7:24-cv-00339-ADA **JURY TRIAL DEMANDED**

FIRST AMENDED COMPLAINT FOR PATENT INFRINGEMENT AGAINST ORACLE CORPORATION

This is an action for patent infringement arising under the Patent Laws of the United States of America, 35 U.S.C. § 1 *et seq.*, in which Plaintiff VirtaMove Corp. (collectively, "Plaintiff" or "VirtaMove") makes the following allegations against Defendant Oracle Corporation ("Defendant" or "Oracle"):

INTRODUCTION AND PARTIES

1. This complaint arises from Defendant's unlawful infringement of the following United States patents owned by VirtaMove, each of which generally relate to novel containerization systems and methods: United States Patent Nos. 7,519,814 and 7,784,058 (collectively, the "Asserted Patents"). VirtaMove owns all right, title, and interest in each of the Asserted Patents to file this case.

2. VirtaMove, Corp. is a is a corporation organized and existing under the laws of Canada, having its place of business at 110 Didsbury Road, M083, Ottawa, Ontario K2T 0C2. VirtaMove is formerly known as AppZero Software Corp. ("AppZero"), which was established in 2010.

Oracle Ex. 1113, p. 1
Oracle v. VirtaMove
IPR2025-00982

3. VirtaMove is an innovator and pioneer in containerization. At a high level, a container is a portable computing environment. It can hold everything an application needs to run to move it from development to testing to production smoothly. Containerization lowers software and operational costs, using far fewer resources. It provides greater scalability (for example, compared to virtual machines). It provides a lightweight and fast infrastructure to run updates and make changes. It also encapsulates the entire code with its dependencies, libraries, and configuration files, effectively removing errors that can result from traditional configurations.

4. For years, VirtaMove has helped customers repackage, migrate and refactor thousands of important, custom, and packaged Windows Server, Unix Sun Solaris, & Linux applications to modern, secure operating systems, without recoding. VirtaMove's mission is to move and modernize the world's server applications to make organizations more successful and secure. VirtaMove has helped companies from many industries achieve modernization success.

5. The use of containerization has been growing rapidly. For instance, one source predicted the application containers market to reach $2.1 billion in 2019 and $4.3 billion in 2022—a compound annual growth rate ("CAGR") of 30%. *See, e.g.*, https://digiworld.news/news/56020/application-containers-market-to-reach-43-billion-by-2022. Another source reported the application containers market had a market size of $5.45 billion in 2024 and estimated it to reach $19.41 billion in 2029—a CAGR of 28.89%. *See, e.g.*, https://www.mordorintelligence.com/industry-reports/application-container-market.

6. On information and belief, Oracle is a Delaware corporation with a principal place of business at 2300 Oracle Way, Austin, Texas 78741. Oracle may be served with process through its registered agent, the Corporation Trust Company, Corporation Trust Center, 1209 Orange St., Wilmington, Delaware 19801.

JURISDICTION AND VENUE

7. This action arises under the patent laws of the United States, Title 35 of the United States Code. This Court has original subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

8. This Court has personal jurisdiction over Defendant in this action because Defendant has committed acts within this District giving rise to this action, and has established minimum contacts with this forum such that the exercise of jurisdiction over Defendant would not offend traditional notions of fair play and substantial justice. Defendant, directly and through subsidiaries or intermediaries, has committed and continue to commit acts of infringement in this District by, among other things, importing, offering to sell, and selling products that infringe the asserted patents.

9. Venue is proper in this District because Defendant resides in this District, has a regular and established place of business in this District, and has committed acts of infringement within this District.

COUNT I

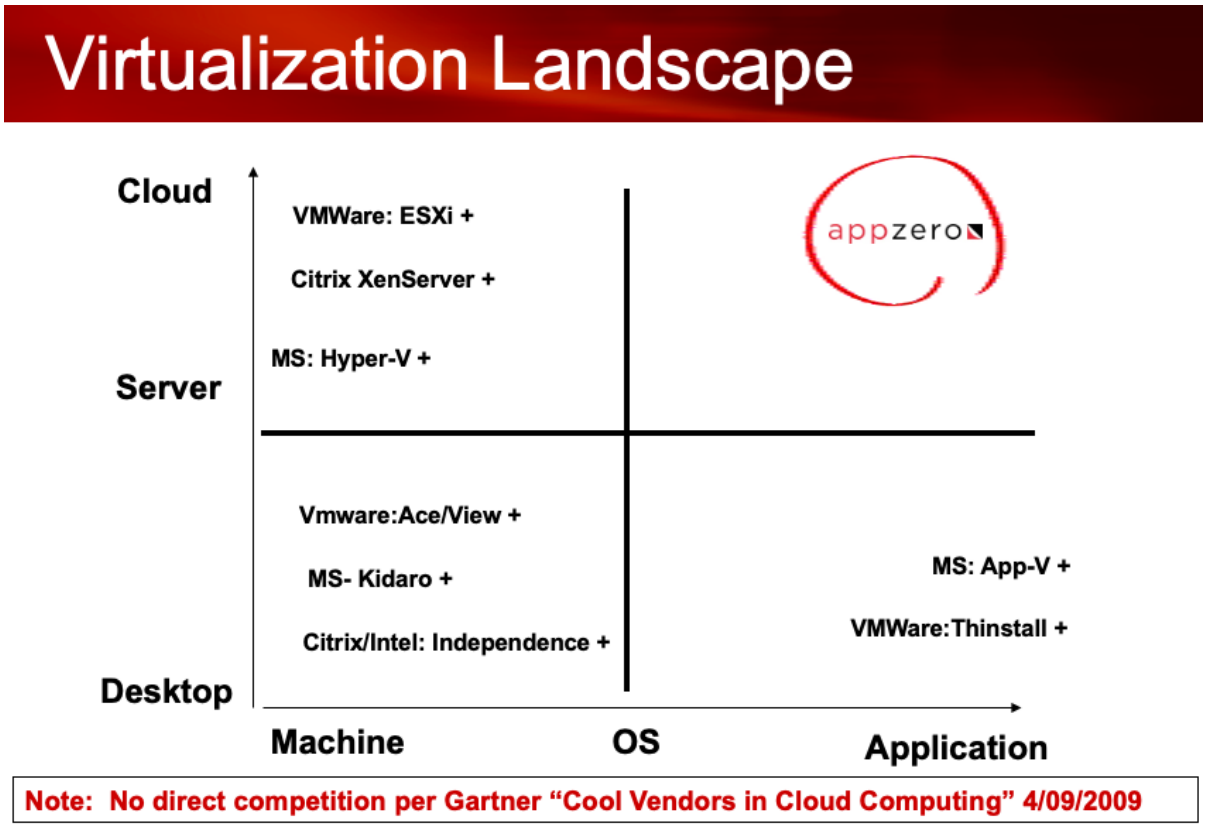
INFRINGEMENT OF U.S. PATENT NO. 7,519,814

10. VirtaMove realleges and incorporates by reference the foregoing paragraphs as if fully set forth herein.

11. VirtaMove owns all rights, title, and interest in U.S. Patent No. 7,519,814 ('814 patent), titled "System for Containerization of Application Sets," issued on April 14, 2009. A true and correct copy of the '814 Patent is attached as **Exhibit 1**.

12. The technology in the '814 Patent was not routine or conventional. Independent business intelligence provider Gartner reported that AppZero had no direct technological

competitors in "Cool Vendors in Cloud Computing" in April 2009. AppZero's software and services, unlike its competitors' conventional offerings, allowed its customers to deploy the technology described in the '814 patent.



13. Additionally, the USPTO examined the patented technology and concluded that the prior art "fails to anticipate, disclose, teach, or suggest alone, or in combination, at the time of the invention, the features as set forth of the claim . . . . Nowhere in the prior art is found collectively the *italicized* claim elements (i.e., the various aspects of applications software not being sharable between the plurality of secure (and isolated) containers of application software, and unique root file systems different from an operating system's root file system, so as to allow for different versions of the same operating running on the same system/server environment), at the ***time*** *of the invention*, serving to patently distinguish the prior art." Ex. 5 at 2–5. Thus, multiple aspects of the

claims were not found anywhere in the prior art of record, alone or in combination, such that they cannot be described as "routine" or "conventional." *Id.*

14. The technology claimed is not directed to an abstract idea. Rather, it is directed to a technological improvement in computer technology, and it provides a technological solution to a fundamentally technological problem. Indeed, the problems of securely sharing resources among a plurality of services in disparate computing environments and application portability across operating systems and disparate computing environments are inherently rooted in computer technology with no non-technological analog, nor can the solution be performed solely within the human mind. In part, the "invention provides a solution whereby a plurality of services can conveniently be installed on one or more servers in a cost effective and secure manner." Ex. 1 at 2:13–15. This is despite how "a collection of applications . . . must be separated with each application installed on an individual computer system," and despite how "certain applications require a specific version of operating system facilities and as such will not co-exist with applications that require another version." *Id.* at 1:27–41. Moreover, the technology is distinguished from the Virtual Machine approach in that, with Virtual Machines, "an operating system, including files and a kernel, must be deployed for each application" while the technology of the '814 patent requires only requires one operating system regardless of the number of application containers deployed, and it "offers the ability for applications to more effectively share a common compute platform, and also allow applications to be easily moved between platforms, without the requirement for a separate and distinct operating system for each application." *Id.* at 1:51–2:3

15. On information and belief, Defendant makes, uses, offers for sale, sells, and/or imports certain products ("Accused Products"), such as, e.g., Oracle Cloud Infrastructure ("OCI")

and Oracle Kubernetes Engine ("OCE"), that directly infringe, literally and/or under the doctrine of equivalents, claims of the '814 patent. The infringement of the Asserted Patents is also attributable to Defendant. Defendant directs and controls use of the Accused Products to perform acts that result in infringement the Asserted Patents, conditioning benefits on participation in the infringement and establishing the timing and manner of the infringement.

16. Defendant also knowingly and intentionally induces infringement of claims of the '814 patent in violation of 35 U.S.C. § 271(b). Defendant has had knowledge of the '814 patent and the infringing nature of the Accused Products at least as early as when this Complaint was filed and served on Defendant. Despite this knowledge of the '814 patent, Defendant continues to actively encourage and instruct its customers and end users (for example, through user manuals and online instruction materials on its website) to use the Accused Products in ways that directly infringe the '814 patent. Defendant does so, knowing and intending that its customers and end users will commit these infringing acts. Defendant also continues to make, use, offer for sale, sell, and/or import the Accused Products, despite its knowledge of the '814 patent, thereby specifically intending for and inducing its customers to infringe the '814 patent through the customers' normal and customary use of the Accused Products.

17. Defendant has also infringed, and continue to infringe, claims of the '814 patent by offering to commercially distribute, commercially distributing, making, and/or importing the Accused Products, which are used in practicing the process, or using the systems, of the patent, and constitute a material part of the invention. Defendant knows the components in the Accused Products to be especially made or especially adapted for use in infringement of the patent, not a staple article, and not a commodity of commerce suitable for substantial noninfringing use. Accordingly, Defendant has been, and currently are, contributorily infringing the '814 patent, in

violation of 35 U.S.C. § 271(c).

18. The Accused Products satisfy all claim limitations of one or more claims of the ’814 patent. A claim chart comparing independent claim 1 of the ’814 patent to a representative Accused Product is attached as **Exhibit 2**, which is hereby incorporated by reference in its entirety.

19. By making, using, offering for sale, selling and/or importing into the United States the Accused Products, Defendant has injured VirtaMove and is liable for infringement of the ’814 patent pursuant to 35 U.S.C. § 271.

20. As a result of Defendant’s infringement of the ’814 patent, VirtaMove is entitled to monetary damages in an amount adequate to compensate for Defendant’s infringement, but in no event less than a reasonable royalty for the use made of the invention by Defendant, together with interest and costs as fixed by the Court. VirtaMove is entitled to past damages under 35 U.S.C. § 287. VirtaMove has complied with the requirements of 35 U.S.C. § 287 and is not aware of any unmarked products that practice the claims of the ’814 patent. In the alternative, either VirtaMove’s product was marked before the filing of this lawsuit, or no requirement for marking applies.

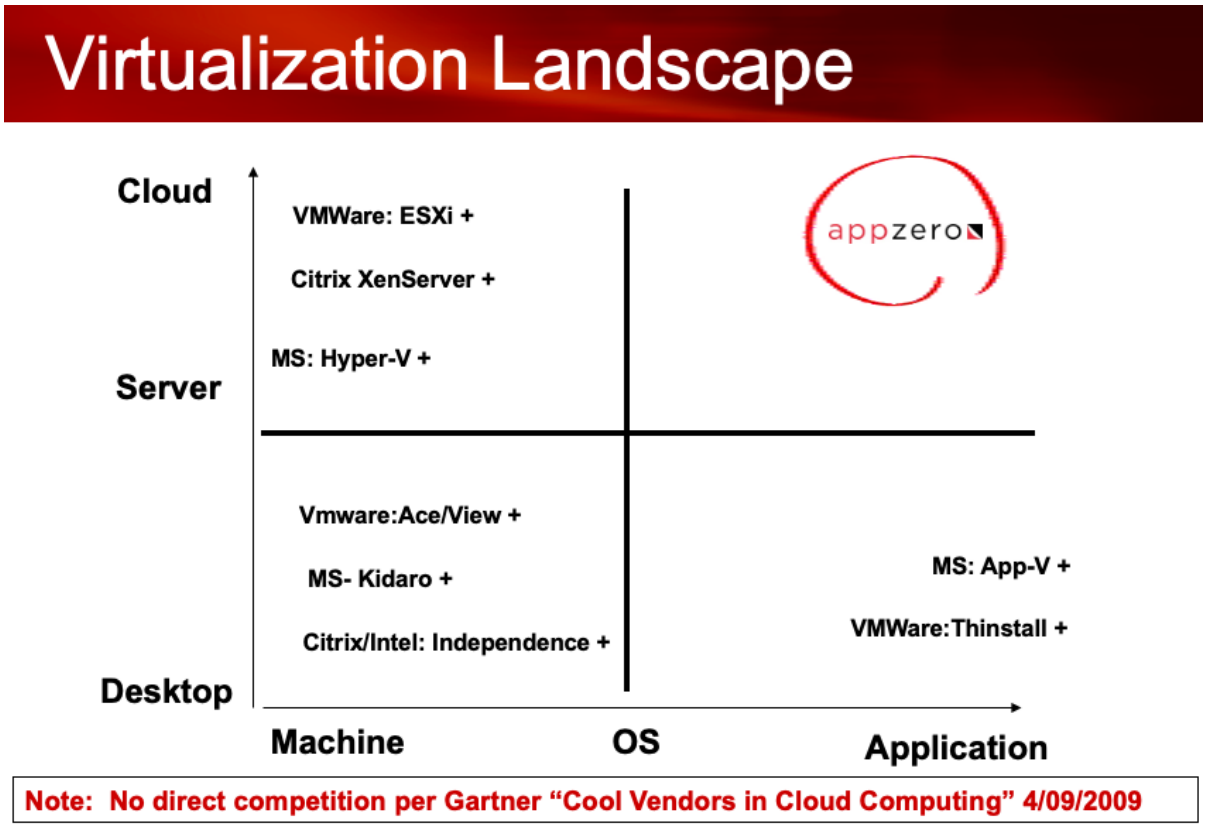
COUNT II

INFRINGEMENT OF U.S. PATENT NO. 7,784,058

21. VirtaMove realleges and incorporates by reference the foregoing paragraphs as if fully set forth herein.

22. VirtaMove owns all rights, title, and interest in U.S. Patent No. 7,784,058 (’058 patent), titled “Computing System Having User Mode Critical System Elements as Shared Libraries,” issued on August 24, 2010. A true and correct copy of the ’058 patent is attached as **Exhibit 3**.

23. The technology in the ’058 Patent was not routine or conventional. Independent business intelligence provider Gartner reported that AppZero had no direct technological competitors in “Cool Vendors in Cloud Computing” in April 2009. AppZero’s software and services, unlike its competitors’ conventional offerings, allowed its customers to deploy the technology described in the ’058 patent.



24. Additionally, the USPTO examined the patented technology and concluded that the none of the prior art discloses “a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more

of the plurality of software applications, and ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing the same function. iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously." Ex. 6 at 4–5. Thus, multiple aspects of the claims were not found anywhere in the prior art of record, alone or in combination, such that they cannot be described as "routine" or "conventional." *Id.*

25. The technology claimed is not directed to an abstract idea. Rather, it is directed to a technological improvement in computer technology, and it provides a technological solution to a fundamentally technological problem. Indeed, the problems of compatibility conflicts between applications and application portability across operating systems are inherently rooted in computer technology with no non-technological analog, nor can the solution be performed solely within the human mind. In part, the technology allows "a CSE to execute in the same context as an application. This then allows, among other things, an ability to deploy multiple instances of a CSE." Ex. 3 at 1:46–50. This is despite how "two software applications that require the same file, yet each requires a different version of the file will conflict," and despite how "two applications that require independent access to specific network services will conflict." *Id.* at 1:27–41. Moreover, the technology overcomes the "performance and operational differences" suffered by

other technological solutions. *Id.* at 1:41–45.

26. On information and belief, Defendant makes, uses, offers for sale, sells, and/or imports certain products ("Accused Products"), such as, e.g., Oracle Cloud Infrastructure ("OCI") and Oracle Kubernetes Engine ("OCE"), that directly infringe, literally and/or under the doctrine of equivalents. The infringement of the Asserted Patents is also attributable to Defendant. Defendant directs and controls use of the Accused Products to perform acts that result in infringement the Asserted Patents, conditioning benefits on participation in the infringement and establishing the timing and manner of the infringement.

27. Defendant also knowingly and intentionally induces infringement of claims of the '058 patent in violation of 35 U.S.C. § 271(b). Defendant has had knowledge of the '058 patent and the infringing nature of the Accused Products at least as early as when this Complaint was filed and served. Despite this knowledge of the '058 patent, Defendant continues to actively encourage and instruct its customers and end users (for example, through user manuals and online instruction materials on its website) to use the Accused Products in ways that directly infringe the '058 patent. Defendant does so knowing and intending that its customers and end users will commit these infringing acts. Defendant also continues to make, use, offer for sale, sell, and/or import the Accused Products, despite its knowledge of the '058 patent, thereby specifically intending for and inducing its customers to infringe the '058 patent through the customers' normal and customary use of the Accused Products.

28. Defendant has also infringed, and continue to infringe, claims of the '058 patent by offering to commercially distribute, commercially distributing, making, and/or importing the Accused Products, which are used in practicing the process, or using the systems, of the patent, and constitute a material part of the invention. Defendant knows the components in the Accused

Products to be especially made or especially adapted for use in infringement of the patent, not a staple article, and not a commodity of commerce suitable for substantial noninfringing use. Accordingly, Defendant has been, and currently are, contributorily infringing the ’058 patent, in violation of 35 U.S.C. § 271(c).

29. The Accused Products satisfy all claim limitations of one or more claims of the ’058 patent. A claim chart comparing claim 1 of the ’058 patent to a representative Accused Product is attached as **Exhibit 4**, which is hereby incorporated by reference in its entirety.

30. By making, using, offering for sale, selling and/or importing into the United States the Accused Products, Defendant has injured VirtaMove and are liable for infringement of the ’058 patent pursuant to 35 U.S.C. § 271.

31. As a result of Defendant’s infringement of the ’058 patent, VirtaMove is entitled to monetary damages in an amount adequate to compensate for Defendant’s infringement, but in no event less than a reasonable royalty for the use made of the invention by Defendant, together with interest and costs as fixed by the Court. VirtaMove is entitled to past damages under 35 U.S.C. § 287. VirtaMove has complied with the requirements of 35 U.S.C. § 287. 35 U.S.C. § 287 requires the marking of a “patented articles,” however VirtaMove makes, offers for sale, and/or sells software and software support services. VirtaMove is unaware of instances where it made, offered for sale, and/or sold “a processor,” as required by claim 1 of the ’058 patent, with these products. Thus, VirtaMove is not required to mark any of these products that it makes, offers for sale, and/or sells. In the alternative, to the extent VirtaMove is found to have “made,” “offered for sale,” and/or “sold” “patented articles,” including to the extent VirtaMove is found to have made, offered for sale, and/or sold products with “a processor” as required by claim 1 of the ’058 patent, VirtaMove’s products were marked before the filing of this lawsuit.

PRAYER FOR RELIEF

WHEREFORE, VirtaMove respectfully requests that this Court enter:

a. A judgment in favor of VirtaMove that Defendant has infringed, either literally and/or under the doctrine of equivalents, each of the Asserted Patents;

b. A permanent injunction prohibiting Defendant from further acts of infringement of each of the ’814 and ’058 patents;

c. A judgment and order requiring Defendant to pay VirtaMove its damages, costs, expenses, and pre-judgment and post-judgment interest for Defendant’s infringement of each of the Asserted Patents;

d. A judgment and order requiring Defendant to provide an accounting and to pay supplemental damages to VirtaMove, including without limitation, pre-judgment and post-judgment interest;

e. A judgment and order finding that this is an exceptional case within the meaning of 35 U.S.C. § 285 and awarding to VirtaMove its reasonable attorneys’ fees against Defendant; and

f. Any and all other relief as the Court may deem appropriate and just under the circumstances.

DEMAND FOR JURY TRIAL

VirtaMove, under Rule 38 of the Federal Rules of Civil Procedure, requests a trial by jury of any issues so triable by right.

Dated: April 11, 2025

Respectfully submitted,

/s/ Reza Mirzaie

Reza Mirzaie (CA SBN 246953)
rmirzaie@raklaw.com
Marc A. Fenster (CA SBN 181067)
mfenster@raklaw.com
Neil A. Rubin (CA SBN 250761)
nrubin@raklaw.com
Jacob R. Buczko (CA SBN 269408)
jbuczko@raklaw.com
James S. Tsuei (CA SBN 285530)
jtsuei@raklaw.com
James A. Milkey (CA SBN 281283)
jmilkey@raklaw.com
Christian W. Conkle (CA SBN 306374)
cconkle@raklaw.com
Jonathan Ma (CA SBN 312773)
jma@raklaw.com
Daniel Kolko (CA SBN 341680)
dkolko@raklaw.com
Mackenzie Paladino (NY SBN 6039366)
mpaladino@raklaw.com
RUSS AUGUST & KABAT
12424 Wilshire Boulevard, 12th Floor
Los Angeles, CA 90025
Telephone: (310) 826-7474

Qi (Peter) Tong (TX SBN 24119042)
ptong@raklaw.com
RUSS AUGUST & KABAT
8080 N. Central Expy., Suite 1503
Dallas, TX 75206
Telephone: (310) 826-7474

Attorneys for Plaintiff VirtaMove, Corp.

CERTIFICATE OF SERVICE

I certify that on April 11, 2025, a true and correct copy of the foregoing document was electronically filed with the Court and served on all parties of record via the Court's CM/ECF system.

/s/ Reza Mirzaie
Reza Mirzaie

Exhibit 1

US007519814B2

(12) **United States Patent**
Rochette et al.

(10) **Patent No.: US 7,519,814 B2**
(45) **Date of Patent: Apr. 14, 2009**

(54) **SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS**

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul O'Leary**, Kanata (CA); **Dean Huffman**, Kanata (CA)

(73) Assignee: **Trigence Corp.**, Ottawa (CA)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 933 days.

(21) Appl. No.: **10/939,903**

(22) Filed: **Sep. 13, 2004**

(65) **Prior Publication Data**

US 2005/0060722 A1 Mar. 17, 2005

Related U.S. Application Data

(60) Provisional application No. 60/512,103, filed on Oct. 20, 2003, provisional application No. 60/502,619, filed on Sep. 15, 2003.

(51) **Int. Cl.**
G06F 3/00 (2006.01)

(52) **U.S. Cl.** **713/167**; 713/164; 709/248; 709/214; 719/319

(58) **Field of Classification Search** 713/167; 719/319
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,381,742 B2 * 4/2002 Forbes et al. 717/176

(Continued)

FOREIGN PATENT DOCUMENTS

WO WO 02/06941 A 1/2002

WO WO 2006/039181 A 4/2006

OTHER PUBLICATIONS

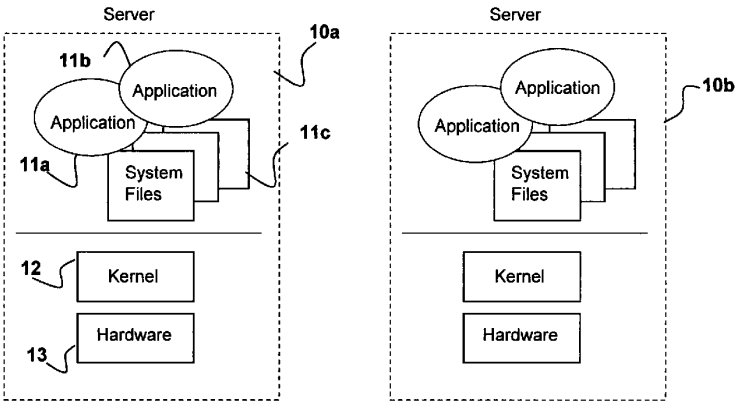
Soltesz, S., et al, 'Container-based Operating System Virtualization:A Scalable, High-performance Alternative to Hypervisors', SIGOPS Oper. Syst. Rev., vol. 41, No. 3. (Jun. 2007), pp. 275-287, entire article, http://delivery.acm.org/10.1145/1280000/1273025/p275-soltesz.pdf?key1=1273025&key2=0279528221&coll=GUIDE&dl=GUIDE&CFID=13091697& CFTOKEN=4667.*

Primary Examiner—Kambiz Zand
Assistant Examiner—Ronald Baum
(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A.

(57) **ABSTRACT**

A system is disclosed having servers with operating systems that may differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor. This invention discloses a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method of this invention requires storing in memory accessible to at least some of the servers a plurality of secure containers of application software. Each container includes one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers. The set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems. The containers of application software exclude a kernel; and some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server.

34 Claims, 17 Drawing Sheets



U.S. PATENT DOCUMENTS

6,847,970	B2 *	1/2005	Keller et al.	707/100
7,076,784	B1 *	7/2006	Russell et al.	719/315
7,287,259	B2 *	10/2007	Grier et al.	719/331
2002/0004854	A1	1/2002	Hartley	
2002/0174215	A1	11/2002	Schaefer	709/224
2003/0101292	A1	5/2003	Fisher	

* cited by examiner

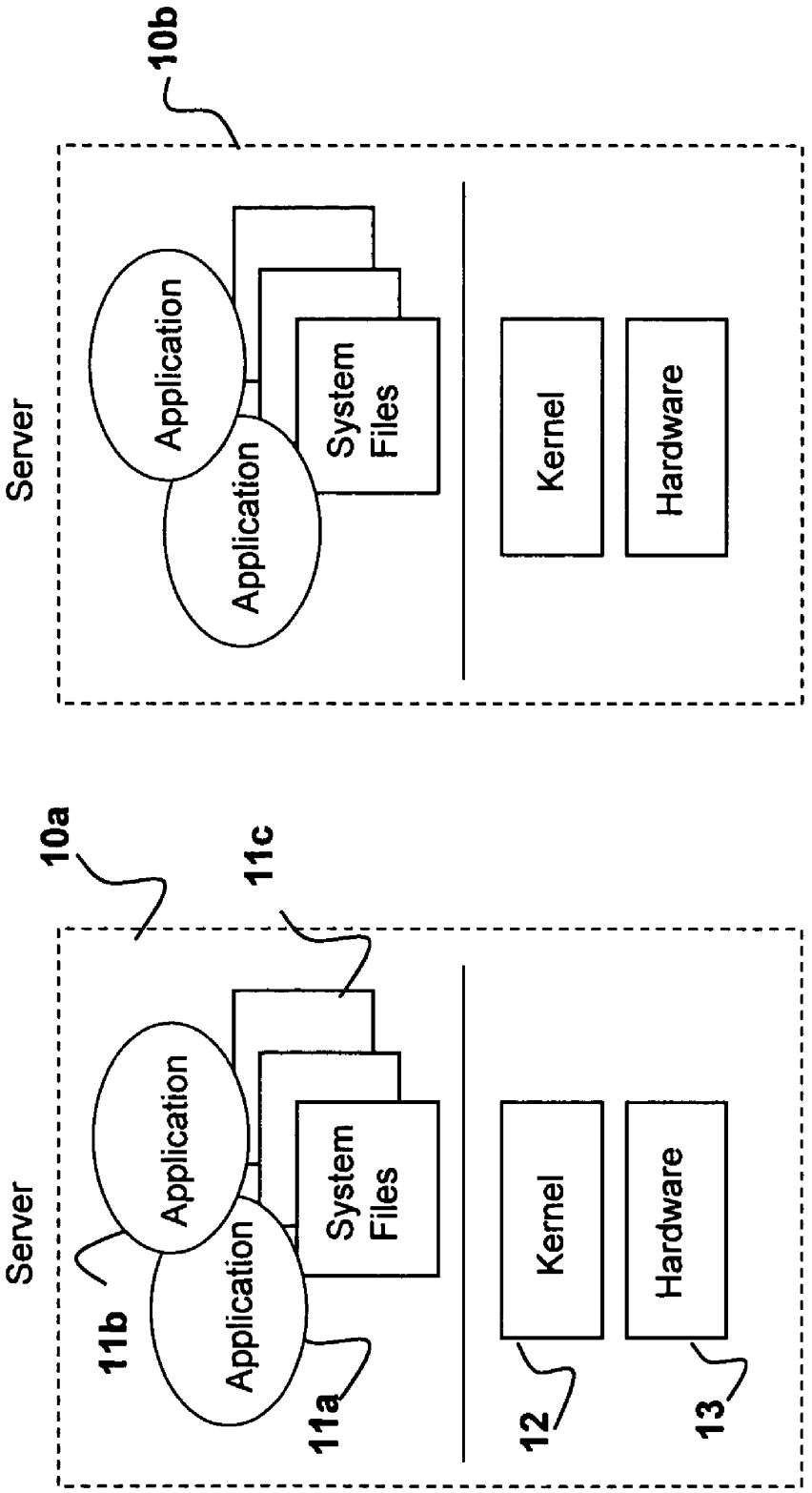

10b
Server
Application
Application
System Files
Kernel
Hardware
10a
11c
Server
Application
Application
System Files
Kernel
Hardware
11b
11a
12
13

Figure 1

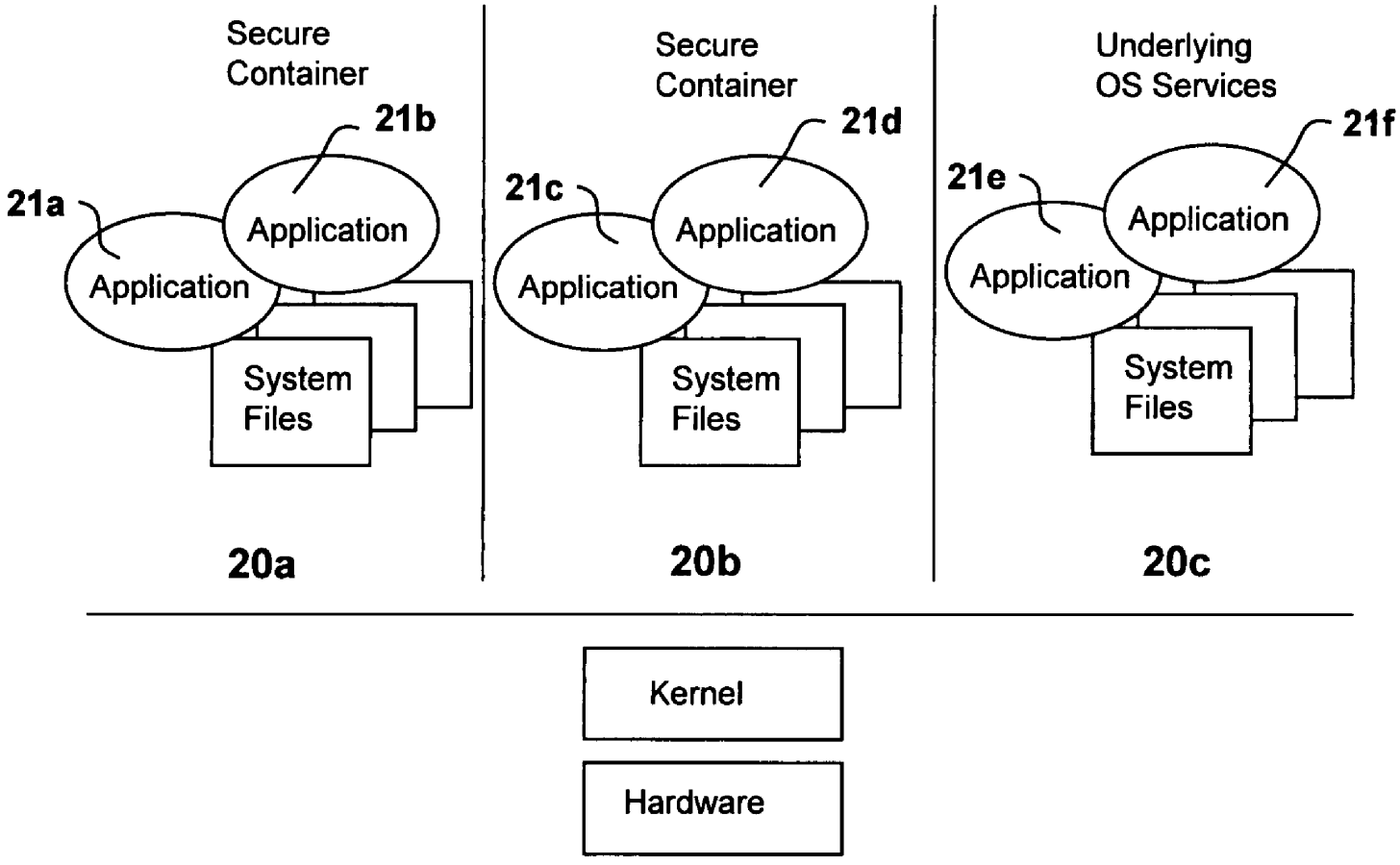

Secure Container
21b
21a
Application
Application
System Files
20a
Secure Container
21d
21c
Application
Application
System Files
20b
Underlying OS Services
21f
21e
Application
Application
System Files
20c
Kernel
Hardware

Figure 2

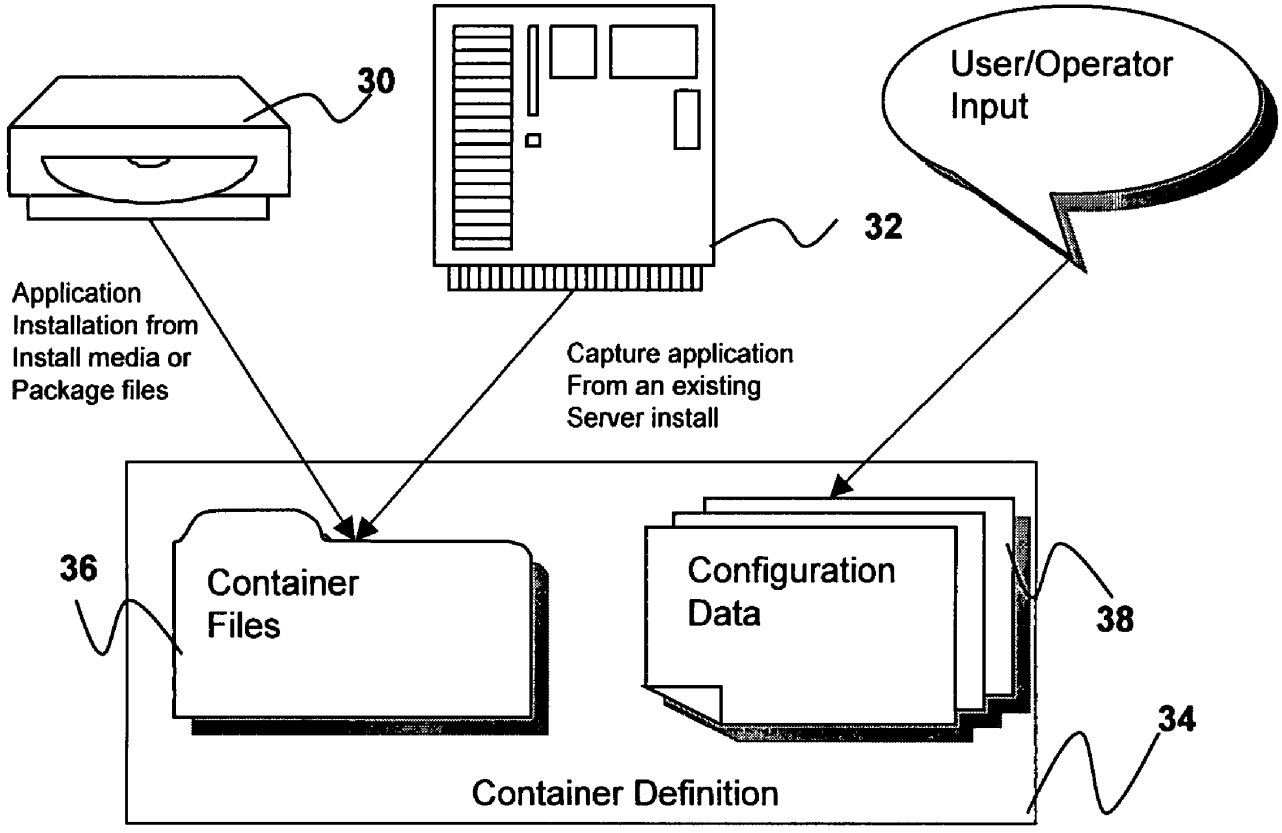

30
32
User/Operator Input
Application Installation from Install media or Package files
Capture application From an existing Server install
36
Container Files
Configuration Data
38
34
Container Definition

Figure 3

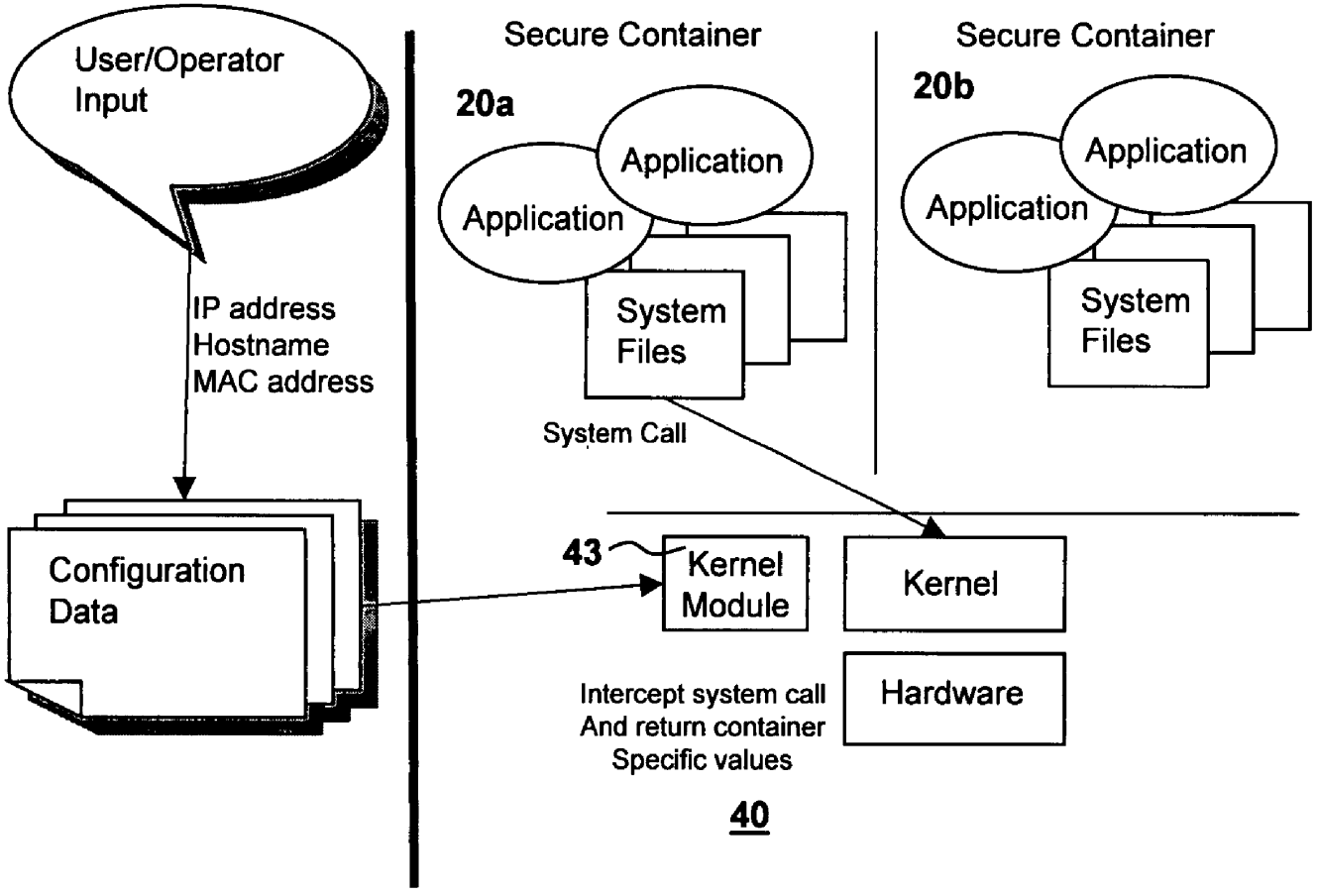

User/Operator Input
IP address
Hostname
MAC address
Configuration Data
Secure Container
20a
Application
Application
System Files
System Call
Secure Container
20b
Application
Application
System Files
43
Kernel Module
Kernel
Hardware
Intercept system call
And return container
Specific values
40

Figure 4

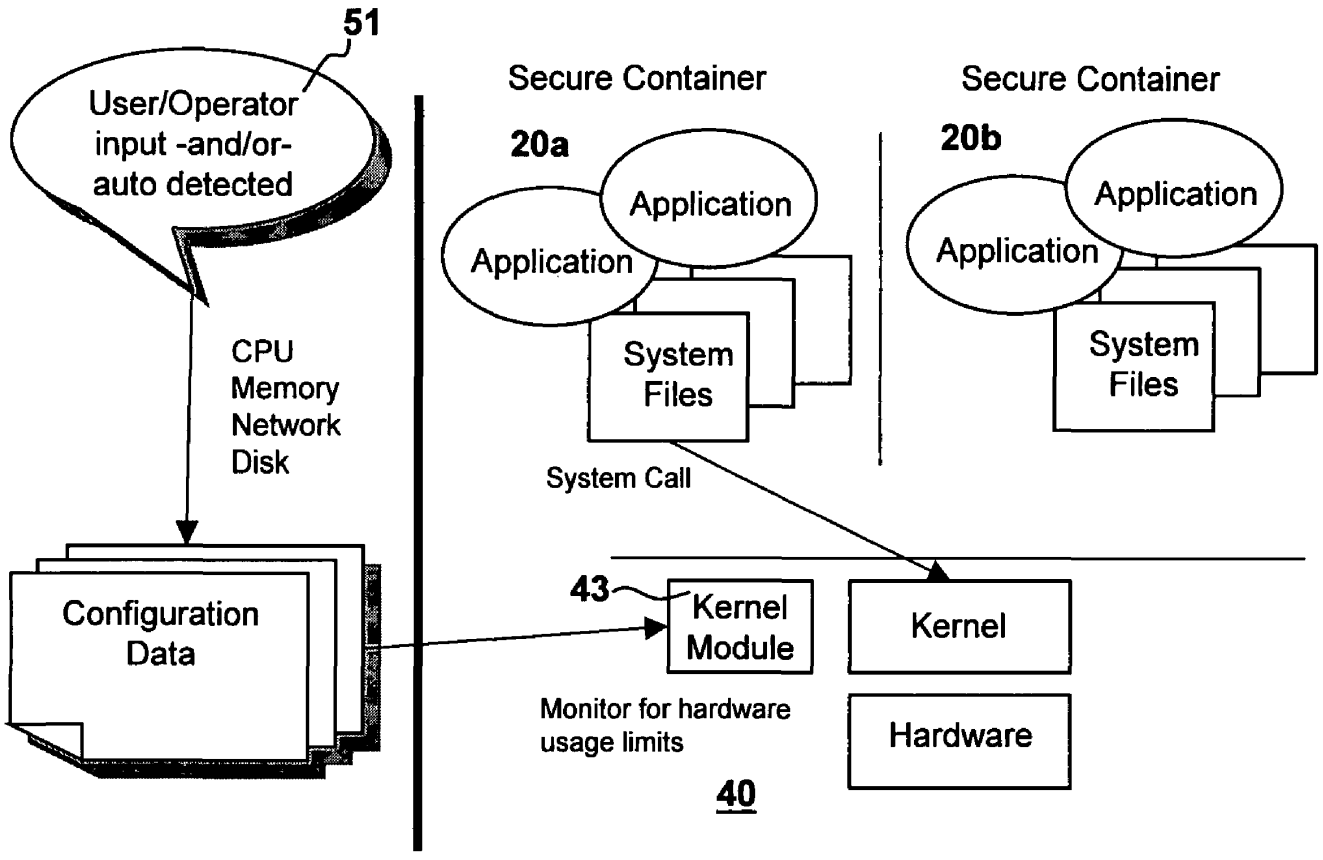

51
User/Operator input -and/or- auto detected
CPU
Memory
Network
Disk
Configuration Data
Secure Container
20a
Application
Application
System Files
Secure Container
20b
Application
Application
System Files
System Call
43
Kernel Module
Monitor for hardware usage limits
Kernel
Hardware
40

Figure 5

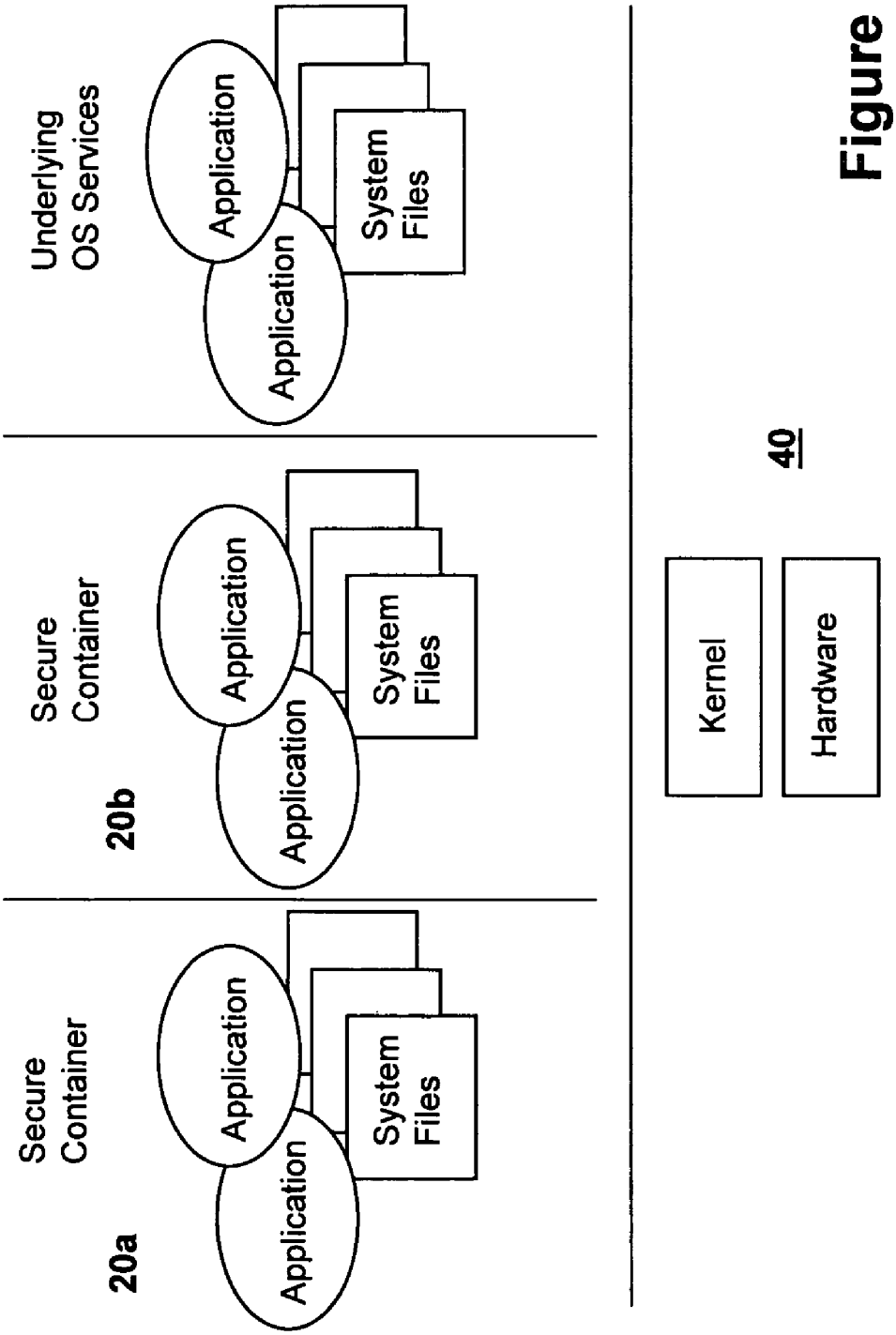

Underlying
OS Services
Application
Application
System
Files
Secure
Container
20b
Application
Application
System
Files
Secure
Container
20a
Application
Application
System
Files
40
Kernel
Hardware

Figure 6

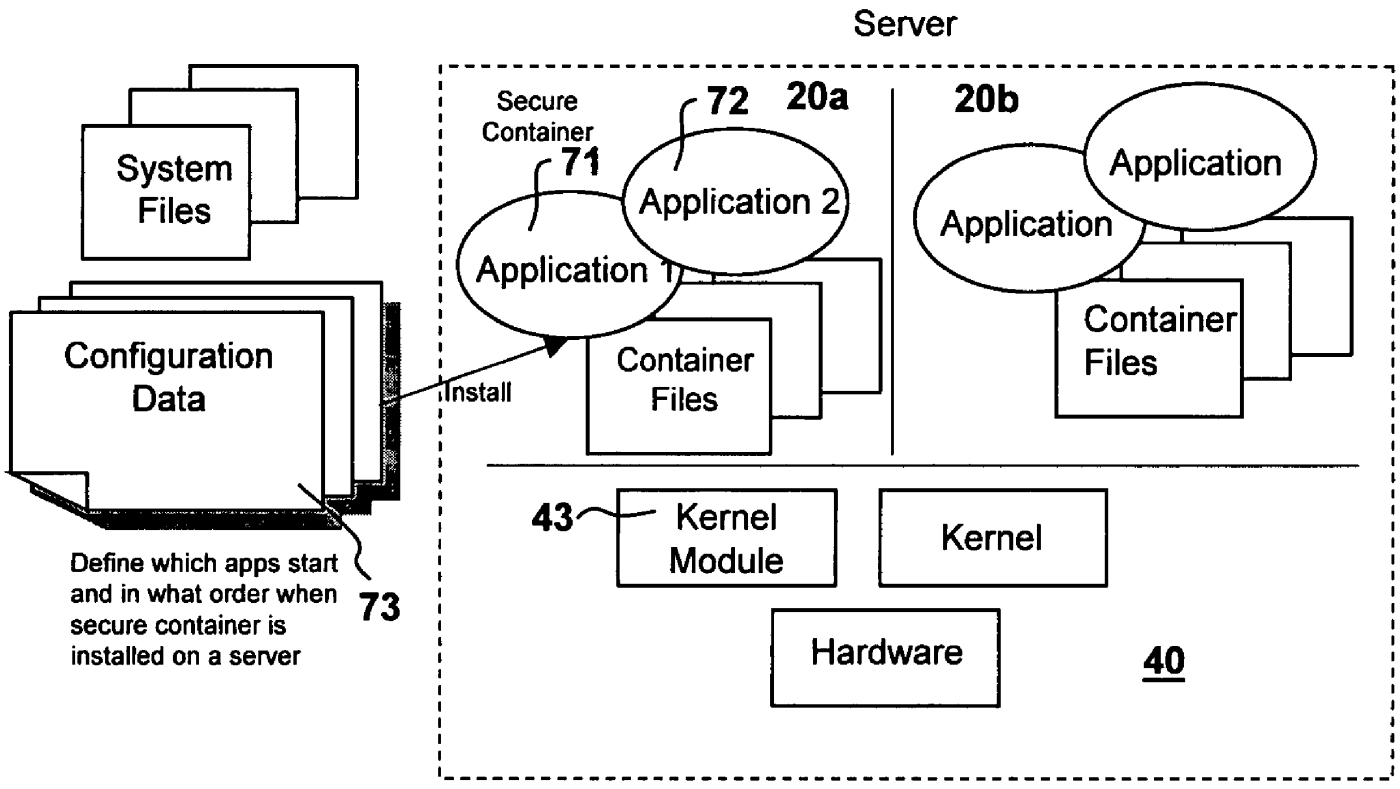

Server
System Files
Configuration Data
Define which apps start and in what order when secure container is installed on a server
73
Install
Secure Container
71
72
20a
Application 1
Application 2
Container Files
20b
Application
Application
Container Files
43
Kernel Module
Kernel
Hardware
40

Figure 7

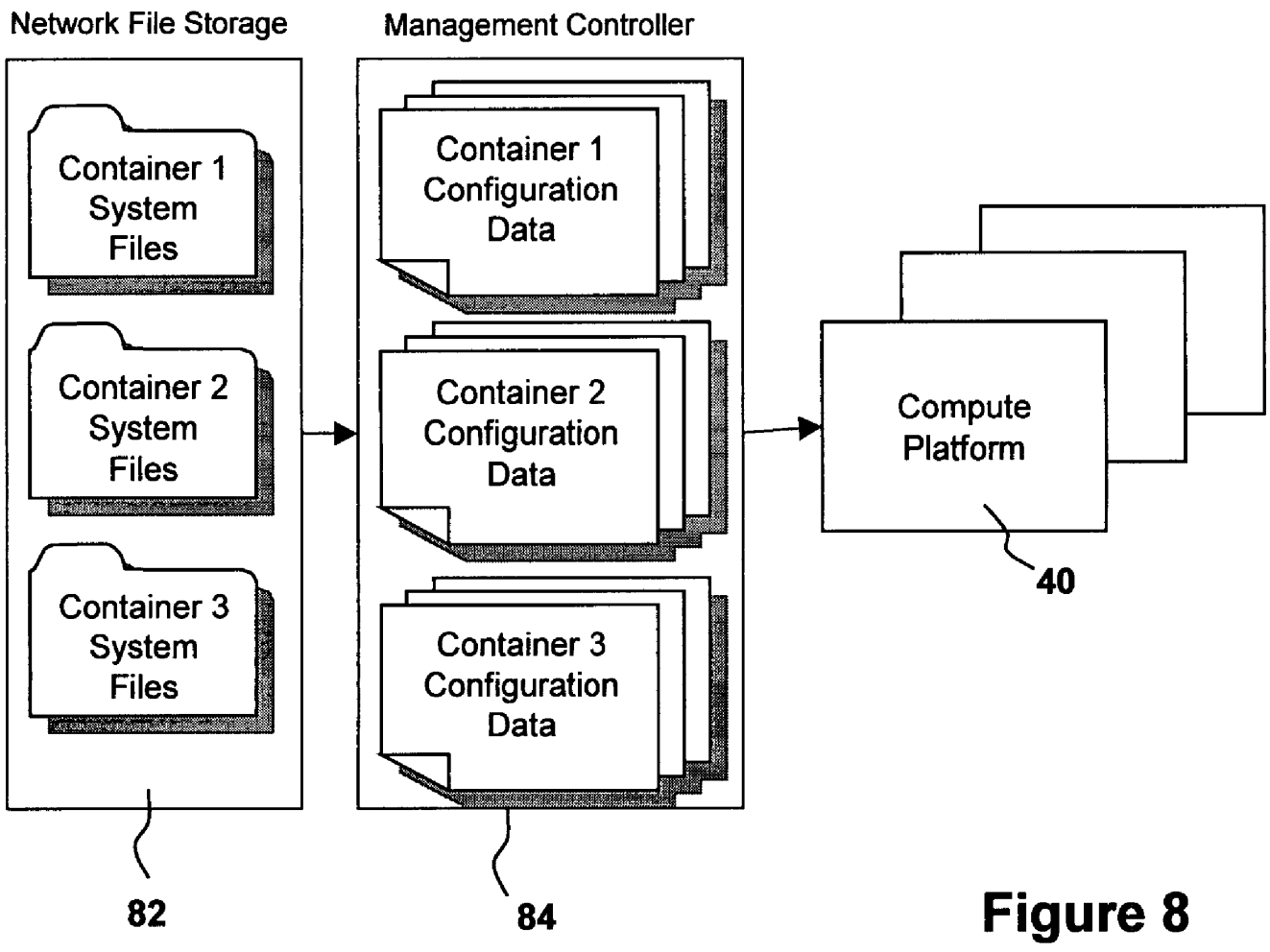

Network File Storage
Management Controller
Container 1 System Files
Container 2 System Files
Container 3 System Files
Container 1 Configuration Data
Container 2 Configuration Data
Container 3 Configuration Data
Compute Platform
82
84
40

Figure 8

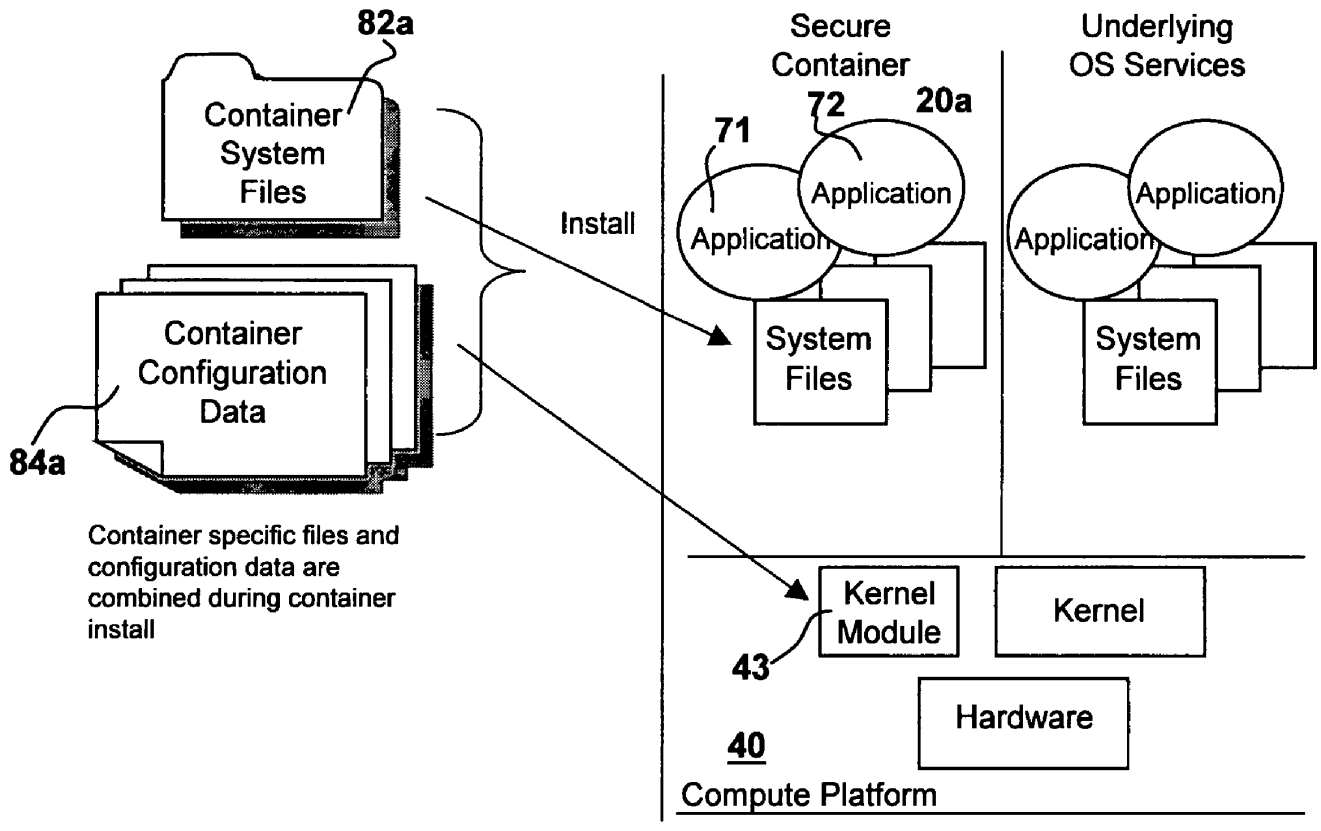

82a
Container System Files
Container Configuration Data
84a
Container specific files and configuration data are combined during container install
Install
Secure Container
Underlying OS Services
71
72
20a
Application
Application
System Files
Application
Application
System Files
Kernel Module
43
Kernel
Hardware
40
Compute Platform

Figure 9

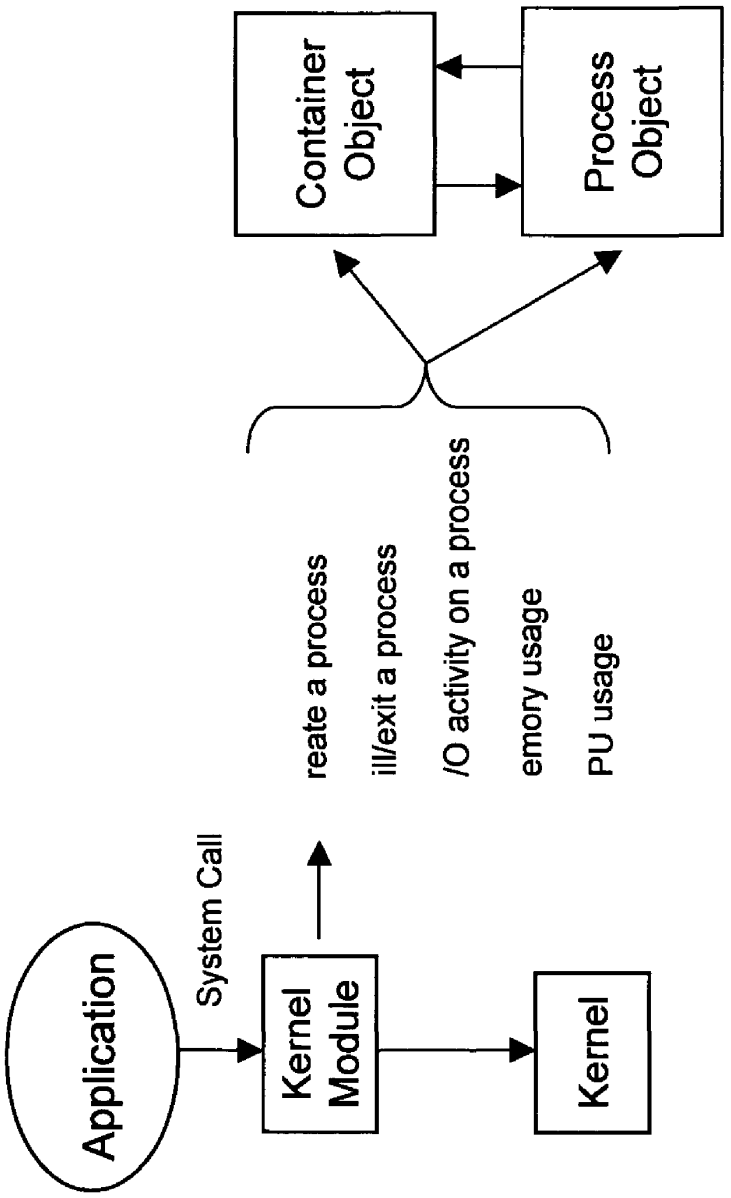

Application
System Call
Kernel Module
Kernel
reate a process
ill/exit a process
/O activity on a process
emory usage
PU usage
Container Object
Process Object

Figure 10

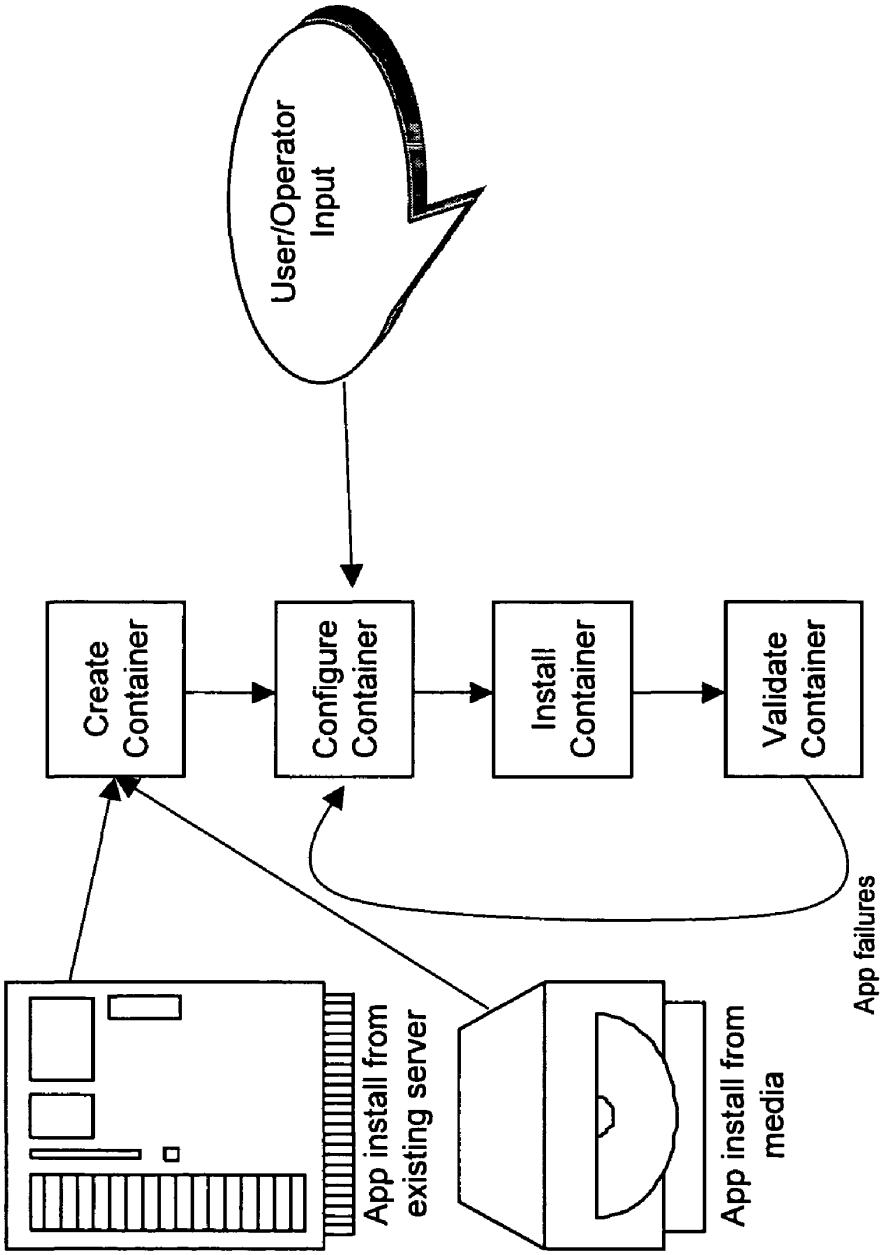

User/Operator Input
Create Container
Configure Container
Install Container
Validate Container
App failures
App install from existing server
App install from media

Figure 11

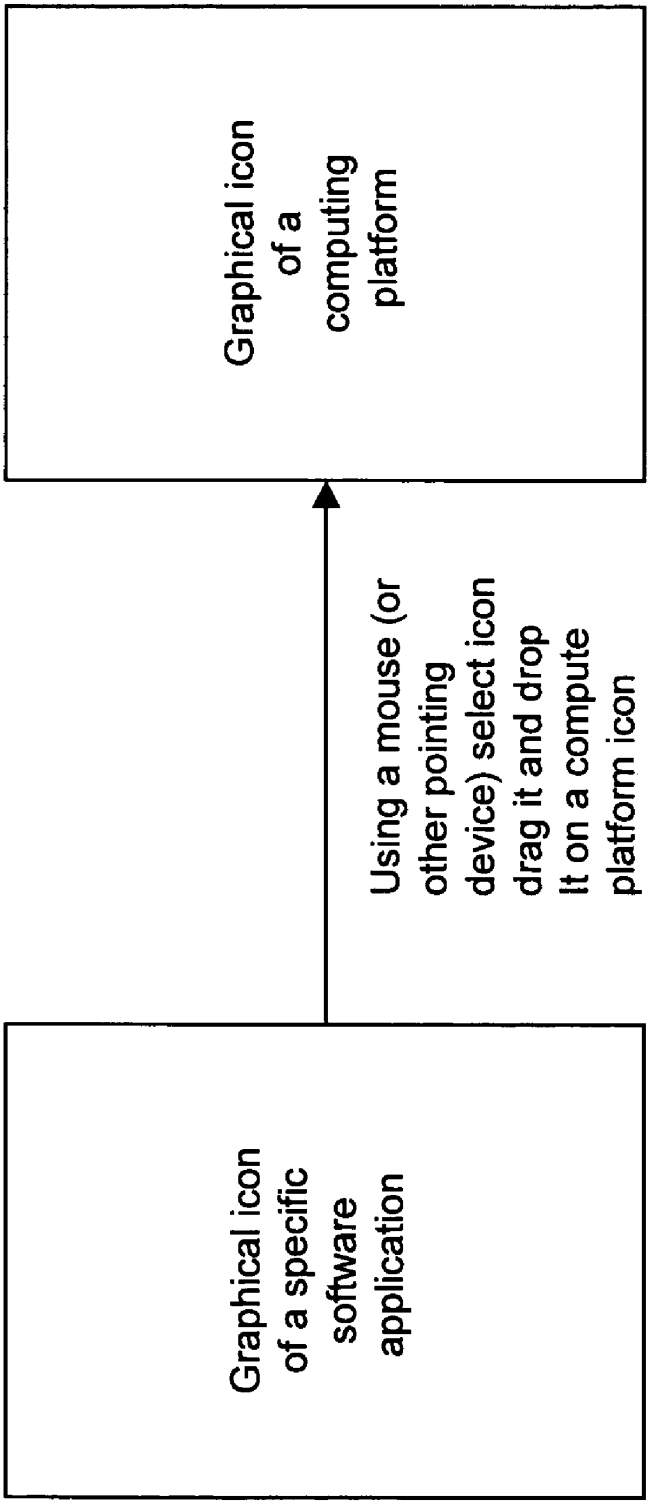

Graphical icon of a specific software application
Using a mouse (or other pointing device) select icon drag it and drop It on a compute platform icon
Graphical icon of a computing platform

Figure 12

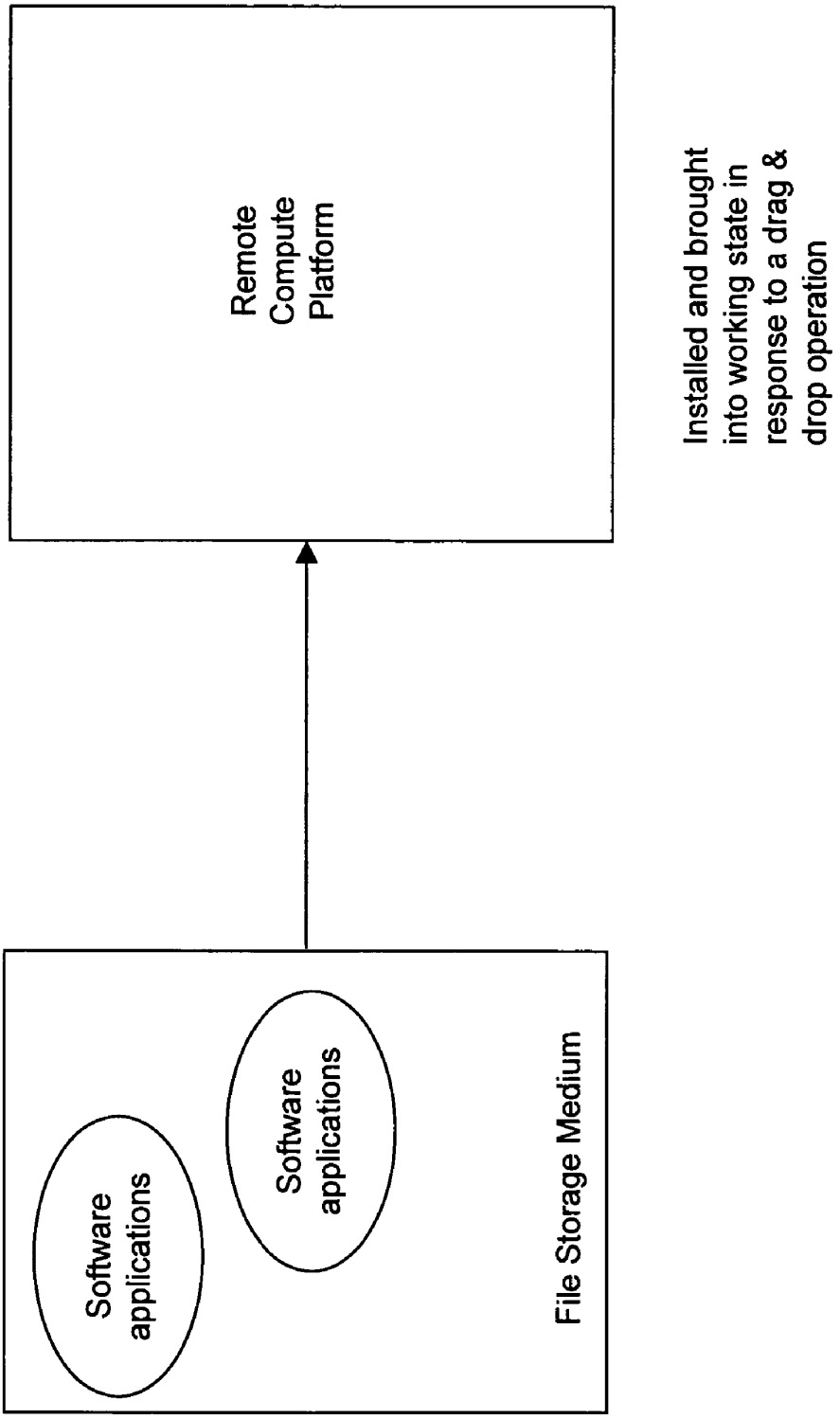

Software applications
Software applications
File Storage Medium
Remote Compute Platform
Installed and brought into working state in response to a drag & drop operation

Figure 13

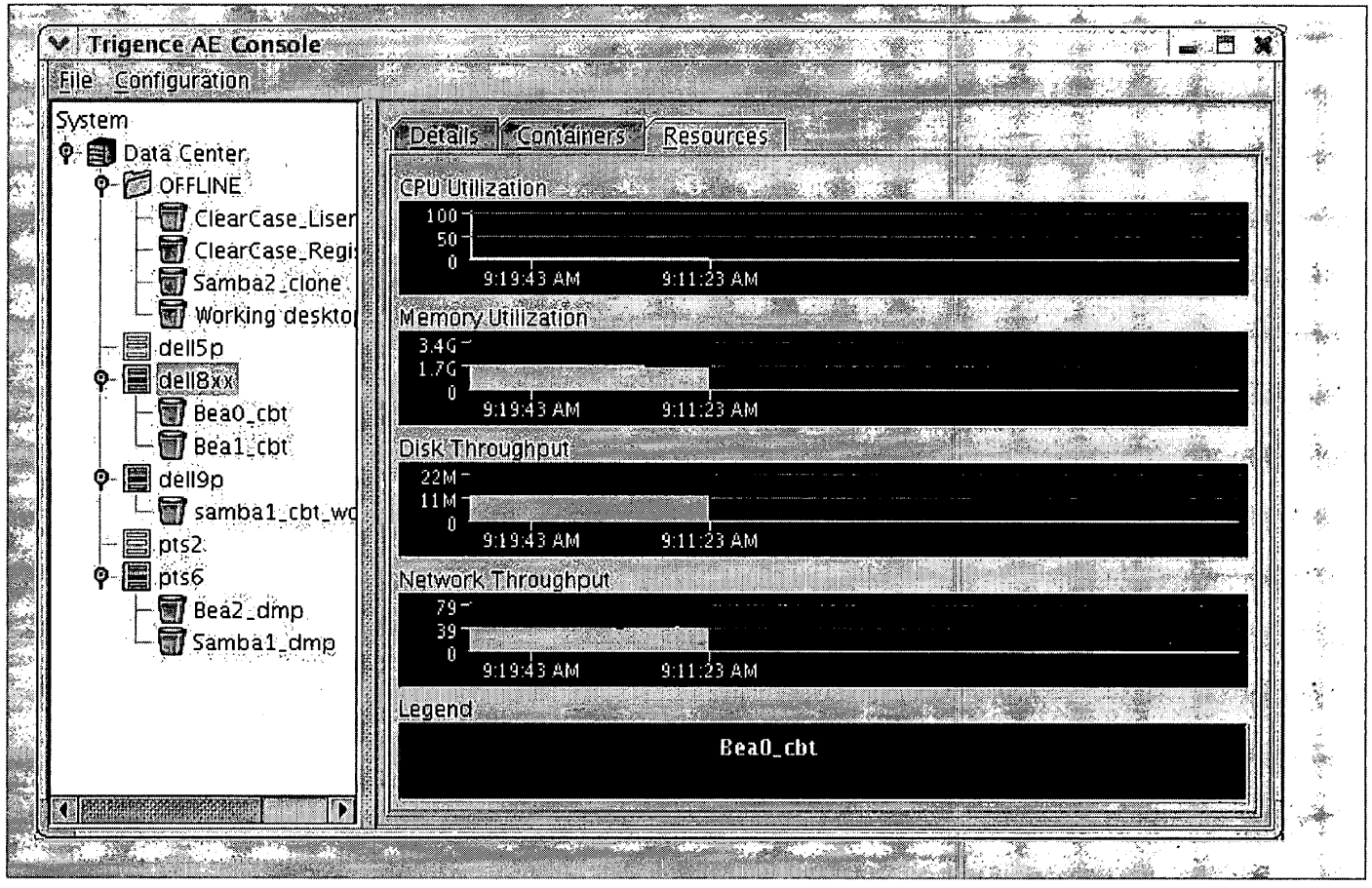

Trigence AE Console
File Configuration
System
Data Center
OFFLINE
ClearCase_Liser
ClearCase_Regi
Samba2_clone
Working deskto
dell5p
dell8xx
Bea0_cbt
Bea1_cbt
dell9p
samba1_cbt_wc
pts2
pts6
Bea2_dmp
Samba1_dmp
Details
Containers
Resources
CPU Utilization
100
50
0
9:19:43 AM
9:11:23 AM
Memory Utilization
3.4G
1.7G
0
9:19:43 AM
9:11:23 AM
Disk Throughput
22M
11M
0
9:19:43 AM
9:11:23 AM
Network Throughput
79
39
0
9:19:43 AM
9:11:23 AM
Legend
Bea0_cbt

Figure 14

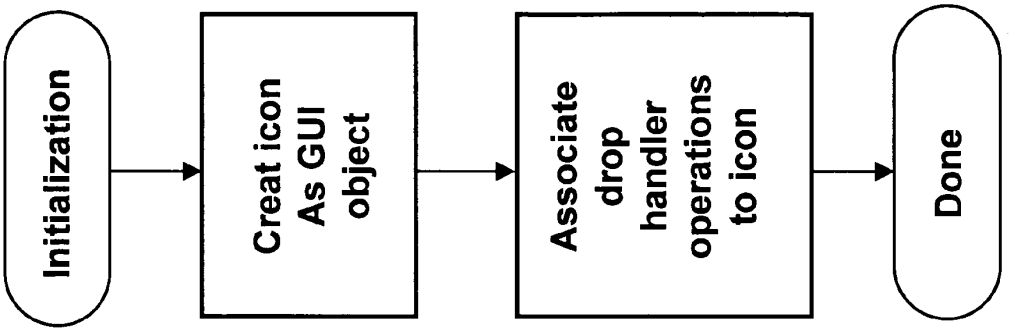

Initialization
Creat icon As GUI object
Associate drop handler operations to icon
Done

Figure 15

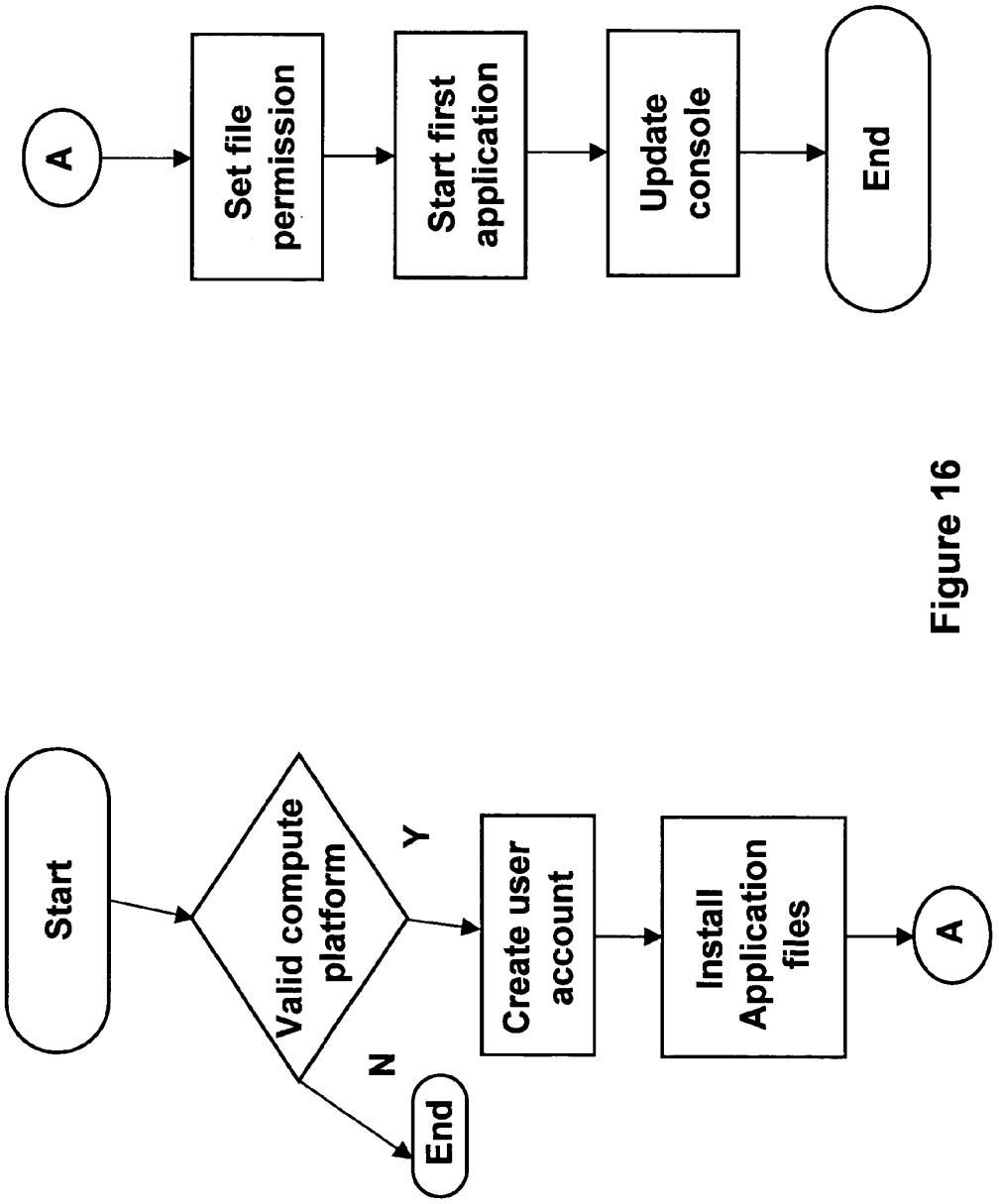

Start
Valid compute platform
Y
N
End
Create user account
Install Application files
A
A
Set file permission
Start first application
Update console
End

Figure 16

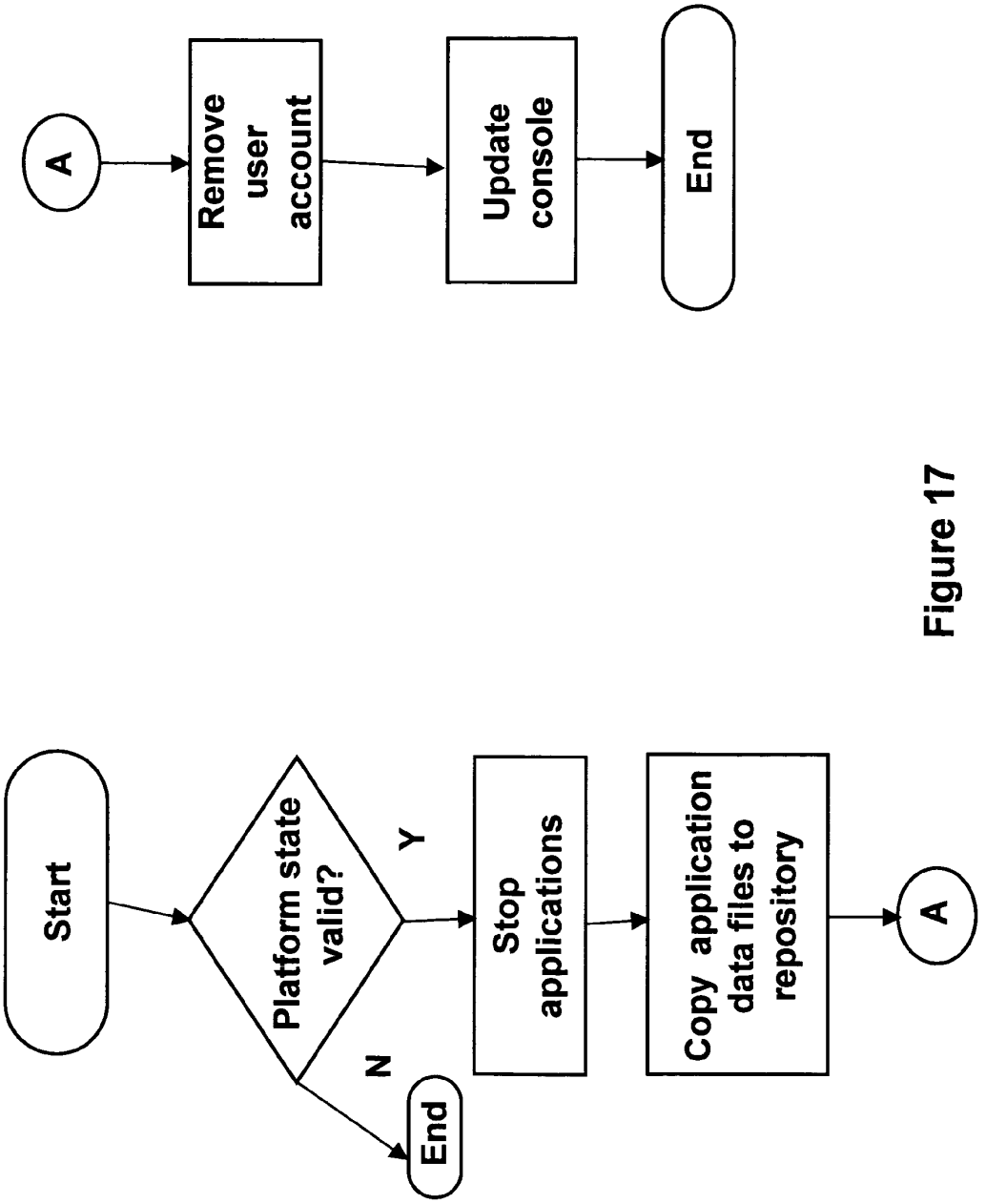

Start
Platform state valid?
Y
N
End
Stop applications
Copy application data files to repository
A
A
Remove user account
Update console
End

Figure 17

SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority of U.S. Provisional Patent Application No. 60/502,619 filed Sep. 15, 2003 and 60/512,103 filed Oct. 20, 2003, which are incorporated herein by reference for all purposes.

FIELD OF THE INVENTION

The invention relates to computer software. In particular, the invention relates to management and deployment of server applications.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that application programs share common resources. It is traditionally the task of an operating system to provide a mechanism to safely and effectively control access to shared resources required by application programs. This is the foundation of multi-tasking systems that allow multiple disparate applications to co-exist on a single computer system.

The current state of the art creates an environment where a collection of applications, each designed for a distinct function, must be separated with each application installed on an individual computer system.

In some instances this separation is necessitated by conflict over shared resources, such as network port numbers that would otherwise occur. In other situations the separation is necessitated by the requirement to securely separate data such as files contained on disk-based storage and/or applications between disparate users.

In yet other situations, the separation is driven by the reality that certain applications require a specific version of operating system facilities and as such will not co-exist with applications that require another version.

As computer system architecture is applied to support specific services, it inevitably requires that separate systems be deployed for different sets of applications required to perform and or support specific services. This fact, coupled with increased demand for support of additional application sets, results in a significant increase in the number of computer systems being deployed. Such deployment makes it quite costly to manage the number of systems required to support several applications.

There are existing solutions that address the single use nature of computer systems. These solutions each have limitations, some of which this invention will address. Virtual Machine technology, pioneered by VmWare, offers the ability for multiple application/operating system images to effectively co-exist on a single compute platform. The key difference between the Virtual Machine approach and the approach described herein is that in the former an operating system, including files and a kernel, must be deployed for each application while the latter only requires one operating system regardless of the number of application containers deployed. The Virtual Machine approach imposes significant performance overhead. Moreover, it does nothing to alleviate the requirement that an operating system must be licensed, managed and maintained for each application. The invention described herein offers the ability for applications to more effectively share a common compute platform, and also allow

applications to be easily moved between platforms, without the requirement for a separate and distinct operating system for each application.

A product offered by Softricity, called SoftGrid®, offers what is described as Application Virtualization. This product provides a degree of separation of an application from the underlying operating system. Unlike Virtual Machine technology a separate operating system is not required for each application. The SoftGrid® product does not isolate applications into distinct environments. Applications executing within a SoftGrid® environment don't possess a unique identity.

This invention provides a solution whereby a plurality of services can conveniently be installed on one or more servers in a cost effective and secure manner.

The following definitions are used herein:

Disparate computing environments: Environments where computers are stand-alone or where there are plural computers and where they are unrelated.

Computing platform: A computer system with a single instance of a fully functional operating system installed is referred to as a computing platform.

Container: An aggregate of files required to successfully execute a set of software applications on a computing platform is referred to as a container. A container is not a physical container but a grouping of associated files, which may be stored in a plurality of different locations that is to be accessible to, and for execution on, one or more servers. Each container for use on a server is mutually exclusive of the other containers, such that read/write files within a container cannot be shared with other containers. The term "within a container", used within this specification, is to mean "associated with a container". A container comprises one or more application programs including one or more processes, and associated system files for use in executing the one or more processes; but containers do not comprise a kernel; each container has its own execution file associated therewith for starting one or more applications. In operation, each container utilizes a kernel resident on the server that is part of the operating system (OS) the container is running under to execute its applications.

Secure application container: An environment where each application set appears to have individual control of some critical system resources and/or where data within each application set is insulated from effects of other application sets is referred to as a secure application container.

Consolidation: The ability to support multiple, possibly conflicting, sets of software applications on a single computing platform is referred to as consolidation.

System files: System files are files provided within an operating system and which are available to applications as shared libraries and configuration files.

By way of example, Linux Apache uses the following shared libraries, supplied by the OS distribution, which are "system" files.

/usr/lib/libz.so.1
/lib/libssl.so.2
/lib/libcrypto.so.2
/usr/lib/libaprutil.so.0
/usr/lib/libgdbm.so.2
/lib/libdb-4.0.so
/usr/lib/libexpat.so.0
/usr/lib/libapr.so.0
/lib/i686/libm.so.6
/lib/libcrypt.so.1

/lib/libnsl.so.1
/lib/libdl.so.2
/lib/i686/libpthread.so.0
/lib/i686/libc.so.6
/lib/ld-linux.so.2

Apache uses the following configuration files, also provided with the OS distribution:

/etc/hosts
/etc/httpd/conf
/etc/httpd/conf.d
/etc/httpd/logs
/etc/httpd/modules
/etc/httpd/run

By way of example, together these shared library files and configuration files form system files provided by the operating system. There may be any number of other files included as system files. Additional files might be included, for example, to support maintenance activities or to start other network services to be associated with a container.

SUMMARY OF THE INVENTION

In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method is provided for providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications may be executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service. The method comprises the steps of:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, and wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files resident on the server prior to said storing step.

In accordance with another aspect of the invention a computer system is provided for performing a plurality of tasks each comprising a plurality of processes comprising:

a plurality of secure stored containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files having its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a MAC address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and, a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.

In accordance with a broad aspect, the invention provides a method of establishing a secure environment for executing, on a computer system, a plurality of applications that require shared resources. The method involves, associating one or more respective software applications and required system files with a container. A plurality of containers have these containerized files within.

Containers residing on or associated with a respective server each have a resource allocation associated therewith to allow the at least one respective application or a plurality of applications residing in the container to be executed on the computer system according to the respective resource allocation without conflict with other applications in other containers which have their given set of resources and settings.

Containers, and therefore the applications within containers, are provided with a secure environment from which to execute. In some embodiments of the invention, each group of applications is provided with a secure storage medium.

In some embodiments of the invention the method involves containerizing the at least one respective application associated respective secure application container, the respective secure application container being storable in a storage medium.

In some embodiments of the invention, groups of applications are containerized into a single container for creating a single service. By way of example applications such as Apache, MySql and PHP may all be grouped in a single container for supporting a single service, such as a web based human resource or customer management type of service.

In some embodiments of the invention, the method involves exporting one or more of the respective secure application containers to a remote computer system.

In an embodiment of the invention, each respective secure application-container has data files for the at least one application within the respective secure application container.

The method involves making the data files within one of the respective secure application containers inaccessible to any respective applications within another one of the secure application containers.

In embodiments of the invention, all applications within a given container have their own common root file system exclusive thereto and unique from other containers and from the operating system's root file system.

In embodiments of the invention, a group of applications within a single container share a same IP address, unique to that container.

Hence in some embodiments of the invention a method is provided for associating a respective IP address for a group of applications within a container.

In some embodiments of the invention, for each container of applications, the method involves associating resource limits for all applications within the container.

In some embodiments of the invention, for each group of the plurality of groups of applications, for example, for each container of applications and system files, the method involves associating resource limits comprising any one or more of limits on memory, CPU bandwidth, Disk size and bandwidth and Network bandwidth for the at least one respective application associated with the group.

For each secure application container, the method involves determining whether resources available on the computer system can accommodate the respective resource allocation associated with the group of applications comprising the secure application container.

By way of example, when a container is to be placed on a platform comprising a computer and an operating system (OS) a check is done to ensure that the computer can accommodate the resources required for the container. Care is exercised to ensure that the installation of a container will not overload the computer. For example, if the computer is using 80% of its CPU capacity and installing the container will increase its capacity past 90% the container is not installed.

If the computer system can accommodate the respective resource allocation associated with the group of applications forming the secure application container, the secure application container is exported to the computer system.

Furthermore, in another embodiment, if one or more secure application containers are already installed on the computer system, the method involves determining whether the computer system has enough resources available to accommodate the one or more secure application containers are already installed in addition to the secure application container being exported.

If there are enough resources available, the resources are distributed between the one or more secure application containers and the secure application container being exported to provide resource control.

In some embodiments of the invention, in determining whether resources available on the computer system to accommodate the respective resource allocation, the method involves verifying whether the computer system supports any specific hardware required by the secure application container to be exported. Therefore, a determination can be made as whether any specific hardware requirements of the applications within a container are supported by the server into which the container is being installed. This is somewhat similar to the abovementioned step wherein a determination is made as to whether the server has sufficient resources to support a container to be installed.

In some embodiments of the invention, for each group of applications within a container, in associating a respective resource allocation with the group, the method involves associating resource limits for the at least one respective application associated with the group.

More particularly, the method also involves, during execution on the computer system:

monitoring resource usage of the at least one respective application associated with the group;

intercepting system calls to kernel mode, made by the at least one respective application associated with the group of applications from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application associated with the group with the resource limits;

and forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

The above steps define that the resource limit checks are done by means of a system call intercept, which is, by definition, a hardware mechanism where an application in user mode transitions to kernel code executing in a protected mode, i.e. kernel mode. Therefore, the invention provides a mechanism whereby certain, but not all system calls are intercepted; and wherein operations specific to the needs of a particular container are performed, for example, checks determines if limits may have been exceeded, and then allows the original system to proceed.

Furthermore, in some instances a modification may be made to what is returned to the application; for example, when a system call is made to retrieve the IP address or hostname. The system in accordance with an embodiment of this invention may choose to return a container specific value as opposed to the value that would have otherwise been normally returned by the kernel. This intervention is termed "spoofing" and will be explained in greater detail within the disclosure.

BRIEF DESCRIPTION OF THE DRAWINGS

Exemplary embodiments may be envisaged without departing from the spirit and scope of the invention.

FIG. **1** is a schematic diagram illustrating a containerized system in accordance with an embodiment of this invention.

FIG. **2** is a schematic diagram illustrating a system wherein secure containers of applications and system files are installed on a server in accordance with an embodiment of the invention.

FIG. **3** is a pictorial diagram illustrating the creation of a container.

FIG. **4** is a diagram illustrating how a container is assigned a unique identity such as IP address, Hostname, and MAC address and introduces the "kernel module" which is used to handle system calls.

FIG. **5** is a diagram similar to FIG. **4**, wherein additional configuration data is provided.

FIG. **6** is a diagram illustrating a system wherein the containers are secure containers.

FIG. **7** is a diagram introducing the sequencing or order in which applications within a container are executed.

FIG. **8** is a diagram illustrating the relationship between the storage of container system files and container configuration data for a plurality of secure containers which may be run on a particular computer platform.

FIG. **9** is a diagram that illustrates the installation of a container on a server.

FIG. **10** is a diagram that illustrates the monitoring of a number of applications and state information.

FIG. **11** is a diagram which shows that the container creation process includes the steps to install the container and validate that applications associated with the container are functioning properly.

FIG. **12** is a schematic diagram showing the operation of the invention, according to an embodiment of the invention.

FIG. **13** is a schematic diagram showing software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention.

FIG. **14** is a screen snapshot of an implementation according to the schematic of FIG. **13**.

FIG. **15** is a flow chart of a method of initializing icons of FIG. **14**.

FIG. **16** is a flow chart of a method of installing a software application on a computing platform in response to a drag and drop operation of FIG. **14**; and,

FIG. **17** is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. **13**.

DETAILED DESCRIPTION

Turning now to FIG. **1**, a system is shown having a plurality of servers **10***a*, **10***b*. Each server includes a processor and an independent operating system. Each operating system includes a kernel **12**, hardware **13** in the form of a processor and a set of associated system files, not shown. Each server with an operating system installed is capable of executing a number of application programs. Unlike typical systems, containerized applications are shown on each server having application programs **11***a*, **11***b* and related system files **11***c*.

These system files are used in place of system files such as library and configuration files present typically used in conjunction with the execution of applications.

FIG. **2** illustrates a system in accordance with the invention in which multiple applications **21***a*, **21***b*, **21***c*, **21***d*, **21***e*, and **21***f* can execute in a secure environment on a single server. This is made possible by associating applications with secure containers **20***a*, **20***b* and **20***c*. Applications are segregated and execute with their own copies of files and with a unique identity. In this manner multiple applications may contend for common resources and utilize different versions of system files. Moreover, applications executing within a secure container can co-exist with applications that are not associated with a container, but which are associated with the underlying operating system.

Containers are created through the aggregation of application specific files. A container contains application and data files for applications that provide a specific service. Examples of specific services include but are not limited to CRM (Customer Relation Management) tools, Accounting, and Inventory.

The Secure application containers **20***a*, **20***b* and **20***c* are shown in FIG. **2** in which each combines un-installed application files on a storage medium or network, application files installed on a computer, and system files. The container is an aggregate of all files required to successfully execute a set of software applications on a computing platform. In embodiments of the invention, containers are created by combining application files with system files.

The containers are output to a file storage medium and installed on the computing platforms from the file storage medium. Each container contains application and data files for applications that together provide a specific service. The containers are created using, for example, Linux, Windows and Unix systems and preferably programmed in C and C++; however, the invention is not limited to these operating systems and programming languages and other operating systems and programming language may be used. An application set is a set of one or more software applications that support a specific service. As shown in the figures, which follow, application files are combined with system files to create a composite or container of the files required to support a fully functional software application.

Referring now to FIG. **3** the creation of a container is illustrated from the files used by a specific application and user defined configuration information. The required files can be captured from an existing computer platform **32** where an application of interest is currently installed or the files can be extracted from install media or package files **30**. Two methods of container creation will be described hereafter.

FIG. **4** illustrates a system having two secure containers **20***a* **20***b*, each provided with a unique identity. This allows, for example, other applications to locate a containerized service in a consistent manner independent of which computer platform the containerized service is located on. Container configuration data, collected from a user or user-defined program, is passed to services resident on a computer platform **40** hosting containers. One embodiment shows these services implemented in a kernel module **43**. The configuration information is transferred one time when the container is installed on a platform. The type of information required in order to provide an application associated with a container a unique identity includes items such as an IP address, MAC address and hostname. As applications execute within a container **20***a*, **20***b* environment system calls are intercepted, by the kernel module **43** passing through services installed as part of this invention, before being passed on to the kernel. This system call intercept mechanism allows applications to be provided with values that are container specific rather than values that would otherwise be returned from the kernel by "spoofing" the system.

As introduced heretofore, "spoofing" is invoked when a system call is made. Instead of returning values ordinarily provided by the operating system's kernel a module in accordance with this invention intervenes and returns container specific values to the application making the system call. By way of example, when an application makes the 'uname' system call, the kernel returns values for hostname, OS version, date, time and processor type. The software module in accordance with this invention intercepts the system call and causes the application to see kernel defined values for date, time and processor type; however, the hostname value is purposely changed to one that is container specific. Thus, the software has 'spoofed' the 'uname' system call to return a container specific hostname value rather than the value the kernel would ordinarily return. This same "spoofing" mechanism applies to system calls that return values such as MAC address, etc. A similar procedure exists for network related system calls, such as bind. For other system calls, such as fork and exit (create and delete a new process) the software does not alter what is returned to the application from the kernel; however, the system records that an operation has occurred, which results in the system call intercept of fork not performing any spoofing. The fork call is intercepted for purposes of tracking container-associated activity.

By way of example, if a process within a container creates a new process, by making a fork system call, the new process would be recorded as a part of the container that created it.

System calls are intercepted to perform the following services:

tracking applications, for example a tracking of which applications are in and out of a specific container;

spoofing particular values returned by the kernel;

applying resource checks and imposing resource limits;

monitoring whether an application is executing as expected, retrieving application parameters; and, which applications are being used, by who and for how long; and,

retrieving more complex application information including information related to which files have been used, which files have been written to, which sockets have been used and information related to socket usage, such as the amount of data transferred through a given socket connection.

Container configuration includes the definition of hardware resource usage, as depicted in FIG. **5** including the amount of CPU, memory, network bandwidth and disk usage required to support the applications to be executed in association with the container **20***a* or **20***b*. These values can be used to determine resource requirements for a given compute platform **40**. They can also be used to limit the amount of hardware resources allocated to applications associated with a container. Values for hardware resources can be defined by a user **51** when a container is created. They can be modified after container creation. It is also possible for container runtime services to detect the amount of resources utilized by applications associated with a container. In the automatic detection method values for hardware resources are updated when the container is removed from a compute platform. The user-defined values can be combined with auto-detected values as needed. In this model a user defines values that are then modified by measured values and updated upon container removal.

It can be seen from FIG. **6** that each application executing associated with a container **20***a* or **20***b*, or contained within a

container **20***a* or **20***b*, is able to accesses files that are dedicated to the container. Applications with a container association, that is, applications contained within a container, are not able to access the files provided with the underlying operating system of the compute platform **40** upon which they execute. Moreover, applications with a container **20***a* association are not able to access the files contained in another container **20***b*.

When a container **20***a* or **20***b* is installed specific applications are started, as is shown in FIG. **7** and container configuration information defines how applications **71**, **72** are started. This includes the definition of a first program to start when a container is installed on a compute platform **40**. The default condition, if not customized for a specific container, will start a script that in turn runs other scripts. A script associated with each application is provided in the container file system. The controller script **73**, delineating the first application started in the container, runs each application specific script in turn. This allows for any number of applications to be started with a container association.

Special handling is required to start the first application such that it is associated with a container. Subsequent applications and processes created, as a result of the first application, will be automatically associated with the container. The automatic association is done through tracking mechanisms based on system call intercept. These are contained in a kernel module **43** in one embodiment of the invention. For example, fork, exit and wait system calls are intercepted such that container association can be applied to applications.

The first application started when a container is installed, is associated with the container by informing the system call intercept capabilities, embodied in the kernel module, shown in FIG. **7**, that the current process is part of the container. Then, the application **71** is started by executing the application as part of the current process. In this way, the first application is started in the context of a process that has been associated with the container.

A container has a physical presence when not installed on a compute platform **40**. It is described as residing on a network accessible repository. As is shown in FIG. **8**, a container has a physical presence in two locations **82**, **84**; or stated differently, the files associated with a container have a physical presence in two locations **82**, **84**. The files used by applications associated with a container reside on a network file server **82**. Container configuration is managed by a controller. The configuration state is organized in a database **84** used by the controller. A container then consists of the files used by applications associated with the container and configuration information. Configuration information includes those values which provide a container with a unique identity, for example a unique IP address, MAC address, name, etc., and which describe how a container is installed, starts a first application and shuts down applications when removed.

In a first embodiment all files are exclusive. That is, each container has a separate physical copy of the files required by applications associated with the container. In future embodiments any files that is read-only will be shared between containers.

When a container is installed on a compute platform the files used by applications associated with the container and container specific configuration information elements are combined. FIG. **9** shows that the files **82***a*, **84***a* are either copied, physically placed on storage local to the compute platform **40**, or they are mounted from a network file storage server. When container files are mounted no copy is employed.

Configuration information is used to direct how the container is installed. This includes operations such as describing the first application to be started, mount the container files, if needed, copy files, if needed and create an IP address alias.

Container information is also used to provide the container with a unique identity, such as IP address, MAC address and name. Identity information is passed to the system call intercept service, shown as part of a kernel module **43** in FIG. **9**.

As the software application associated with or contained in a container is executed it is monitored through the use of system call intercept. FIG. **10** shows that a number of operations are monitored. State information relative to application behavior is stored in data structures managed by the system call intercept capabilities. The information gathered in this manner is used to track which processes are associated with a container, their aggregate hardware resource usage and to control specific values returned to the application. Hardware resource usage includes CPU time, memory, file activity and network bandwidth.

FIG. **3** illustrates container creation. The result of container creation is the collection of files required by applications associated with the container and the assembly of container specific configuration information. Container files include both those files provided by an operating system distribution and those files provided by an application install media or package file.

These files can be obtained by using a compute platform upon which the application of interest has been installed. In this case the files are copied from the existing platform. Alternatively, a process where the files from the relative operating system distribution are used as the container files, the container is installed on a compute platform an then application specific files are applied from applicable install media or package file. The result in either case is the collection of all files needed by applications associated with the container onto a network accessible repository.

Container specific configuration information is assembled from user input. The input can be obtained from forms in a user interface, or programmatically through scripting mechanisms.

Two methods of container creation are provided and either of the two can be utilized, depending upon particular circumstances. In a first method of container creation a "skeleton" file system is used. This is a copy of the system files provided with a given operating system (OS) distribution. The skeleton file system is placed on network storage, accessible to other servers. The skeleton file system includes the OS provided system files before an application is installed. A container is created from this skeleton file system. Subsequently, a user logs into the container and installs applications as needed. Most installations use a service called ssh to login to the system which is used to log into a container.

Referring once again to FIG. **3**, applications may come from third party installation media on CDROM, package files **30**, or as downloads from a web site, etc. Once installed in the container the applications become unique to the container in the manner described way that has been described heretofore.

The second method employed to create a container file system uses an existing server, for example a computer or server already installed in a data center. This is also shown in FIG. **3**. The applications of interest may have been installed on an existing server before the introduction of the software and data structures in accordance with this invention. Files are copied from the existing server **32**, including system files and application specific files to network storage. Once the files are copied from the existing server a container is created from those files.

The description of the two methods above differs in that in one instance the container creation begins with the system

files only. The container is created from the system files and then the applications and required files are added and later installed. In the second instance, which is simpler, both system and application files are retrieved, together, from an existing server, and then the container is created. In this manner, the container file system comprises the application and system files.

Once a set of files that are retrieved for creating a container, for example system files only or system and application files, a subset of the system files are modified. The changes made to this subset are quite diverse. Some examples of these changes are:

modifying the contents a file to add a container specific IP address;

modifying the contents of a directory to define start scripts that should be executed;

modifying the contents of a file to define which mount points will relate to a container;

removing certain hardware specific files that are not relevant in the container context, etc., dependent upon requirements.

In some embodiments of the invention, the method involves copying the application specific files, and related data and configuration information to a file storage medium.

This may be achieved with the use of a user interface in which application programs are displayed and selected for copying to a storage medium. In some embodiments of the invention, the application specific files, and related data and configuration information are made available for installation into secure application containers on a remote computer system.

In some embodiments of the invention, the method involves installing at least one of the pluralities of applications in a container's own root file system.

In summary, the invention involves combining and installing at least one application along with system files or a root file system to create a container file system. A container is then created from a container file system and container configuration parameters.

A resource allocation is associated with the secure application container for at least one respective application containerized within the secure application container to allow the at least one respective application containerized within the secure application container to be executed on the computer system without conflict with other applications containerized within other secure application containers.

Thus, a container has an allocation of resources associated with it to allow the application within the container to be executed without contention.

This allocation of resources is made during the container configuration as a step in creating the container. An active check for resource allocation against resources available is performed. Containerized applications may run together within a container; and, non-containerized applications may run outside of a container context on a same computer platform.

Drag and Drop Application Management

Another aspect of this relates to software that manages the operation of a data center.

There has been an unprecedented proliferation of computer systems in the business enterprise sector. This has been a response to an increase in the number and changing nature of software applications supporting key business processes. Management of computer systems required to support these applications has become an expensive proposition. This is partially due to the fact that each of the software applications are in most cases hosted on an independent computing platform or server. The result is an increase in the number of systems to manage.

An aspect of this invention provides a method of installing a software application on a computing platform. The terms computing platform, server and computer are used interchangeably throughout this specification. The method in accordance with this aspect of the invention includes displaying a software application icon representing the software application and a computing platform icon representing the computing platform.

In operation, a selection of the software application for installation is initiated using the software application icon; and a selection of the computing platform to which the software application is to be installed is initiated using the computing platform icon. Installation of the selected software application is then initiated on the selected computing platform.

The computing platform may be a remote computing platform. In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In a preferred embodiment of the invention, the pointing device is a mouse and the selection of the computing platform is received in response to the software application icon being dragged over the computing platform icon and the software application icon being dropped. The installation of the selected software application on the selected computing platform is initiated in response to the software application icon being dropped over the computing platform icon.

Within this specification reference to drag and drop has a conventional meaning of selecting an icon and dragging it across the screen to a target icon; notwithstanding, selecting a first icon and then pointing to a target icon, shall have a same meaning and effect throughout this specification. Relatively moving two icons is meant to mean moving one icon toward another.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform.

In some embodiments, upon initialization, a user account is created for the selected software application.

In some embodiments, upon initialization, files specific to the selected application software are installed on the selected computing platform.

In some embodiments, upon initialization, file access permissions are set to allow a user to access the application.

In some embodiments, upon initialization, a console on the selected computing platform is updated with information indicating that the selected software application is resident on the selected computing platform.

In accordance with another broad aspect, the invention provides a method of de-installing a software application from a computing platform comprising the steps of displaying a software application icon representing the software application and a computing platform icon representing the computing platform on which the software application is installed. A selection of the software application for de-installation using the software application icon is received. A selection of the computing platform from which the software application is to be de-installed using the computing platform icon is also received. De-installation of the selected software application from the selected computing platform is then initiated.

The computing platform may be a remote computing platform.

In some embodiments, the displaying comprises displaying the software application as being installed on the computing platform with the use of the software application icon and the computing platform icon.

In some embodiments, the selection of the software application is received with the use of a graphical user interface in response to the software application icon being pointed to using a pointing device.

In some embodiments, the pointing device is a mouse.

In some embodiments, the selection of the computing platform is in response to the software application icon being dragged away from the computing platform icon and the software application icon being dropped.

In some embodiments, upon initialization, the selected computing platform is tested to determine whether it is a valid computing platform for de-installation of the software application.

In some embodiments, upon initialization, any process associated with the selected software application is stopped.

In some embodiments, upon initialization, data file changes specific to the software application are copied back to a storage medium from which the data file changes originated prior to installation.

In some embodiments, upon initialization, a user account associated with the selected software application is removed.

In some embodiments, upon initialization, a console on the computing platform is updated with information indicating that the selected software application is no longer resident on the selected computing platform.

The invention provides a GUI (Graphical User Interface) adapted to perform any of the above method steps for installation or de-installation.

The invention provides a GUI adapted to display a software application icon representative of a software application available for installation and display a computing platform icon representative of a computing platform.

The GUI is responsive to a selection of the software application icon and the computing platform icon by initiating installation of the software application on the computing platform. The invention provides a GUI adapted to display a software application icon representative of a software application and display a computing platform icon representative of a computing platform, the GUI being responsive to a selection of the software application icon and the computing platform icon by initiating de-installation of the software application from the computing platform.

The GUI is adapted to display a software application icon representative of a software application installed on a computing platform, the GUI being responsive to a selection of the software application icon by initiating de-installation of the software application from the computing platform.

Selection can be made using a drag and drop operation.

The method of de-installation includes displaying a software application icon representing the software application installed on a computing platform. A selection of the software application for de-installation from the computing platform is received using the software application icon. The de-installation of the selected software application from the computing platform is then initiated. In some embodiments, the selection of the application icon is in response to the software application icon being dragged away. In some embodiments, the de-installation of the selected software application from the computing platform is initiated in response to the software application icon being dropped.

Accordingly, the invention relates, in part to methods of installing and removing software applications through a selection such as, for example, a graphical drag and drop operation. In some embodiments of the invention, functions of the GUI support the ability to select an object, move it and initiate an operation when the object is placed on a specific destination. This process has supported operations including the copy and transfer of files. Some embodiments of the invention extend this operation to allow software applications, represented by a graphical icon, to be installed in working order following a drag and drop in a graphical user interface.

The following definitions are used herein:

Storage repository: A centralized disk based storage containing application specific files that make up a software application.

GUI (Graphical User Interface): A computer-based display that presents a graphical interface by which a user interacts with a computing platform by means of icons, windows, menus and a pointing device is referred to as a GUI.

Icon: An icon is an object supported by a GUI. Normally an icon associates an image with an object that can be operated on through a GUI.

Aspects of the invention include:

GUI supporting drag and drop operations
Icons
Storage medium
Console
Network connections
One or more computing platforms

While software applications are represented as graphical icons, they are physically contained in a storage medium. Such a storage medium consists, for example, of disk-based file storage on a server accessible through a network connection. A computing platform is a computer with an installed operating system capable of hosting software applications.

When a software application icon is "dropped" on a computing platform the applications associated with the icon are installed in working order on the selected platform. The computing platform is generally remote with respect to either the platform hosting the graphical interface or the server hosting the storage medium. This topology is not strictly required, but rather a likely scenario.

Referring to FIG. **12**, shown is a schematic showing the operation of an aspect of the invention, according to an embodiment of the invention.

A graphical icon representing a specific software application is displayed through a graphical user interface. Also displayed is an icon representing a remote computing platform upon which a software application can be hosted. Only one computing platform icon is shown; however, it is to be understood that several computing platform icons each representing a respective remote or local computing platform may also be displayed. Similarly, several software application icons may be displayed depending on the number of software applications available. The software application is selected, through the use of a graphical user interface, by pointing to its software application icon and using a mouse click (or other pointing device). The software application icon is dragged over top of the remote computing platform icon and dropped. The drop initiates the operation of installing software applications on the remote computing platform.

Referring to FIG. **13**, shown is a schematic diagram illustrating software application icons collected in a centralized file storage medium and a remote computing platform icon, according to another embodiment of the invention. The software applications icons collected in the file storage medium,

as shown, are graphical icons each representing respective software applications, which are entries in the file storage medium. The computing platform icon represents a computing platform available to host any one or more of the software applications represented by the software icons. When one of the software application icons is selected, dragged, and dropped on a computing platform icon the respective software applications installed on the remote computing platform corresponding to the computing platform icon.

Referring to FIG. **14**, a screen snapshot of an implementation is shown according to the schematic of FIG. **13**. The screen snap shot shows, as an example implementation, software application and computing platform icons. Available software applications corresponding to software application icons ClearCase_Liserver, ClearCase_Registry & Samba2_clone are grouped under the heading "OFFLINE". Computing platforms available to host software applications are featured under the heading "Data Center" and are represented by computing platform icons dell8xx, dell9p, pts2 & pts6.

Operations involve selecting a software application icon, dragging it over a candidate computing platform icon and releasing, or dropping it on the computing platform icon. The selected software application will then be installed in working order on the selected computing platform. The reverse is true for removal of a software application from a selected computing platform. De-installation is accomplished by selecting an application installed on a compute platform and dragging to the repository. The application in question is shown to be associated with a compute platform in graphical fashion. In one embodiment, as shown in FIG. **14**, applications are associated with a compute platform in hierarchical order, or in a tree view. An application connected to a compute platform as root in a tree view is selected and dropped onto the repository. This initiates the de-install operation.

Referring to FIG. **15**, shown is a flow chart of a method of initializing the icons of FIG. **14**. An icon such as a computing platform icon or software application icon is created as a GUI (Graphical User Interface) object. Drop handler operations are then associated to the computing platform icon. In particular, any operation required for installation is associated with the icon.

With reference to FIGS. **12** to **14**, the installation process is initiated by a drag and drop of a software application icon, from a storage medium, to a top of a computing platform icon. An install drop handler implements the installation of the software application. The install drop handler performs several tasks on the destination computing platform, which:

- Tests whether the computing platform is a valid computing platform;
- Creates a user account for the software application;
- Installs application specific files so that they are accessible to the remote computing platform;
- Sets file access permissions such that a new user can access its applications;
- Starts a first application; and
- Updates a console showing that the selected software application is resident on the selected computing platform.

These steps will now be described with reference to FIG. **16** which is a flow chart of a method of installing a software application on a computing platform in response to the drag and drop operation of FIG. **2**. As discussed above, in operation the installation process is invoked when a software application icon is dropped on a computing platform icon.

The method steps of FIG. **16** are performed by an install drop handler. During installation, verification is performed to determine whether the selected computing platform is a valid candidate for installing a selected software application. If the computing platform is a valid candidate, a user account is created for the software application; otherwise, the installation process ends. Once the user account is created, application files associated with the software applications are installed on the selected computing platform so that they are accessible to the computing platform. File access permissions are then set such that one or more new users can access the application being installed. A first application is then started. In some embodiments, an application, for example accounting or payroll represent several programs/processes. In order to start the accounting/payroll service a first application is started that may in turn start other programs/processes. The first program is started. It is then up to the specific service, accounting/payroll, to start any other services it requires.

A console on the selected computing platform on which the software application is being installed is then updated with information to show that the selected software application is resident on the selected computing platform. The console is operational on any desktop computing platform for example, Unix, Linux and Windows.

With reference to FIG. **17**, the removal process is initiated by a drag and drop operation of a software application icon from a computing platform icon to the repository. For this purpose each computing platform icon shows, with the use of associated software application icons (not shown in FIG. **15**), each application software installed on the computing platform.

The de-installation of application software from a selected computing platform is done by a remove drop handler which performs the following tasks on the selected computing platform:

- Tests whether the computing platform is a valid computing platform; Tests are made to verify whether the computing platform is operational. For example, it may have been taken off-line due to a problem or routine maintenance. If so, then applications should not be removed or installed until this state changes.
- Stops processes associated with the software application;
- Copies application specific data file changes from the computing platform back to the storage medium;
- Removes user accounts associated with the software application; and
- Updates the console to show that the selected software application is resident in the repository.

These steps will now be described with reference to FIG. **17** which is a flow chart of a method of de-installing a software application from a computing platform in response to a drag and drop operation of FIG. **13**. The state of the platform is verified to determine whether it is valid. The state includes, for example, on-line, off-line (a problem state) or maintenance (off-line, but for a scheduled task). If the state of the platform is valid and a process or processes associated with a software application selected to be de-installed are stopped; otherwise the de-installation process ends. Once the processes are stopped, application specific data file changes are copied from the computing platform back to the storage medium. This is a process of capturing changes that may have taken place while the application ran and that need to be saved for future instances when the application runs again. For example, payroll may be run every other week. Changes made to the system, accrued amounts, year to date payments, etc. will need to be saved so that when payroll is run the next time these values are updated. Depending on how the operator configures a system, it may be required to copy changes made

when payroll ran back to the repository so that the next time payroll is run these values are installed along with the application.

Any user account associated with the selected software application is removed and a console on the computing platform from which the selected software application is being de-installed is updated with information to show that the selected software application is resident in the repository. Using the method steps of FIGS. **16** and **17**, software applications are therefore installed on a computing platform and removed from the computing platform through a graphical user interface drag and drop operation.

A number of programming languages may be used to implement programs used in embodiments of the invention. Some example programming languages used in embodiments of the invention include, but are not limited to, C++, Java and a number of scripting languages including TCL/TK and PERL.

Installation of software applications is preferably performed on computing platforms that are remote from a console computing platform. However, some embodiments of the invention also have installation of software applications performed on a computing platform that is local to a console computing platform. Numerous modifications and variations of the present invention are possible in light of the above teachings. It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:

storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers; wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems, the containers of application software excluding a kernel, wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server, wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server, and wherein the application software cannot be shared between the plurality of secure containers of application software, and wherein each of the containers has a unique root file system that is different from an operating system's root file system.

2. A method as defined in claim **1**, wherein each container has an execution file associated therewith for starting the one or more applications.

3. A method as defined in claim **2**, wherein the execution file includes instructions related to an order in which executable applications within will be executed.

4. A method as defined in claim **1** further comprising the step of pre-identifying applications and system files required for association with the one or more containers prior to said storing step.

5. A method as defined in claim **2**, further comprising the step of modifying at least some of the associated system files in plural containers to provide an association with a container specific identity assigned to a particular container.

6. A method as defined in claim **2**, comprising the step of assigning a unique associated identity to each of a plurality of the containers, wherein the identity includes at least one of IP address, host name, and MAC address.

7. A method as defined in claim **2** further comprising the step of modifying at least some of the system files to define container specific mount points associated with the container.

8. A method as defined in claim **1**, wherein the one or more applications and associated system files are retrieved from a computer system having a plurality of secure containers.

9. A method as defined in claim **2**, wherein server information related to hardware resource usage including at least one of CPU memory, network bandwidth, and disk allocation is associated with at least some of the containers prior to the applications within the containers being executed.

10. A method as defined in claim **2**, wherein in operation when an application residing within a container is executed, said application has no access to system files or applications in other containers or to system files within the operating system during execution thereof.

11. A method as defined in claim **2**, wherein containers include files stored in network file storage, and parameters forming descriptors of containers stored in a separate location.

12. A method as defined in claim **11**, further comprising the step of merging the files stored in network storage with the parameters to affect the step of storing in claim **1**.

13. A method as defined in claim **1** further comprising the step of associating with a plurality of containers a stored history of when processes related to applications within the container are executed for at least one of, tracking statistics, resource allocation, and for monitoring the status of the application.

14. A method as defined in claim **1** comprising the step of creating containers prior to said step of storing containers in memory, wherein containers are created by:

a) running an instance of a service on a server;

b) determining which files are being used; and,

c) copying applications and associated system files to memory without overwriting the associated system files so as to provide a second instance of the applications and associated system files.

15. A method as defined in claim **14** comprising the steps of:

assigning an identity to the containers including at least one of a unique IP address, a unique Mac address and an estimated resource allocation;

installing the container on a server; and,

testing the applications and files within the container.

16. A method as defined in claim **1** comprising the step of creating containers prior to said step of storing containers in memory, wherein a step of creating containers includes:

using a skeleton set of system files as a container starting point and installing applications into that set of files.

17. A method as defined in claim **1** further comprising installing a service on a target server selected from one of the plurality of servers, wherein installing the service includes:

using a graphical user interface, associating a unique icon representing a service with an unique icon representing

a server for hosting applications related to the service and for executing the service, so as to cause the applications to be distributed to, and installed on the target server.

18. A method as defined in claim **17** wherein the target server and the graphical user interface are at remote locations.

19. A method as defined in claim **18**, wherein the graphical user interface is installed on a computing platform, and wherein the computing platform is a different computing platform than the target server.

20. A method as defined in claim **19**, wherein the step of associating includes the step of relatively moving the unique icon representing the service to the unique icon representing a server.

21. A method as defined in claim **20** further comprising starting a distributed software application.

22. A method according to anyone of claims **20** further comprising updating a console on the selected target server with information indicating that the service is resident on the selected target server.

23. A method claim **17**, further comprising, the step of testing to determine if the selected target server is a valid computing platform, prior to causing the applications to be distributed to, and installed on the target server.

24. A method according to claim **17** further comprising creating a user account for the service.

25. A method as defined in claim **17**, further comprising the step of installing files specific to the selected application on the selected server.

26. A method according to claim **17** further comprising the steps of setting file access permissions to allow a user to access the one of the applications to be distributed.

27. A method as defined in claim **1**, further comprising de-installing a service from a server, comprising:

displaying the icon representing the service;

displaying the icon representing the server on which the service is installed;

and utilizing the icon representing the service and the icon representing the server to initiating the de-installation of the selected service from the server on which it was installed.

28. A method according to claim **27** further comprising separating icon representing the service from the icon representing the server.

29. A method according to claim **27** further comprising testing whether the selected server is a valid computing platform for de-installation of the service.

30. A method according to claim **27** further comprising copying data file changes specific to the service back to a storage medium from which the data file changes originated prior to installation.

31. A computing system for performing a plurality of tasks each comprising a plurality of processes comprising:

a system having a plurality of secure containers of associated files accessible to, and for execution on, one or more servers, each container being mutually exclusive of the other, such that read/write files within a container cannot be shared with other containers, each container of files is said to have its own unique identity associated therewith, said identity comprising at least one of an IP address, a host name, and a Mac_address; wherein, the plurality of files within each of the plurality of containers comprise one or more application programs including one or more processes, and associated system files for use in executing the one or more processes wherein the associated system files are files that are copies of files or modified copies of files that remain as part of the operating system, each container having its own execution file associated therewith for starting one or more applications, in operation, each container utilizing a kernel resident on the server and wherein each container exclusively uses a kernel in an underlying operation system in which it is running and is absent its own kernel; and,

a run time module for monitoring system calls from applications associated with one or more containers and for providing control of the one or more applications.

32. A computing system as defined in claim **31**, further comprising a scheduler comprising values related to an allotted time in which processes within a container may utilize predetermined resources.

33. A computing system as defined in claim **32**, wherein the run time module includes an intercepting module associated with the plurality of containers for intercepting system calls from any of the plurality of containers and for providing values alternate to values the kernel would have assigned in response to the system calls, so that the containers can run independently of one another without contention, in a secure manner, the values corresponding to at least one of the IP address, the host name and the Mac_Address.

34. A computing system as defined in claim **31**, wherein the run time module performs:

monitoring resource usage of applications executing; intercepting system calls to kernel mode, made by the at least one respective application within a container, from user mode to kernel mode;

comparing the monitored resource usage of the at least one respective application with the resource limits; and,

forwarding the system calls to a kernel on the basis of the comparison between the monitored resource usage and the resource limits.

* * * * *

Exhibit 2

U.S. Patent No. 7,519,814 vs. Oracle

Accused Instrumentalities: Oracle products and services using secure containerized applications, including without limitation Oracle Cloud Infrastructure ("OCI") and Oracle Kubernetes Engine ("OCE"), and all versions and variations thereof since the issuance of the asserted patent.

Claim 1

Claim 1	Accused Instrumentalities
[1pre] 1. In a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, the method comprising:	To the extent the preamble is limiting, Oracle and/or its customer practices, through the Accused Instrumentalities, in a system having a plurality of servers with operating systems that differ, operating in disparate computing environments, wherein each server includes a processor and an operating system including a kernel a set of associated local system files compatible with the processor, a method of providing at least some of the servers in the system with secure, executable, applications related to a service, wherein the applications are executed in a secure environment, wherein the applications each include an object executable by at least some of the different operating systems for performing a task related to the service, as claimed. For example, Oracle Kubernetes Engine runs on individual servers, each of which runs an independent operating system running either on bare metal, through an on-premises virtualized infrastructure, through one or more cloud services, or through any other supported deployment. In an exemplary deployment, two or more servers use different operating systems. The servers operate in disparate computing environments, including because each server is a stand-alone computer and/or each server is unrelated to the other servers due to having independent hardware and, in some instances, independent software. Oracle requires and/or provides that each server includes a processor with one or more cores available to the OS kernel. Oracle further requires and/or provides that each server has a supported operating system, which includes a kernel and associated local system files, including for example libraries such as libc/glibc, configuration files, etc. In the infringing system, at least two servers have different operating systems. In at least some instances, Oracle directly owns, operates, controls, and/or benefits from the claimed system and/or method. In other instances, Oracle's customer makes and uses the system and/or method either by following Oracle's direction and control, including Oracle's documentation, or automatically through the ordinary and expected operation of Oracle's software, or a combination thereof.

Claim 1	Accused Instrumentalities
	See claim limitations below. *See also, e.g.*: Why Choose OKE? **Price-Performance** OKE is the lowest cost Kubernetes service amongst all hyperscalers, especially for serverless. **Autoscaling** OKE automatically adjusts compute resources based on demand, which can reduce your costs. **Efficiency** GPUs can be scarce, but OKE job scheduling makes it easy to maximize resource utilization. **Portability** OKE is consistent across clouds and on-premises, enabling portability and avoiding vendor lock-in. **Simplicity** OKE reduces the time and cost needed to manage the complexities of Kubernetes infrastructure. **Reliability** Automatic upgrades and security patching boost reliability for the control plane and worker nodes. https://www.oracle.com/cloud/cloud-native/kubernetes-engine/

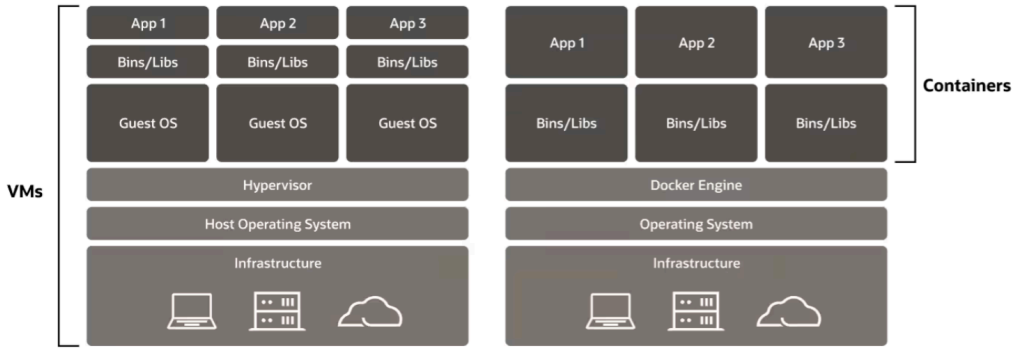
Claim 1	Accused Instrumentalities
	Welcome to Oracle Cloud Infrastructure Oracle Cloud Infrastructure (OCI) is a set of complementary cloud services that enable you to build and run a range of applications and services in a highly available hosted environment. OCI provides high-performance compute capabilities (as physical hardware instances) and storage capacity in a flexible overlay virtual network that is securely accessible from your on-premises network. https://docs.oracle.com/en-us/iaas/Content/GSG/Concepts/baremetalintro.htm **Existing applications can benefit by migrating to OCI and OKE** OKE offers lower total cost of ownership and improved time to market. OKE simplifies operations at scale in the following ways: - Lift and shift; there's no need to rearchitect - Reduce operations burden with automation - Save time on infrastructure management - Increase resource utilization and efficiency - Improve agility, flexibility, uptime, and resilience - Reduce compliance risk and enhance security https://www.oracle.com/cloud/cloud-native/kubernetes-engine/#app-migration

Claim 1	Accused Instrumentalities
	What is OCI Kubernetes Engine (OKE)? Oracle Cloud Infrastructure Kubernetes Engine (OKE) is a managed Kubernetes service that simplifies the development, deployment, and operation of containerized workloads at scale. OKE enables you to quickly create, manage, and consume Kubernetes clusters that leverage underlying OCI compute, networking, and storage services. **When should I use OKE?** You should use OKE when you want to leverage Kubernetes to deploy and manage your Kubernetes-based container applications. It allows you to combine the production-grade container orchestration of standard upstream Kubernetes with the control, security, and high, predictable performance of OCI. https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/ **How does OKE provide resiliency?** When you create an OKE cluster, OKE automatically creates and manages multiple Kubernetes control plane nodes spread across fault domains and availability domains (logical data centers). This is done to help ensure that the managed Kubernetes control plane is highly available. Control plane operations, such as upgrading to newer versions of Kubernetes, can be performed without service interruptions. Additionally, when you provision worker nodes, you can use a placement configuration to control the fault domain and availability domain where they are created. Nodes will automatically come online with labels, which you can use to schedule your workloads so they are robust and highly available. https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/ **Can I deploy private Kubernetes clusters?** Yes; with OKE, your Kubernetes clusters are integrated in your VCN. Your cluster worker nodes, load balancers, and the Kubernetes API endpoint are part of a private or public subnet of your VCN. Regular VCN routing and firewall rules control access to the Kubernetes API endpoint, making it accessible from a corporate network only, through a bastion host, or by specific platform services. https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/

Claim 1	Accused Instrumentalities
	When should I use virtual nodes, managed nodes, or self-managed nodes? • **Virtual nodes** Virtual nodes offer a serverless Kubernetes experience. This option is ideal if you'd rather focus on your application and avoid managing the underlying infrastructure. Virtual nodes relieve you of management-related tasks such as scaling, upgrading, patching, troubleshooting, and provisioning worker nodes. • **Managed nodes** Managed nodes are a good choice for general purpose workloads. They offer an extensive list of customizable configuration options that have been tested by the OKE service. Unlike fully managed virtual nodes, you share the management of worker nodes with OCI. OKE simplifies the management process through features such as on-demand cycling to automate worker node updates, cluster self-healing upon failure detection, autoscaling, and more. • **Self-managed nodes** Self-managed nodes offer access to the underlying infrastructure, configuration options, and compute shapes that aren't currently available to managed nodes. This includes access to specialized infrastructure, such as RDMA-enabled bare metal cluster networks or confidential compute shapes. This advanced control makes self-managed nodes ideal for specialized use cases that aren't supported with managed nodes. Note that with self-managed nodes, you are fully responsible for managing the worker nodes—without the automated features provided by managed or virtual nodes. https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/ **What are the storage options for virtual nodes?** OKE virtual nodes do not yet have persistent storage capabilities. However, there are plans to introduce support for attaching persistent volumes backed by OCI Block Storage and OCI File Storage. If your Kubernetes application requires persistent storage, it's advisable to use OKE managed nodes. https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/

Claim 1	Accused Instrumentalities
	Supported Images for Managed Nodes Kubernetes Engine supports the provisioning of worker nodes (managed nodes only) using some, but not all, of the latest Oracle Linux images provided by Oracle Cloud Infrastructure. You can use these Oracle Linux images when provisioning managed nodes: • OKE Images • Platform Images • Custom Images https://docs.oracle.com/en-us/iaas/Content/ContEng/Reference/contengimagesshapes.htm **What is Docker?** A Docker container is a packaging format that packages all the code and dependencies of an application in a standard format that allows it to run quickly and reliably across computing environments. A Docker container is a popular lightweight, standalone, executable container that includes everything needed to run an application, including libraries, system tools, code, and runtime. Docker is also a software platform that allows developers to build, test, and deploy containerized applications quickly. Containers as a Service (CaaS) or Container Services are managed cloud services that manage the lifecycle of containers. Container services help orchestrate (start, stop, scale) the runtime of containers. Using container services, you can simplify, automate, and accelerate your application development and deployment lifecycle. Docker and Container Services have seen rapid adoption and have been a tremendous success over the last several years. From an almost unknown and rather technical open source technology in 2013, Docker has evolved into a standardized runtime environment now officially supported for many Oracle enterprise products. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Container: Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the "user space." Containers share the host system's kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images. **Docker engine:** The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu. **Docker images:** Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image. **Docker Registry:** Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/ **Docker versus Kubernetes** Linux containers have existed since 2008, but they were not well known until the emergence of Docker containers in 2013. With the onset of Docker containers, came the explosion of interest in developing and deploying containerized applications. As the number of containerized applications grew to span hundreds of containers deployed across multiple servers, operating them became more complex. How do you coordinate, scale, manage, and schedule hundreds of containers? This is where Kubernetes can help. Kubernetes is an open source orchestration system that allows you to run your Docker containers and workloads. It helps you manage the operating complexities when moving to scale multiple containers deployed across multiple servers. The Kubernetes engine automatically orchestrates the container lifecycle, distributing the application containers across the hosting infrastructure. Kubernetes can quickly scale resources up or down, depending on the demand. It continually provisions, schedules, deletes, and monitors the health of the containers. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

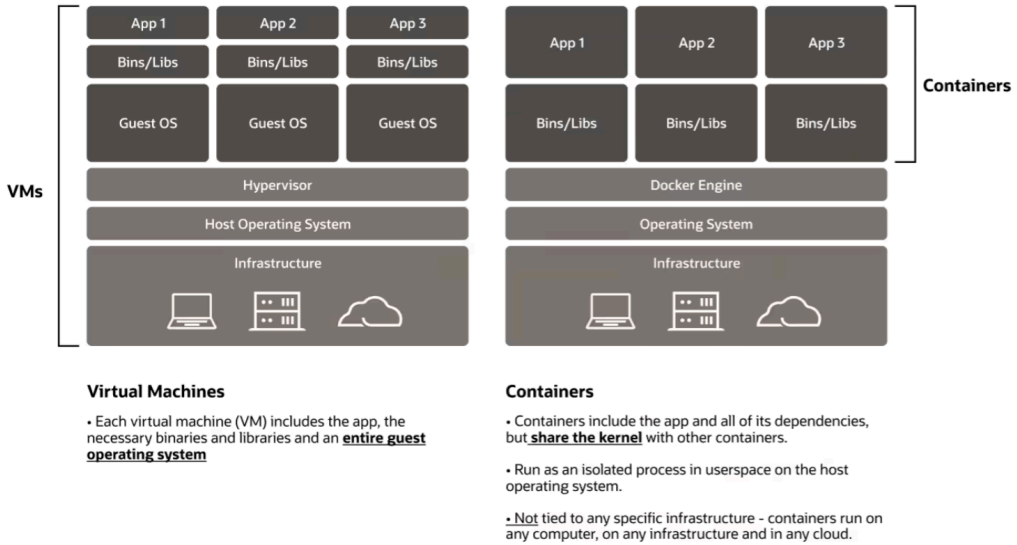
Claim 1	Accused Instrumentalities
	Docker Basics The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more. A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.  **Virtual Machines** • Each virtual machine (VM) includes the app, the necessary binaries and libraries and an **entire guest operating system** **Containers** • Containers include the app and all of its dependencies, but **share the kernel** with other containers. • Run as an isolated process in userspace on the host operating system. • Not tied to any specific infrastructure - containers run on any computer, on any infrastructure and in any cloud. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Container Cloud Services The first part of this article explained some important Docker concepts. However, in a production environment it is not enough to simply run an application in a Docker container. To setup and operate a production environment requires hardware to run the containers. Software such as Docker, along with repositories and cluster managers, must be installed, upgraded and patched. If several Docker containers communicate across hosts, a network must be created. Clustered containers should be restarted if they fail. In addition, a set of containers linked to each other should be deployable as easily as a single logical application instance. An example of this could be a load balancer, a few web servers, some Oracle WebLogic Server instances with an admin server, a managed server, and a database. To manage containerized applications at scale, requires a container orchestration system like Kubernetes or Docker Swarm. Deploying, managing, and operating orchestration systems like Kubernetes can be challenging and time-consuming. To make it easier and more efficient for developers to create containerized applications, cloud providers offer Container Cloud Services or Containers as a Service (CaaS). Container Cloud Services help developers and operations teams streamline and manage the lifecycle of containers in an automated fashion. These orchestration services, typically built using Kubernetes, make it easier for DevOps teams to manage and operate containerized applications at scale. Oracle Cloud Infrastructure Kubernetes Engine and Azure Kubernetes Service are two examples of popular container orchestration managed cloud services. Oracle Cloud Infrastructure Kubernetes Engine is a fully managed, scalable, and highly available service that you can use to deploy your containerized applications in the cloud. Use Kubernetes Engine (sometimes abbreviated to just OKE) when your development team wants to reliably build, deploy, and manage cloud native applications. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/ Kernel mode refers to the processor mode that enables software to have full and unrestricted access to the system and its resources. The OS kernel and kernel drivers, such as the file system driver, are loaded into protected memory space and operate in this highly privileged kernel mode. https://www.techtarget.com/searchdatacenter/definition/kernel

Claim 1	Accused Instrumentalities
[1a] storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers;	The method practiced by Oracle and/or its customer through the Accused Instrumentalities includes a step of storing in memory accessible to at least some of the servers a plurality of secure containers of application software, each container comprising one or more of the executable applications and a set of associated system files required to execute the one or more applications, for use with a local kernel residing permanently on one of the servers. For example, OCI and/or OKE stores application containers, sometimes called Docker containers, container images, Kubernetes containers, or Kubernetes pods, in persistent storage available to each node running the application. The terms "node" and "host" are both used to refer to the claimed server. The container might be in a format defined by the Open Container Initiative. This storage may be physically attached to the server or connected through any supported interconnect, including over a network. In addition to the application software, each container includes associated system files, including a Linux user space required to execute the application, for example libc/glibc and other shared libraries, configuration files, etc. necessary for the application. For example, the container includes a base OS image provided by Oracle or by a third party. The container is compatible with the host kernel, for example because the container libraries are linked against the Linux kernel, and the supported host operating systems also use the Linux kernel, which has a stable binary interface. The containers are secure containers as claimed. For example, the data within an individual container is insulated from the effects of other containers except to the extent the container is specifically configured to allow other containers to modify its data, for example using a shared volume. *See, e.g.*: Overview of File Storage Oracle Cloud Infrastructure File Storage service provides a durable, scalable, secure, enterprise-grade network file system. You can connect to a File Storage service file system from any bare metal, virtual machine, or container instance in your Virtual Cloud Network (VCN). You can also access a file system from outside the VCN using VCN peering, Oracle Cloud Infrastructure FastConnect, and Internet Protocol security (IPSec) virtual private network (VPN). https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	What is Docker? A Docker container is a packaging format that packages all the code and dependencies of an application in a standard format that allows it to run quickly and reliably across computing environments. A Docker container is a popular lightweight, standalone, executable container that includes everything needed to run an application, including libraries, system tools, code, and runtime. Docker is also a software platform that allows developers to build, test, and deploy containerized applications quickly. Containers as a Service (CaaS) or Container Services are managed cloud services that manage the lifecycle of containers. Container services help orchestrate (start, stop, scale) the runtime of containers. Using container services, you can simplify, automate, and accelerate your application development and deployment lifecycle. Docker and Container Services have seen rapid adoption and have been a tremendous success over the last several years. From an almost unknown and rather technical open source technology in 2013, Docker has evolved into a standardized runtime environment now officially supported for many Oracle enterprise products. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

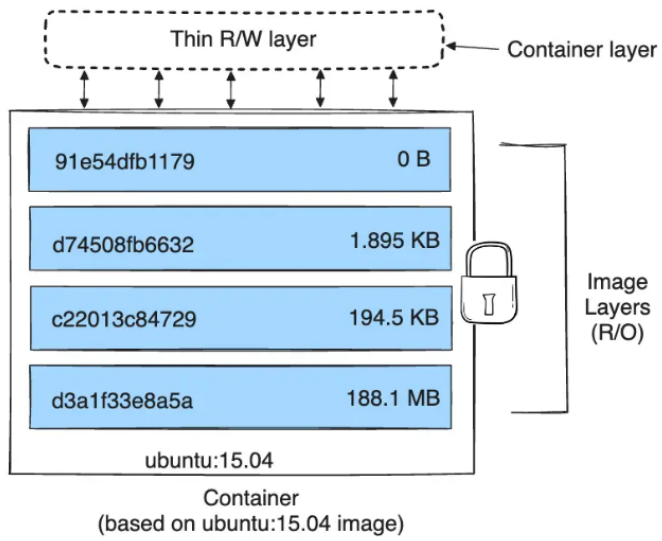
Claim 1	Accused Instrumentalities
	Container: Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the "user space." Containers share the host system's kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images. **Docker engine:** The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu. **Docker images:** Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image. **Docker Registry:** Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/ **Docker versus Kubernetes** Linux containers have existed since 2008, but they were not well known until the emergence of Docker containers in 2013. With the onset of Docker containers, came the explosion of interest in developing and deploying containerized applications. As the number of containerized applications grew to span hundreds of containers deployed across multiple servers, operating them became more complex. How do you coordinate, scale, manage, and schedule hundreds of containers? This is where Kubernetes can help. Kubernetes is an open source orchestration system that allows you to run your Docker containers and workloads. It helps you manage the operating complexities when moving to scale multiple containers deployed across multiple servers. The Kubernetes engine automatically orchestrates the container lifecycle, distributing the application containers across the hosting infrastructure. Kubernetes can quickly scale resources up or down, depending on the demand. It continually provisions, schedules, deletes, and monitors the health of the containers. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Docker Basics The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more. A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.  https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Docker **Images and Containers** An image is a read-only template with instructions for creating a Docker container and an image is based on another image. A container is a standard unit of software that packages up code and all its dependencies. Hence, the application runs quickly and reliably from one environment to another. A Docker Container Image is a lightweight, standalone, executable package of software that includes everything needed to run an application such as code, runtime, system tools, system libraries, and settings. Container images become containers at runtime and for Docker containers, the images become containers when they run on the engine. Containers are available for both Linux and Windows-based applications. The containerized software always runs the same code, regardless of the infrastructure. The container isolates software from its environment and ensures that it works uniformly despite differences for instance between Development, Staging, and Production. **Kubernetes (K8)** Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications. It groups the containers that makes an application into logical units for easy management and discovery. https://docs.oracle.com/en/industries/financial-services/microservices-common/14.6.1.0.0/contg/technologies.html Container images A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings. https://kubernetes.io/docs/concepts/containers/

Claim 1	Accused Instrumentalities
	6. **Do Docker containers package up the entire OS and make it easier to deploy?** Docker containers do not package up the OS. They package up the applications with everything that the application needs to run. The engine is installed on top of the OS running on a host. Containers share the OS kernel allowing a single host to run multiple containers. https://www.docker.com/blog/the-10-most-common-questions-it-admins-ask-about-docker/ **About storage drivers** To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way. **Storage drivers versus Docker volumes** Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer. Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance. https://docs.docker.com/storage/storagedriver/

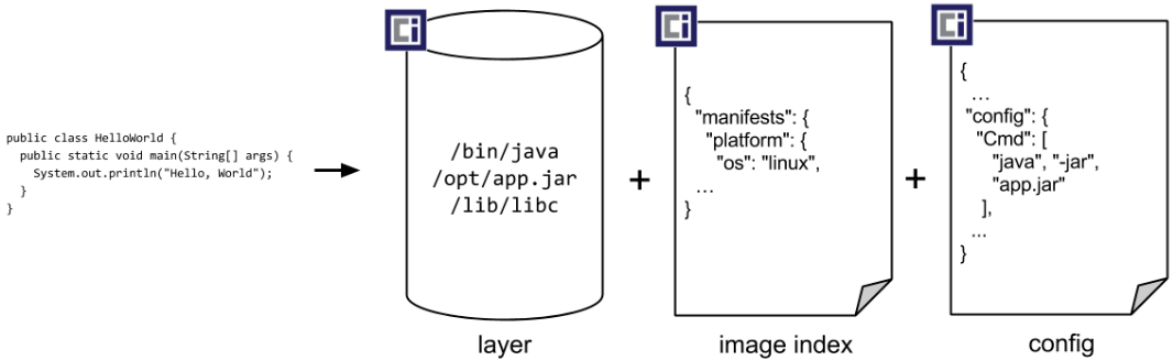
Claim 1	Accused Instrumentalities
	Images and layers A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile: ``` # syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r $HOME/.cache CMD python /app/app.py ``` This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The `FROM` statement starts out by creating a layer from the `ubuntu:22.04` image. The `LABEL` command only modifies the image's metadata, and doesn't produce a new layer. The `COPY` command adds some files from your Docker client's current directory. The first `RUN` command builds your application using the `make` command, and writes the result to a new layer. The second `RUN` command removes a cache directory, and writes the result to a new layer. Finally, the `CMD` instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer. https://docs.docker.com/storage/storagedriver/

Claim 1	Accused Instrumentalities
	Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images. The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.  https://docs.docker.com/storage/storagedriver/

Claim 1	Accused Instrumentalities
	Volumes Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts: https://kubernetes.io/docs/concepts/storage/volumes/ **Container environment** The Kubernetes Container environment provides several important resources to Containers: • A filesystem, which is a combination of an image and one or more volumes. • Information about the Container itself. • Information about other objects in the cluster. https://kubernetes.io/docs/concepts/containers/container-environment/

Claim 1	Accused Instrumentalities
	Images A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment. You typically create a container image of your application and push it to a registry before referring to it in a Pod. https://kubernetes.io/docs/concepts/containers/images/ **Volumes** On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a `Pod` and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes volume abstraction solves both of these problems. Familiarity with Pods is suggested. https://kubernetes.io/docs/concepts/storage/volumes/

Claim 1	Accused Instrumentalities
	# Open Container Initiative ## Image Format Specification This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration. The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md

Claim 1	Accused Instrumentalities
	Overview At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.  https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md

Claim 1	Accused Instrumentalities
	OCI Image Configuration An OCI *Image* is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers. This section defines the `application/vnd.oci.image.config.v1+json` media type. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md

Claim 1	Accused Instrumentalities
	Layer • Image filesystems are composed of *layers*. • Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer. • Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer. • Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. **Image JSON** • Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes. • The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers. • This JSON is considered to be immutable, because changing it would change the computed ImageID. • Changing it means creating a new derived image, instead of changing the existing image. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md

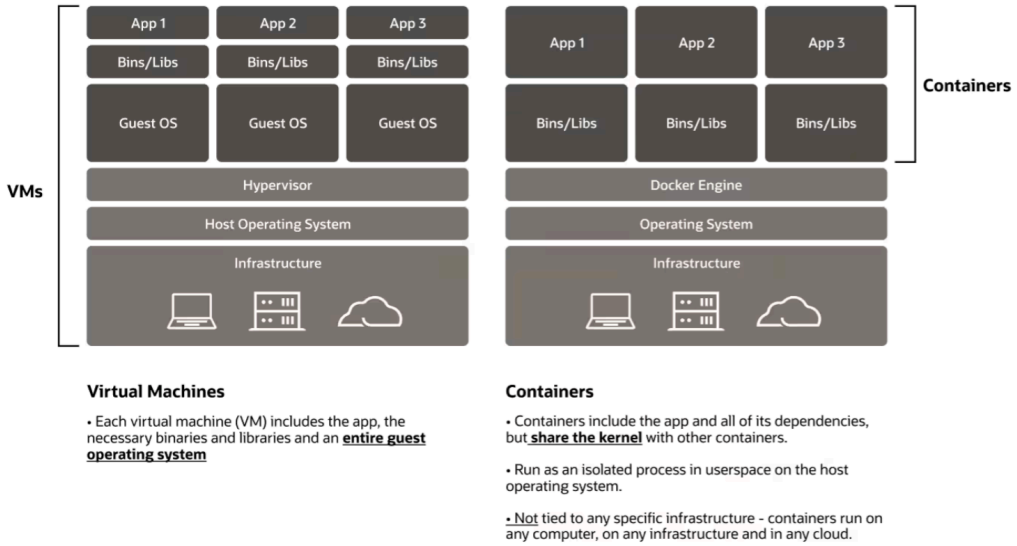
Claim 1	Accused Instrumentalities
	• **rootfs** *object*, REQUIRED The rootfs key references the layer content addresses used by the image. This makes the image config hash depend on the filesystem hash. ○ **type** *string*, REQUIRED MUST be set to `layers`. Implementations MUST generate an error if they encounter a unknown value while verifying or unpacking an image. ○ **diff_ids** *array of strings*, REQUIRED An array of layer content hashes (`DiffIDs`), in order from first to last. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md

Claim 1	Accused Instrumentalities
[1b] wherein the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems,	In the method practiced by Oracle and/or its customer through the Accused Instrumentalities, the set of associated system files are compatible with a local kernel of at least some of the plurality of different operating systems. The associated system files in the container are compatible with the host kernel, for example because they are linked against the Linux kernel and the supported host operating systems also use the Linux kernel, which has a stable binary interface. *See* discussion in element [1a] above. *See also, e.g.*: **Container:** Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the "user space." Containers share the host system's kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
[1c] the containers of application software excluding a kernel,	In the method practiced by Oracle and/or its customer through the Accused Instrumentalities, the containers of application software exclude a kernel. *See, e.g.*: **Container:** Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the "user space." Containers share the host system's kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/ 6. **Do Docker containers package up the entire OS and make it easier to deploy?** Docker containers do not package up the OS. They package up the applications with everything that the application needs to run. The engine is installed on top of the OS running on a host. Containers share the OS kernel allowing a single host to run multiple containers. https://www.docker.com/blog/the-10-most-common-questions-it-admins-ask-about-docker/

Claim 1	Accused Instrumentalities
[1d] wherein some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server,	In the method practiced by Oracle and/or its customer through the Accused Instrumentalities, some or all of the associated system files within a container stored in memory are utilized in place of the associated local system files that remain resident on the server. For example, each container will utilize its own associated system files, including libraries such as libc/glibc and configuration files, not the corresponding associated local system files (*e.g.*, libraries and configuration files of the host OS). As described above and below, in the Accused Instrumentalities the associated system files provide at least some of the same functionalities as the associated local system files. The host/node's associated local system files remain resident on the host/node, for example for use by system processes or applications outside the container environment. *See, e.g.*: **Container:** Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the "user space." Containers share the host system's kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images. **Docker engine:** The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu. **Docker images:** Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image. **Docker Registry:** Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

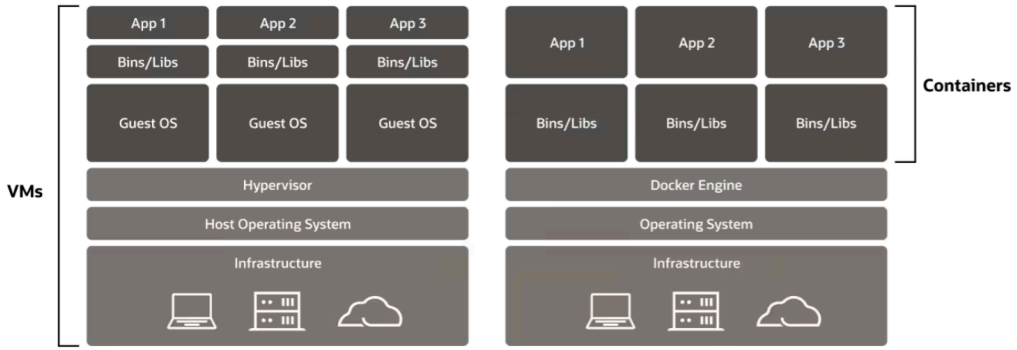
Claim 1	Accused Instrumentalities
	Docker versus Kubernetes Linux containers have existed since 2008, but they were not well known until the emergence of Docker containers in 2013. With the onset of Docker containers, came the explosion of interest in developing and deploying containerized applications. As the number of containerized applications grew to span hundreds of containers deployed across multiple servers, operating them became more complex. How do you coordinate, scale, manage, and schedule hundreds of containers? This is where Kubernetes can help. Kubernetes is an open source orchestration system that allows you to run your Docker containers and workloads. It helps you manage the operating complexities when moving to scale multiple containers deployed across multiple servers. The Kubernetes engine automatically orchestrates the container lifecycle, distributing the application containers across the hosting infrastructure. Kubernetes can quickly scale resources up or down, depending on the demand. It continually provisions, schedules, deletes, and monitors the health of the containers. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Docker Basics The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more. A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.  https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Container Cloud Services The first part of this article explained some important Docker concepts. However, in a production environment it is not enough to simply run an application in a Docker container. To setup and operate a production environment requires hardware to run the containers. Software such as Docker, along with repositories and cluster managers, must be installed, upgraded and patched. If several Docker containers communicate across hosts, a network must be created. Clustered containers should be restarted if they fail. In addition, a set of containers linked to each other should be deployable as easily as a single logical application instance. An example of this could be a load balancer, a few web servers, some Oracle WebLogic Server instances with an admin server, a managed server, and a database. To manage containerized applications at scale, requires a container orchestration system like Kubernetes or Docker Swarm. Deploying, managing, and operating orchestration systems like Kubernetes can be challenging and time-consuming. To make it easier and more efficient for developers to create containerized applications, cloud providers offer Container Cloud Services or Containers as a Service (CaaS). Container Cloud Services help developers and operations teams streamline and manage the lifecycle of containers in an automated fashion. These orchestration services, typically built using Kubernetes, make it easier for DevOps teams to manage and operate containerized applications at scale. Oracle Cloud Infrastructure Kubernetes Engine and Azure Kubernetes Service are two examples of popular container orchestration managed cloud services. Oracle Cloud Infrastructure Kubernetes Engine is a fully managed, scalable, and highly available service that you can use to deploy your containerized applications in the cloud. Use Kubernetes Engine (sometimes abbreviated to just OKE) when your development team wants to reliably build, deploy, and manage cloud native applications. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
[1e] wherein said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server,	In the method practiced by Oracle and/or its customer through the Accused Instrumentalities, said associated system files utilized in place of the associated local system files are copies or modified copies of the associated local system files that remain resident on the server. For example, in some cases the host OS and container will use one or more identical system files, for example when both the host and the container incorporate the same Linux distribution version, or when both host and container use the same version of libc. In the case where the associated system files are identical to the associated local system files, they are copies thereof. In other cases modified copies are used instead, for example when different versions of the same library, or configuration files with different parameters, are used by the host and container. *See, e.g.*: **Container:** Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the "user space." Containers share the host system's kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images. **Docker engine:** The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu. **Docker images:** Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image. **Docker Registry:** Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Docker versus Kubernetes Linux containers have existed since 2008, but they were not well known until the emergence of Docker containers in 2013. With the onset of Docker containers, came the explosion of interest in developing and deploying containerized applications. As the number of containerized applications grew to span hundreds of containers deployed across multiple servers, operating them became more complex. How do you coordinate, scale, manage, and schedule hundreds of containers? This is where Kubernetes can help. Kubernetes is an open source orchestration system that allows you to run your Docker containers and workloads. It helps you manage the operating complexities when moving to scale multiple containers deployed across multiple servers. The Kubernetes engine automatically orchestrates the container lifecycle, distributing the application containers across the hosting infrastructure. Kubernetes can quickly scale resources up or down, depending on the demand. It continually provisions, schedules, deletes, and monitors the health of the containers. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

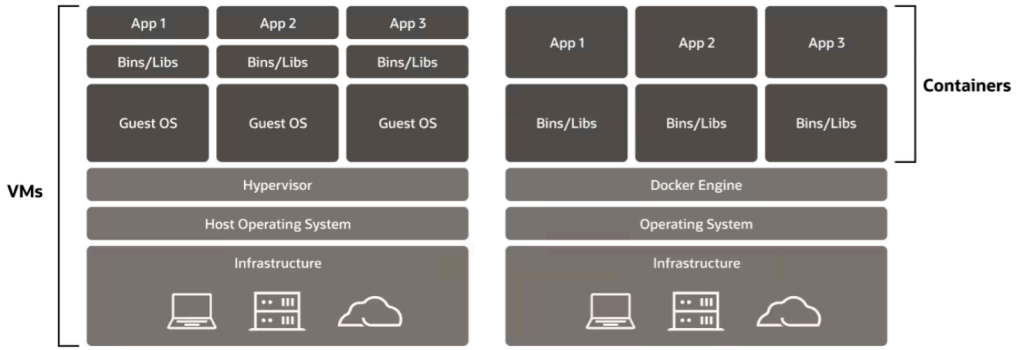
Claim 1	Accused Instrumentalities
	Docker Basics The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more. A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.  **Virtual Machines** • Each virtual machine (VM) includes the app, the necessary binaries and libraries and an **entire guest operating system** **Containers** • Containers include the app and all of its dependencies, but **share the kernel** with other containers. • Run as an isolated process in userspace on the host operating system. • Not tied to any specific infrastructure - containers run on any computer, on any infrastructure and in any cloud. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Container Cloud Services The first part of this article explained some important Docker concepts. However, in a production environment it is not enough to simply run an application in a Docker container. To setup and operate a production environment requires hardware to run the containers. Software such as Docker, along with repositories and cluster managers, must be installed, upgraded and patched. If several Docker containers communicate across hosts, a network must be created. Clustered containers should be restarted if they fail. In addition, a set of containers linked to each other should be deployable as easily as a single logical application instance. An example of this could be a load balancer, a few web servers, some Oracle WebLogic Server instances with an admin server, a managed server, and a database. To manage containerized applications at scale, requires a container orchestration system like Kubernetes or Docker Swarm. Deploying, managing, and operating orchestration systems like Kubernetes can be challenging and time-consuming. To make it easier and more efficient for developers to create containerized applications, cloud providers offer Container Cloud Services or Containers as a Service (CaaS). Container Cloud Services help developers and operations teams streamline and manage the lifecycle of containers in an automated fashion. These orchestration services, typically built using Kubernetes, make it easier for DevOps teams to manage and operate containerized applications at scale. Oracle Cloud Infrastructure Kubernetes Engine and Azure Kubernetes Service are two examples of popular container orchestration managed cloud services. Oracle Cloud Infrastructure Kubernetes Engine is a fully managed, scalable, and highly available service that you can use to deploy your containerized applications in the cloud. Use Kubernetes Engine (sometimes abbreviated to just OKE) when your development team wants to reliably build, deploy, and manage cloud native applications. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/ `COPY` and `ADD`: These commands copy files and directories from your local filesystem into the Docker image. They are often used to include your application code, configuration files, and dependencies. https://medium.com/@swalperen3008/what-is-dockerize-and-dockerize-your-project-a-step-by-step-guide-899c48a34df6

Claim 1	Accused Instrumentalities
	Container images A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings. https://kubernetes.io/docs/concepts/containers/

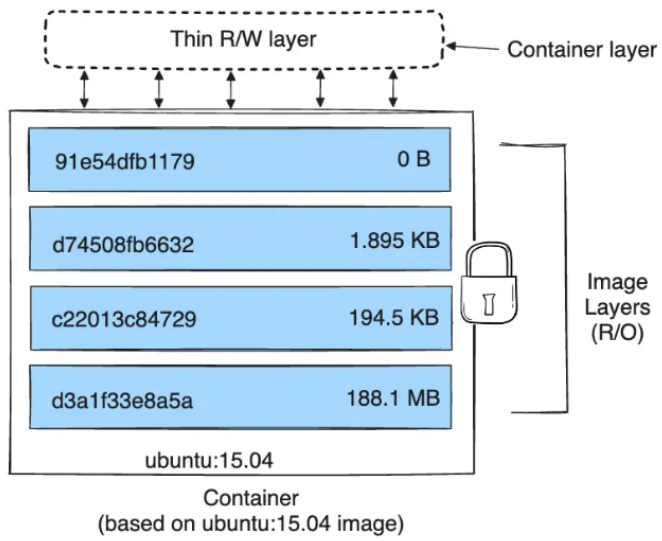
Claim 1	Accused Instrumentalities
[1f] and wherein the application software cannot be shared between the plurality of secure containers of application software,	In the method practiced by Oracle and/or its customer through the Accused Instrumentalities, the application software cannot be shared between the plurality of secure containers of application software. For example, each container has an isolated runtime environment that cannot be accessed by other containers, for example including a per-container writeable layer or other ephemeral per-container storage. For another example, when the plurality of secure containers each corresponds to a different container image, each container cannot access another container's image and therefore application software. *See, e.g.*: **Cgroups and Namespaces History** The underlying Linux kernel features that Docker uses are cgroups and namespaces. In 2008 cgroups were introduced to the Linux kernel based on work previously done by Google developers[1]. Cgroups limit and account for the resource usage of a set of operating system processes. The Linux kernel uses namespace to isolate the system resources of processes from each other. The first namespace, i.e. the mount namespace, was introduced as early as 2002.[2] https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Container: Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the "user space." Containers share the host system's kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images. **Docker engine:** The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu. **Docker images:** Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image. **Docker Registry:** Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/ **Docker versus Kubernetes** Linux containers have existed since 2008, but they were not well known until the emergence of Docker containers in 2013. With the onset of Docker containers, came the explosion of interest in developing and deploying containerized applications. As the number of containerized applications grew to span hundreds of containers deployed across multiple servers, operating them became more complex. How do you coordinate, scale, manage, and schedule hundreds of containers? This is where Kubernetes can help. Kubernetes is an open source orchestration system that allows you to run your Docker containers and workloads. It helps you manage the operating complexities when moving to scale multiple containers deployed across multiple servers. The Kubernetes engine automatically orchestrates the container lifecycle, distributing the application containers across the hosting infrastructure. Kubernetes can quickly scale resources up or down, depending on the demand. It continually provisions, schedules, deletes, and monitors the health of the containers. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Docker Basics The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more. A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.  **Virtual Machines** • Each virtual machine (VM) includes the app, the necessary binaries and libraries and an **entire guest operating system** **Containers** • Containers include the app and all of its dependencies, but **share the kernel** with other containers. • Run as an isolated process in userspace on the host operating system. • Not tied to any specific infrastructure - containers run on any computer, on any infrastructure and in any cloud. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	# About storage drivers To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way. ## Storage drivers versus Docker volumes Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer. Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance. https://docs.docker.com/storage/storagedriver/

Claim 1	Accused Instrumentalities
	Images and layers A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile: ``` # syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r $HOME/.cache CMD python /app/app.py ``` This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The `FROM` statement starts out by creating a layer from the `ubuntu:22.04` image. The `LABEL` command only modifies the image's metadata, and doesn't produce a new layer. The `COPY` command adds some files from your Docker client's current directory. The first `RUN` command builds your application using the `make` command, and writes the result to a new layer. The second `RUN` command removes a cache directory, and writes the result to a new layer. Finally, the `CMD` instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer. https://docs.docker.com/storage/storagedriver/

Claim 1	Accused Instrumentalities
	Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images. The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.  https://docs.docker.com/storage/storagedriver/

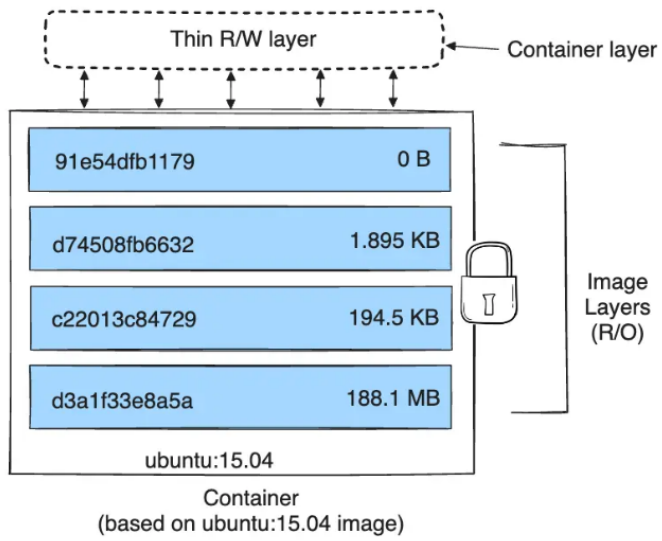
Claim 1	Accused Instrumentalities
[1g] and wherein each of the containers has a unique root file system that is different from an operating system's root file system.	In the method practiced by Oracle and/or its customer through the Accused Instrumentalities, each of the containers has a unique root file system that is different from an operating system's root file system. For example, the container's root file system comprises the image layer(s), an ephemeral writeable layer (e.g., in Docker terminology the container layer), and optionally one or more volumes. This root file system is distinct and isolated from the host operating system's root file system. *See, e.g.*:

Claim 1	Accused Instrumentalities
	# Setting Up Storage for Kubernetes Clusters *Find out how to define and apply persistent volume claims to clusters you've created using Kubernetes Engine (OKE). With Oracle Cloud Infrastructure as the underlying IaaS provider, you can provision persistent volume claims by attaching volumes from the Block Volume service or by mounting file systems from the File Storage service.* Container storage via a container's root file system is ephemeral, and can disappear upon container deletion and creation. To provide a durable location to prevent data from being lost, you can create and use persistent volumes to store data outside of containers. A persistent volume offers persistent storage that enables your data to remain intact, regardless of whether the containers to which the storage is connected are terminated. A persistent volume claim (PVC) is a request for storage, which is met by binding the PVC to a persistent volume (PV). A PVC provides an abstraction layer to the underlying storage. With Oracle Cloud Infrastructure, you can provision persistent volume claims: • By attaching volumes from the Oracle Cloud Infrastructure Block Volume service. The volumes are connected to clusters created by Kubernetes Engine using CSI (Container Storage Interface) or FlexVolume volume plugins deployed on the clusters. Oracle recommends the CSI volume plugin since the upstream Kubernetes project deprecates the FlexVolume volume plugin in Kubernetes version 1.23. See Provisioning PVCs on the Block Volume Service. • By mounting file systems in the Oracle Cloud Infrastructure File Storage service. The File Storage service file systems are mounted inside containers running on clusters created by Kubernetes Engine using a CSI (Container Storage Interface) volume plugin deployed on the clusters. See Provisioning PVCs on the File Storage Service. https://docs.oracle.com/en-us/iaas/Content/ContEng/Tasks/contengcreatingpersistentvolumeclaim.htm

Claim 1	Accused Instrumentalities
	Configuring Docker Storage The Docker Engine is configured to use `overlay2` as the default storage driver to manage Docker containers. This provides a performance and scalability improvement on earlier releases that used the device mapper as the default storage driver, but the technology is new and should be tested properly before use in production environments. For more information on `overlay2`, see: https://docs.docker.com/engine/userguide/storagedriver/overlayfs-driver/ Overlay file systems can corrupt when used in conjunction with any file system that does not have dtype support enabled. https://docs.oracle.com/en/operating-systems/oracle-linux/docker/docker-InstallingOracleContainerRuntimeforDocker.html In Kubernetes, each container can read and write to its own file system. But when a container is restarted, all data is lost. Therefore, containers that need to maintain state would store data in a persistent storage such as Network File System (NFS). What's already stored in NFS isn't deleted when a pod, which might contain one or more containers, is destroyed. Also, an NFS can be accessed from multiple pods at the same time, so an NFS can be used to share data between pods. This behavior is really useful when containers or applications need to read configuration data from a single shared file system or when multiple containers need to read from and write data to a single shared file system. Oracle Cloud Infrastructure File Storage provides a durable, scalable, and distributed enterprise-grade network file system that supports NFS version 3 along with Network Lock Manager (NLM) for a locking mechanism. You can connect to File Storage from any bare metal, virtual machine, or container instance in your virtual cloud network (VCN). You can also access a file system from outside the VCN by using Oracle Cloud Infrastructure FastConnect or an Internet Protocol Security (IPSec) virtual private network (VPN). File Storage is a fully managed service so you don't have to worry about hardware installations and maintenance, capacity planning, software upgrades, security patches, and so on. You can start with a file system that contains only a few kilobytes of data and grows to handle 8 exabytes of data. https://blogs.oracle.com/cloud-infrastructure/post/using-file-storage-service-with-container-engine-for-kubernetes

Claim 1	Accused Instrumentalities
	# About storage drivers To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way. ## Storage drivers versus Docker volumes Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer. Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance. https://docs.docker.com/storage/storagedriver/

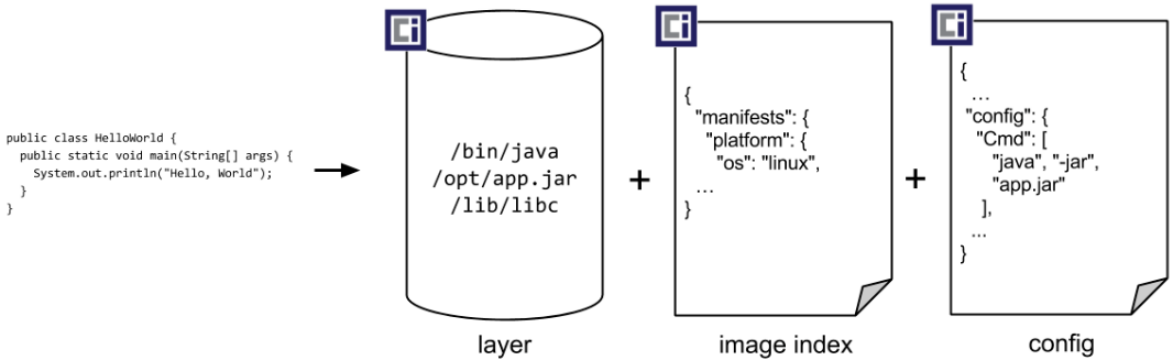
Claim 1	Accused Instrumentalities
	Images and layers A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile: ``` # syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r $HOME/.cache CMD python /app/app.py ``` This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The `FROM` statement starts out by creating a layer from the `ubuntu:22.04` image. The `LABEL` command only modifies the image's metadata, and doesn't produce a new layer. The `COPY` command adds some files from your Docker client's current directory. The first `RUN` command builds your application using the `make` command, and writes the result to a new layer. The second `RUN` command removes a cache directory, and writes the result to a new layer. Finally, the `CMD` instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer. https://docs.docker.com/storage/storagedriver/

Claim 1	Accused Instrumentalities
	Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images. The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.  https://docs.docker.com/storage/storagedriver/

Claim 1	Accused Instrumentalities
	Volumes Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts: https://kubernetes.io/docs/concepts/storage/volumes/ **Container environment** The Kubernetes Container environment provides several important resources to Containers: • A filesystem, which is a combination of an image and one or more volumes. • Information about the Container itself. • Information about other objects in the cluster. https://kubernetes.io/docs/concepts/containers/container-environment/

Claim 1	Accused Instrumentalities
	Images A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment. You typically create a container image of your application and push it to a registry before referring to it in a Pod. https://kubernetes.io/docs/concepts/containers/images/ **Volumes** On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a `Pod` and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes volume abstraction solves both of these problems. Familiarity with Pods is suggested. https://kubernetes.io/docs/concepts/storage/volumes/

Claim 1	Accused Instrumentalities
	# Open Container Initiative ## Image Format Specification This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration. The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md

Claim 1	Accused Instrumentalities
	Overview At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.  https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md

Claim 1	Accused Instrumentalities
	# OCI Image Configuration An OCI *Image* is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers. This section defines the `application/vnd.oci.image.config.v1+json` media type. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md

Claim 1	Accused Instrumentalities
	Layer • Image filesystems are composed of *layers*. • Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer. • Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer. • Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. **Image JSON** • Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes. • The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers. • This JSON is considered to be immutable, because changing it would change the computed ImageID. • Changing it means creating a new derived image, instead of changing the existing image. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md

Claim 1	Accused Instrumentalities
	• **rootfs** *object*, REQUIRED The rootfs key references the layer content addresses used by the image. This makes the image config hash depend on the filesystem hash. ○ **type** *string*, REQUIRED MUST be set to `layers`. Implementations MUST generate an error if they encounter a unknown value while verifying or unpacking an image. ○ **diff_ids** *array of strings*, REQUIRED An array of layer content hashes (`DiffIDs`), in order from first to last. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md

Exhibit 3

US007784058B2

(12) **United States Patent**
Rochette et al.

(10) **Patent No.: US 7,784,058 B2**
(45) **Date of Patent: Aug. 24, 2010**

(54) **COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES**

(75) Inventors: **Donn Rochette**, Fenton, IA (US); **Paul O'Leary**, Kanata (CA); **Dean Huffman**, Kanata (CA)

(73) Assignee: **Trigence Corp.**, Ottawa, Ontario (CA)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1293 days.

(21) Appl. No.: **10/946,536**

(22) Filed: **Sep. 21, 2004**

(65) **Prior Publication Data**

US 2005/0066303 A1 Mar. 24, 2005

Related U.S. Application Data

(60) Provisional application No. 60/504,213, filed on Sep. 22, 2003.

(51) **Int. Cl.**
G06F 9/22 (2006.01)

(52) **U.S. Cl.** **719/310**; 719/319

(58) **Field of Classification Search** 719/310, 719/319

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,481,706	A *	1/1996	Peek	710/200
6,212,574	B1 *	4/2001	O'Rourke et al.	719/321
6,260,075	B1 *	7/2001	Cabrero et al.	719/310
2002/0004854	A1	1/2002	Hartley	
2002/0174215	A1	11/2002	Schaefer	709/224
2003/0101292	A1	5/2003	Fisher	
2004/0025165	A1 *	2/2004	Desoli et al.	719/310

FOREIGN PATENT DOCUMENTS

WO	WO 02/06941	A	1/2002
WO	WO 2006/039181	A	4/2006

OTHER PUBLICATIONS

U.S. Appl. No. 60/512,103, filed Oct. 20, 2003, Rochette et al.

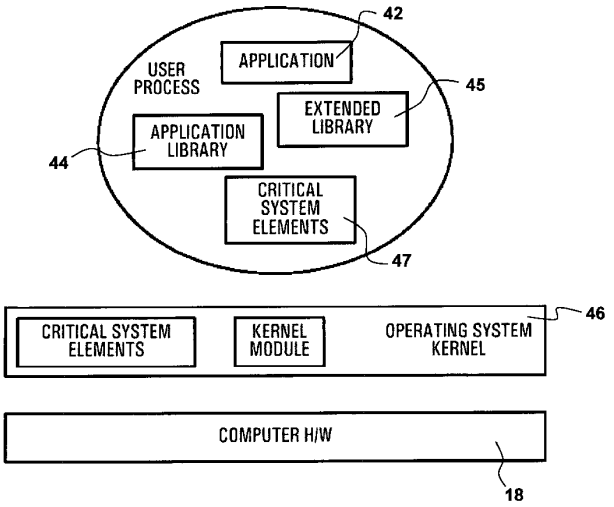
* cited by examiner

Primary Examiner—Hyung S Sough
Assistant Examiner—Syed Roni
(74) *Attorney, Agent, or Firm*—Allen, Dyer, Doppelt, Milbrath & Gilchrist, P.A. Attorneys at Law

(57) **ABSTRACT**

A computing system and architecture is provided that affects and extends services exported through application libraries. The system has an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and, a shared library having critical system elements (SLCSEs) stored within the shared library for use by the software applications in user mode. The SLCSEs stored in the shared library are accessible to the software applications and when accessed by a software application forms a part of the software application. When an instance of an SLCSE provided to an application from the shared library it is ran in a context of the software application without being shared with other software applications. The other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function, and can be run simultaneously.

18 Claims, 7 Drawing Sheets



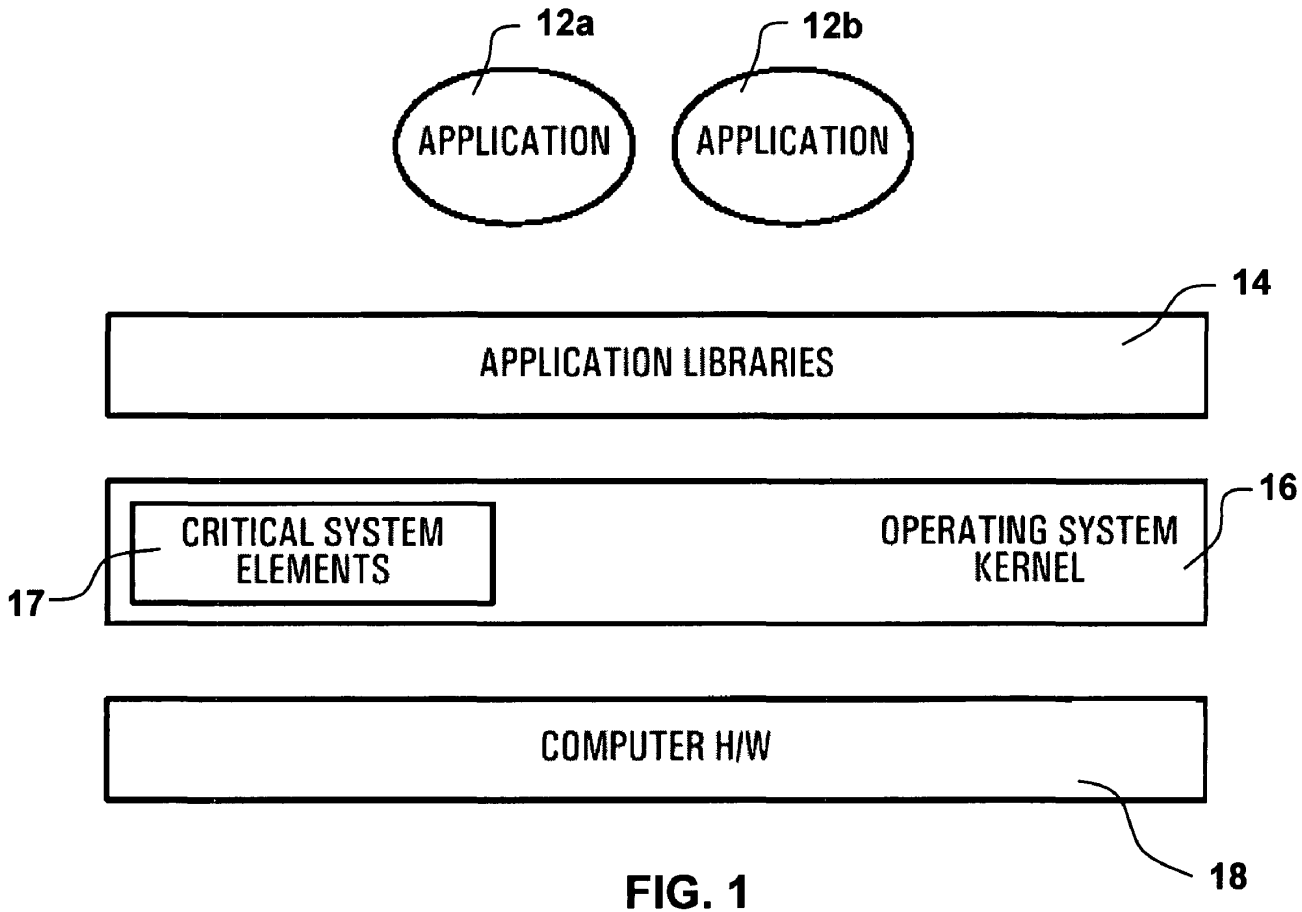


FIG. 1
Prior Art

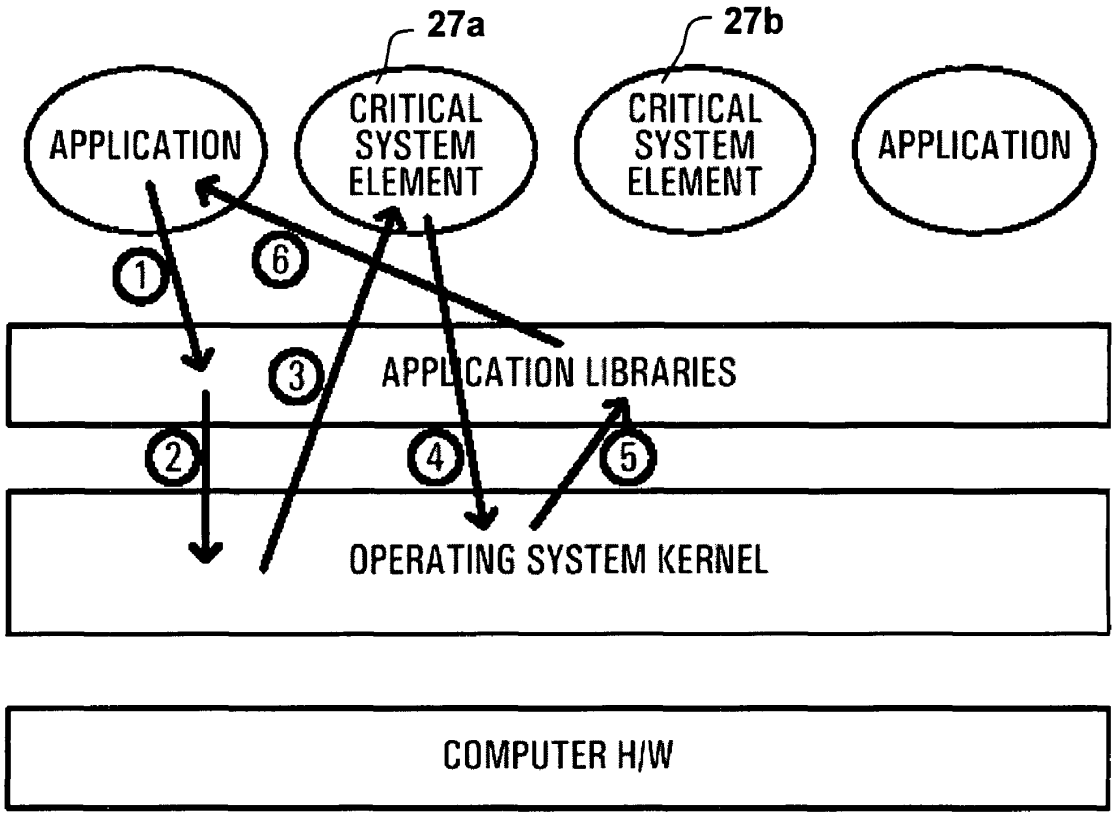


FIG. 2a
Prior Art

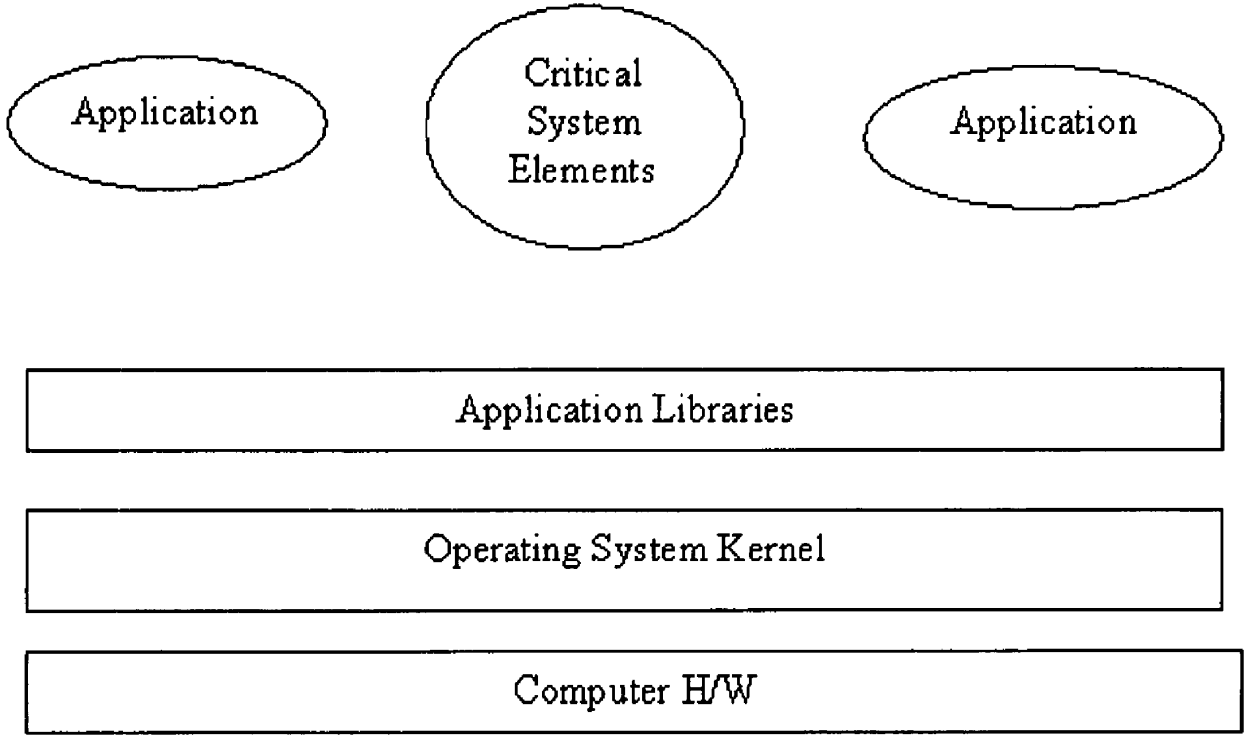


FIG. 2b
Prior Art

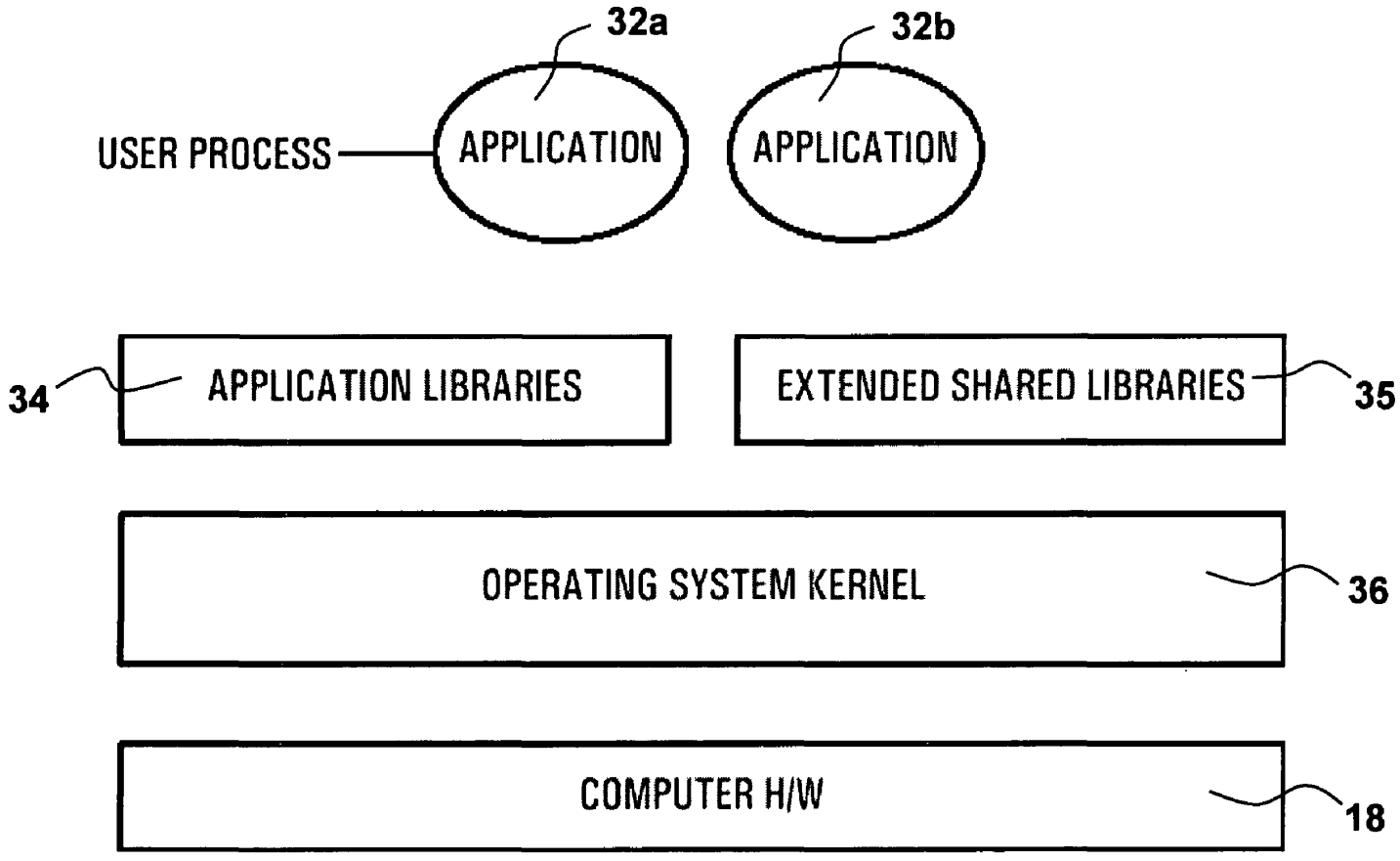

32a
32b
USER PROCESS
APPLICATION
APPLICATION
34
APPLICATION LIBRARIES
EXTENDED SHARED LIBRARIES
35
OPERATING SYSTEM KERNEL
36
COMPUTER H/W
18

FIG. 3

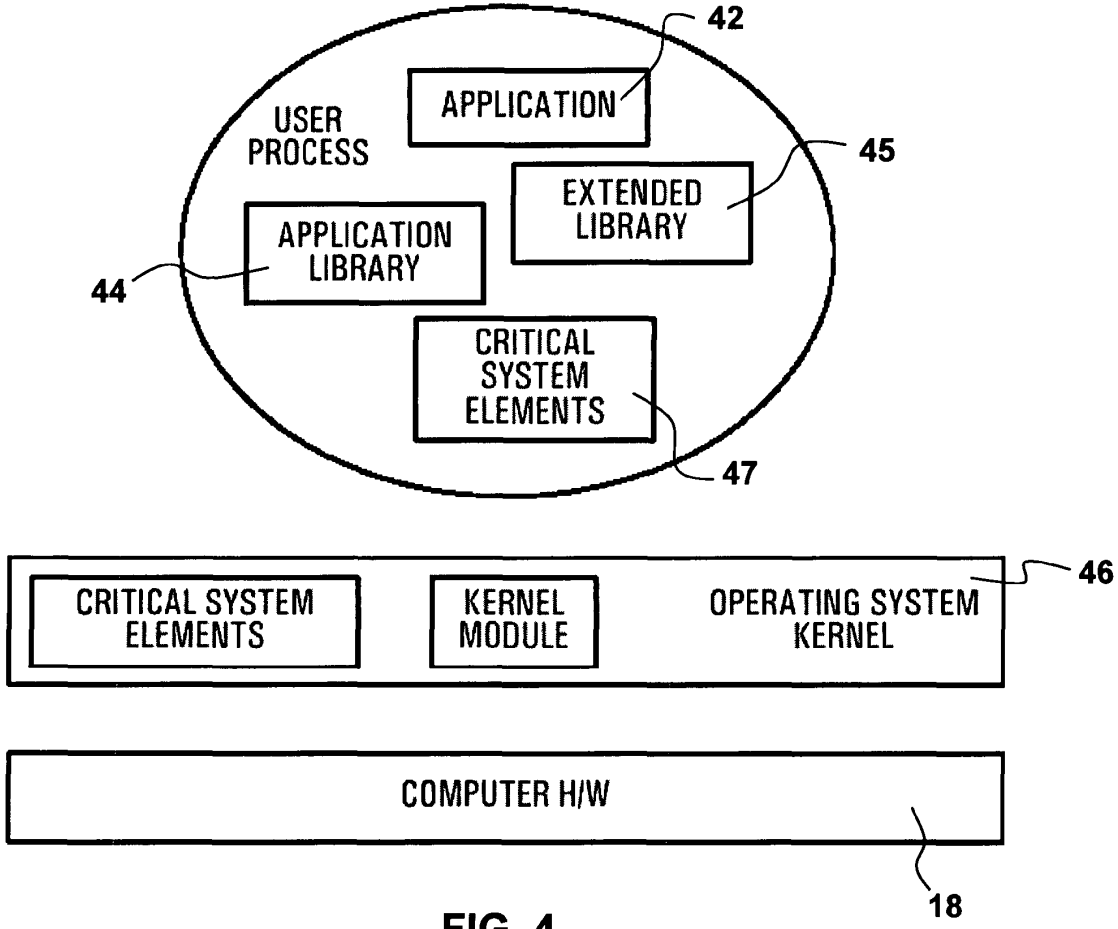

USER PROCESS
APPLICATION
42
EXTENDED LIBRARY
45
APPLICATION LIBRARY
44
CRITICAL SYSTEM ELEMENTS
47
CRITICAL SYSTEM ELEMENTS
KERNEL MODULE
OPERATING SYSTEM KERNEL
46
COMPUTER H/W
18

FIG. 4

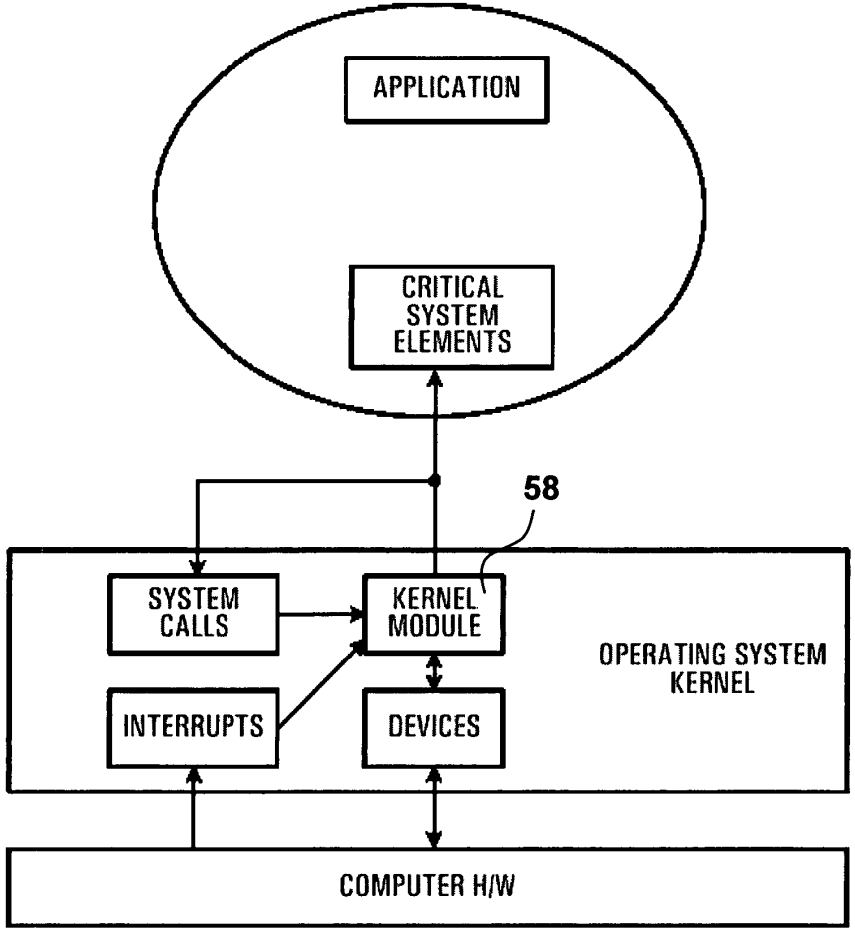

APPLICATION
CRITICAL SYSTEM ELEMENTS
58
SYSTEM CALLS
KERNEL MODULE
OPERATING SYSTEM KERNEL
INTERRUPTS
DEVICES
COMPUTER H/W

FIG. 5

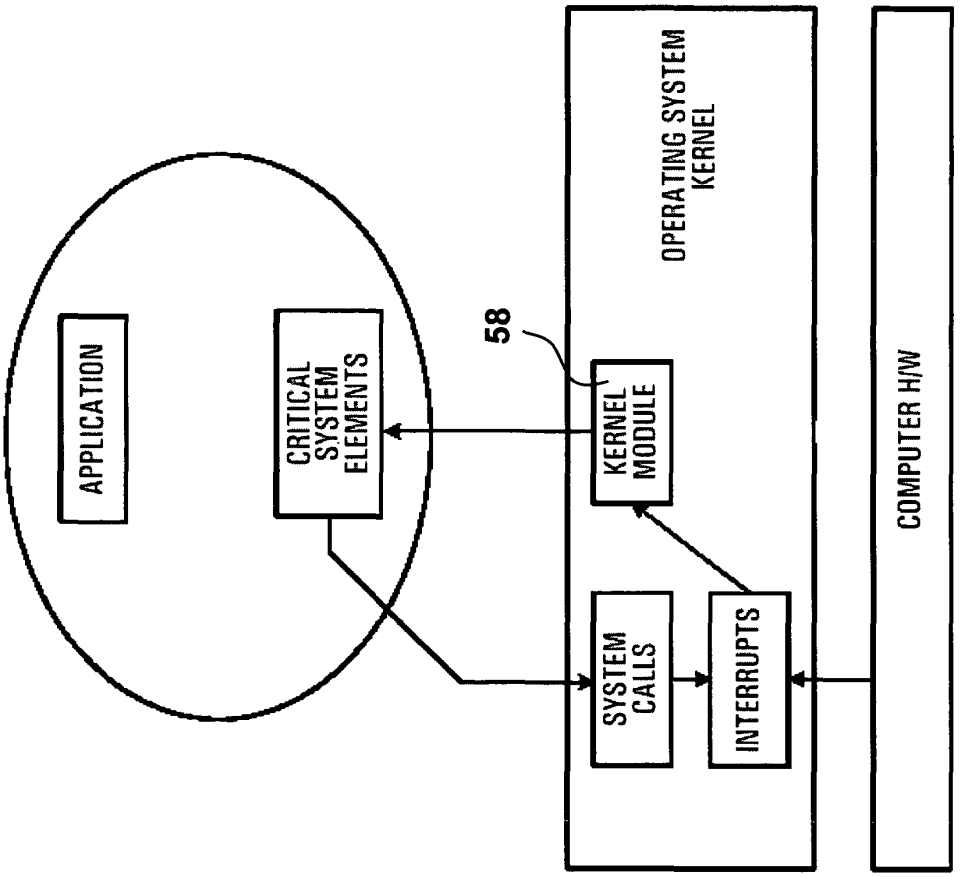

APPLICATION
CRITICAL SYSTEM ELEMENTS
58
KERNEL MODULE
OPERATING SYSTEM KERNEL
SYSTEM CALLS
INTERRUPTS
COMPUTER H/W

FIG. 6

COMPUTING SYSTEM HAVING USER MODE CRITICAL SYSTEM ELEMENTS AS SHARED LIBRARIES

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority of U.S. Provisional Patent Application No. 60/504,213 filed Sep. 22, 2003, entitled "User Mode Critical System Element as Shared Libs", which is incorporated herein by reference.

FIELD OF THE INVENTION

This invention relates to a computing system, and to an architecture that affects and extends services exported through application libraries.

BACKGROUND OF THE INVENTION

Computer systems are designed in such a way that software application programs share common resources. It is traditionally the task of an operating system to provide mechanisms to safely and effectively control access to shared resources. In some instances the centralized control of elements, critical to software applications, hereafter called critical system elements (CSEs) creates a limitation caused by conflicts for shared resources.

For example, two software applications that require the same file, yet each requires a different version of the file will conflict. In the same manner two applications that require independent access to specific network services will conflict. A common solution to these situations is to place software applications that may potentially conflict on separate compute platforms. The current state of the art, defines two architectural approaches to the migration of critical system elements from an operating system into an application context.

In one architectural approach, a single server operating system places critical system elements in the same process. Despite the flexibility offered, the system elements continue to represent a centralized control point.

In the other architectural approach, a multiple server operating system places critical system elements in separate processes. While offering even greater options this architecture has suffered performance and operational differences.

An important distinction between this invention and prior art systems and architectures is the ability to allow a CSE to execute in the same context as an application. This then allows, among other things, an ability to deploy multiple instances of a CSE. In contrast, existing systems and architectures, regardless of where a service is defined to exist, that is, in kernel mode, in user mode as a single process or in user mode as multiple processes, all support the concept of a single shared service.

SUMMARY OF THE INVENTION

In accordance with a first broad aspect of this invention, a computing system is provided that has an operating system kernel having operating system critical system elements (OSCSEs) and adapted to run in kernel mode; and a shared library adapted to store replicas of at least some of the critical system elements, for use by the software applications in user mode executing in the context of the application. The critical system elements are run in a context of a software application.

The term replica used herein is meant to denote a CSE having similar attributes to, but not necessarily and preferably

not an exact copy of a CSE in the operating system (OS); notwithstanding, a CSE for use in user mode, may in a less preferred embodiment be a copy of a CSE in the OS.

In accordance with the invention, a computing system for executing a plurality of software applications is provided, comprising:

an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode wherein some of the CSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications by being linked thereto, and wherein an instance of a CSE provided to an application from the shared library is run in a context of said software application without being shared with other software applications and where some other applications running under the operating system can, have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In accordance with the invention, there is provided, a computing system for executing a plurality of software applications comprising an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode; and,

1. a shared library having critical system elements (SLCSEs) stored therein for use by the software applications in user mode as a service distinct from that supplied in the OS kernel.
2. wherein some of the SLCSEs stored in the shared library are accessible to a plurality of the software applications and form a part of at least some of the software applications, and wherein a unique instance of a CSE provided to an application from the shared library is run in a context of said software application and where some other applications running under the operating system each have use of a unique instance of a corresponding critical system element for performing essentially the same function.

In some embodiments, the computing system has application libraries accessible by the software applications and augmented by the shared library.

Application libraries are libraries of files required by an application in order to run. In accordance with this invention, SLCSEs are placed in shared libraries, thereby becoming application libraries, loaded when the application is loaded. A shared library or dynamic linked library (DLL) refers to an approach, wherein the term application library infers a dependency on a set of these libraries used by an application.

Typically, the critical system elements are not removed from operating system kernel.

In some embodiments, the operating system kernel has a kernel module adapted to serve as an interface between a service in the context of an application program and a device driver.

In some embodiments, the critical system elements in the context of an application program use system calls to access services in the kernel module.

In some embodiments, the kernel module is adapted to provide a notification of an interruption to a service in the context of an application program.

In some embodiments, the interrupt handling capabilities are initialized through a system call.

In some embodiments, the kernel module comprises a handler which is installed for a specific device interrupt.

In some embodiments, the handler is called when an interrupt request is generated by a hardware device.

In some embodiments, the handler notifies the service in the context of an application through the use of an up call mechanism.

In some embodiments, function overlays are used to intercept software application accesses to operating system services.

In some embodiments, the critical system elements stored in the shared library are linked to the software applications as the software applications are loaded.

In some embodiments, the critical system elements rely on kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

In some embodiments, the kernel services are replicated in user mode and contained in the shared library with the critical system elements. In the preferred embodiment the kernel itself is not copied. CSEs are not taken from the kernel and copied to user mode. An instance of a service that is also provided in the kernel is created to be implemented in user mode. By way of example, the kernel may provide a TCP/IP service and in accordance with this invention, a TCP/IP service is provided in user mode as an SLCSE. The TCP/IP service in the kernel remains fully intact. The CSE is not shared as such. Each application that will use a CSE will link to a library containing the CSE independent of any other application. The intent is to provide an application with a unique instance of a CSE.

In accordance with this invention, the code shared is by all applications on the same compute platform. However, this shared code does not imply that the CSE itself is shared. This sharing of code is common practice. All applications currently share code for common services, such services include operations such as such as open a file, write to the file, read from the file, etc. Each application has its own unique data space. This indivisible data space ensures that CSEs are unique to an application or more commonly to a set of applications associated with a container, for example. Despite the fact that all applications may physically execute the same set of instructions in the same physical memory space, that is, shared code space, the instructions cause them to use separate data areas. In this manner CSEs are not shared among applications even though the code is shared.

In some embodiments, the kernel services used by CSEs comprise memory allocation, synchronization and device access.

In some embodiments, the kernel services that are platform specific are not replicated, or provided as SLCSEs.

In some embodiments, the kernel services which are platform specific are called by a critical system element running in user mode. In some embodiments, a user process running under the computing system has a respective one of the software applications, the application libraries, the shared library and the critical system elements all of which are operating in user mode.

In some embodiments, the software applications are provided with respective versions of the critical system elements. In some embodiments, the system elements which are application specific reside in user mode, while the system elements which are platform specific reside in the operating system kernel. In some embodiments, a control code is placed in kernel mode.

In some embodiments, the kernel module is adapted to enable data exchange between the critical system elements in user mode and a device driver in kernel mode.

In some embodiments, the data exchange uses mapping of virtual memory such that data is transferred both from the critical system elements in user mode to the device driver in kernel mode and from the device driver in kernel mode to the critical system elements in user mode.

In some embodiments, the kernel module is adapted to export services for device interface.

In some embodiments, the export services comprise initialization to establish a channel between a critical system element of the critical system elements in user mode and a device.

In some embodiments, the export services comprise transfer of data from a critical system element of the critical system elements in user mode to a device managed by the operating system kernel.

In some embodiments, the export services include transfer of data from a device to a critical system element of the critical system elements in user mode.

According to a second broad aspect, the invention provides an operating system comprising the above computing system.

According to a third broad aspect, the invention provides a computing platform comprising the above operating system and computing hardware capable of running under the operating system.

According to a fourth broad aspect, the invention provides a shared library accessible to software applications in user mode, the shared library being adapted to store system elements which are replicas of systems elements of an operating system kernel and which are critical to the software applications.

According to a fifth broad aspect, the invention provides an operating system kernel having system elements and adapted to run in a kernel mode and to replicate, for storing in a shared library which is accessible by software applications in user mode, system elements which are critical to the software applications.

According to a sixth broad aspect, the invention provides an article of manufacture comprising a computer usable medium having computer readable program code means embodied therein for a computing architecture. The computer readable code means in said article of manufacture has computer readable code means for running an operating system kernel having critical system elements in kernel mode; and computer readable code means for storing in a shared library replicas of at least some of the critical system elements, for use by software applications in user mode.

The critical system elements are run in a context of a software application. Accordingly, elements critical to a software application are migrated from centralized control in an operating system into the same context as the application. Advantageously, the invention allows specific operating system services to efficiently operate in the same context as a software application.

In accordance with this invention, critical system elements normally embodied in an operating system are exported to software applications through shared libraries. The shared library services provided in an operating system are used to expose these additional system elements.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the invention will now be described with reference to the attached drawings in which:

FIG. **1** is an architectural view of the traditional monolithic prior art operating system;

FIG. **2***a* is an architectural view of a multi-server operating system in which some critical system elements are removed from the operating system kernel and are placed in multiple distinct processes or servers;

FIG. **2***b* is illustrative of a known system architecture where critical system elements execute in user mode and execute in distinct context from applications in a single application process context;

FIG. **3** is an architectural view of an embodiment of the invention;

FIG. **4** is a functional view showing how critical system elements exist in the same context as an application;

FIG. **5** is a block diagram showing a kernel module provided by an embodiment of the invention; and,

FIG. **6** shows how interrupt handling occurs in an embodiment of the invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Embodiments of the invention enable the replication of critical system elements normally found in an operating system kernel. These replicated CSEs are then able to run in the context of a software application. Critical system elements are replicated through the use of shared libraries. A CSE is replicated by way of a kernel service being repeated in user space. This replicated CSE may differ slightly from its counterpart in the OS. Replication is achieved placing CSEs similar to those in the OS in shared libraries which provides a means of attaching or linking a CSE service to an application having access to the shared library. Therefore, a service in the kernel is substantially replicated in user mode through the use of shared libraries.

The term replication implies that system elements become separate extensions accessed through shared libraries as opposed to being replaced from an operating system. Hence, CSEs are generally not copies of a portion of the OS; they are an added additional service in the form of a CSE. Shared libraries are used as a mechanism where by an application can utilize a CSE that is part of a library. By way of example; a Linux based platform provides a TCP/IP stack in the Linux kernel. This invention provides a TCP/IP stack in the form of a CSE that is a different implementation of a TCP/IP stack, from Berkeley Software Distribution (BSD) by way of example. Applications on a Linux platform may use a BSD TPC/IP stack in the form of a CSE. The mechanism for attaching the BSD TCP/IP stack to an application is in the form of a shared library. Shared libraries as opposed to being copies from the OS to another space are a distinct implementation of the service (i.e. TCP/IP) packaged in the form of a shared library capable of executing in user mode linked to an application.

In a broad sense, the TCP/IP services in the CSE are the same as those provided in the Linux kernel. The reason two of these service may be required is to allow an application to utilize network services provided by a TCP/IP stack, that have unique configuration or settings for the application. Or the setting used by one application may conflict with the settings needed by another application. If, by way of example, a first application requires that specific network packets using a range of port numbers be passed to itself, it would need to configure route information to allow the TCP/IP stack to process these packets accordingly. On the same platform a second application is then exposed to either undesirable performance issues or security risks by allowing network packets with a broad port number range to be processed by the TCP/IP stack. In another scenario the configuration/settings established to support one application may have an adverse effect on the system as a whole.

By way of introduction, a number of terms will now be defined.

Critical System Element (CSE): Any service or part of a service, "normally" supplied by an operating system, that is critical to the operation of a software application.

A CSE is a dynamic object providing some function that is executing instructions used by applications.

BY WAY OF EXAMPLE CSEs INCLUDE:

Network services including TCP/IP, Bluetooth, ATM; or message passing protocols

File System services that offer extensions to those supplied by the OS

1. Access files that reside in different locations as though they were all in a single locality
2. Access files in a compressed, encrypted or packaged image as though they were in a local directory in a standard format

Implementation of file system optimizations for specific application behavior

Implementation of network optimizations for specific application services such as:

1. Kernel bypass where hardware supported protocol processing is provided
2. Modified protocol processing for custom hardware services

Compute platform: The combination of computer hardware and a single instance of an operating system.

User mode: The context in which applications execute.

Kernel mode: The context in which the kernel portion of an operating system executes. In conventional systems, there is a physical separation enforced by hardware between user mode and kernel mode. Application code cannot run in kernel mode.

Application Programming Interface (API): An API refers to the operating system and programming language specific functions used by applications. These are typically supplied in libraries which applications link with either when the application is created or when the application is loaded by the operating system. The interfaces are described by header files provided with an operating system distribution. In practice, system APIs are used by applications to access operating system services.

Application library: A collection of functions in an archive format that is combined with an application to export system elements.

Shared library: An application library code space shared among all user mode applications. The code space is different than that occupied by the kernel and its associated files. The shared library files are placed in an address space that is accessible to multiple applications.

Static library: An application library whose code space is contained in a single application.

Kernel module: A set of functions that reside and execute in kernel mode as extensions to the operating system kernel. It is common in most systems to include kernel modules which provide extensions to the existing operating system kernel.

Up call mechanism: A means by which a service in kernel mode executes a function in a user mode application context.

FIG. **1** shows a conventional architecture where critical system elements execute in kernel mode. Critical system elements are contained in the operating system kernel. Applications access system elements through application libraries.

In order for an application of FIG. **1** to make use of a critical system element **17** in the kernel **16**, the application **12***a* or **12***b*

makes a call to the application libraries **14**. It is impractical to write applications, which handle CPU specific/operating specific issues directly. As such, what is commonly done is to provide an application library in shared code space, which multiple applications can access. This provides a platform independent interface where applications can access critical system elements. When the application **12***a*, **12***b* makes a call to a critical system element **17** through the application library **14**, a system call may be used to transition from user mode to kernel mode. The application stops running as the hardware **18** enters kernel mode. The processor makes the transition to a protected/privileged mode of execution called kernel mode. Code supplied by the OS in the form of a kernel begins execution when the transition is made. The application code is not used when executing in kernel mode. The operating system kernel then provides the required functionality. It is noted that each oval **12***a*, **12***b* in FIG. **1** represents a different context. There are two application contexts in the illustrated example and the operating system context is not shown as an oval but also has its own context. There are many examples of this architecture in the prior art including SUN Solaris™, IBM AIX™, HP-UX™ and Linux™.

FIG. **2***a* shows a system architecture where critical system elements **27***a*, **27***b* execute in user mode but in a distinct context from applications. Some critical system elements are removed from the operating system kernel. They reside in multiple distinct processes or servers. An example of the architecture described in FIG. **2***a* is the GNU Hurd operating system.

The servers that export critical system elements execute in a context distinct from the operating system kernel and applications. These servers operate at a peer level with respect to other applications. Applications access system elements through application libraries. The libraries in this case communicate with multiple servers in order to access critical system elements. Thus, in the illustrated example, there are two application contexts and two critical system element contexts. When an application requires the use of a critical system element, which is being run in user mode, a sequence of events must take place. Typically the application first makes a platform independent call to the application library. The application library in turn makes a call to the operating system kernel. The operating system kernel then schedules the server with the critical system element in a different user mode context. After the server completes the operation, a switch back to kernel mode is made which then responds back to the application through the application library. Due to this architecture, such implementations may result in poor performance. Ideally, an application, which runs on the system of FIG. **1**, should be able to run on the system of FIG. **2***a* as well. However, in practice it is difficult to maintain the same characteristics and performance using such an architecture.

FIG. **2***b* is illustrative of a known system architecture where critical system elements execute in user mode. The critical system elements also execute in a distinct context from applications. Some critical system elements are removed from the operating system kernel. The essential difference between the architecture described in FIG. **2***a* and FIG. **2***b* is the use of a single process context to contain all user mode critical system elements. An example of the architecture described in FIG. **2***b* is Apple's MAC OS X™.

This invention is contrasted with all three of the prior art examples. Critical system elements as defined in the invention are not isolated in the operating system kernel as is the case of a monolithic architecture shown in FIG. **1**; also they are not removed from the context of an application, as is the case with a multi-server architecture depicted in FIG. **2***a*. Rather, they are replicated, and embodied in the context of an application.

FIG. **3** shows an architectural view of the overall operation of this invention. Multiple user processes execute above a single instance of an operating system.

Software applications **32***a*, **32***b*, utilize shared libraries **34** as is done in U.S. Provisional Patent Application Ser. No. 60/512,103 entitled "SOFTWARE SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS" which is incorporated herein by reference. The standard libraries are augmented by an extension, which contains critical system elements, that reside in extended shared libraries **35**. Extended services are similar to those that appear in the context of the operating system kernel **36**.

FIG. **4** illustrates the functionality of an application process as it exists above an operating system that was described in FIG. **3**. Applications exist and run in user mode while the operating system kernel **46** itself runs in kernel mode. User mode functionality includes the user applications **42**, the standard application libraries **44**, and one or more critical system elements, **45** & **47**. FIG. **4** shows one embodiment of the invention where the CSE functionality is contained in two shared libraries. An extended library, **45**, provides control operations while the CSE shared library, **47**, provides an implementation of a critical system element.

The CSE library includes replicas or substantial functional equivalents or replacements of kernel functions. The term replica, shall encompass any of these meanings, and although not a preferred embodiment, may even be a copy of a CSE that is part of the OS. These functions can be directly called by the applications **42** and as such can be run in the same context as the applications **42**. In preferred embodiments, the kernel functions which are included in the extended shared library and critical system element library **45**,**47** are also included in the operating system kernel **46**.

Furthermore, there might be different versions of a given critical system element **47** with different applications accessing these different versions within their respective context.

In preferred embodiments, the platform specific aspects of the critical system element are left in the operating system kernel. Then the critical system elements running in user mode may still make use of the operating system kernel to implement these platform specific functions.

FIG. **5** represents the function of the kernel module **58**, described in more detail below. A critical system element in the context of an application program uses system calls to access services in the kernel module. The kernel module serves as an interface between a service in the application context and a device driver. Specific device interrupts are vectored to the kernel module. A service in the context of an application is notified of an interrupt by the kernel module.

FIG. **6** represents interrupt handling. Interrupt handling is initialized through a system call. A handler contained in the kernel module **58** is installed for a specific device interrupt. When a hardware device generates an interrupt request the handler contained in the kernel module is called. The handler notifies a service in the context of an application through the use of an up call mechanism.

Function Overlays

A function overlay occurs when the implementation of a function that would normally be called, is replaced such that an extension or replacement function is called instead. The invention uses function overlays in one embodiment of the invention to intercept software application accesses to operating system services.

The overlay is accomplished by use of an ability supplied by the operating system to allow a library to be preloaded before other libraries are loaded. Such ability is used to cause the loading process, performed by the operating system to link the application to a library extension supplied by the invention rather than to the library that would otherwise be used. The use of this preload capability to attach a CSE to an application is believed to be novel.

The functions overlaid by the invention serve as extensions to operating system services. When a function is overlaid in this manner it enables the service defined by the API to be exported in an alternate manner than that provided by the operating system in kernel mode.

Critical System Elements

According to the invention, some system elements that are critical to the operation of a software application are replicated from kernel mode, into user mode in the same context as that of the application. These system elements are contained in a shared library. As such they are linked to a software application as the application is loaded. This is part of the operation performed when shared libraries are used. A linker/loader program is used to load and start an application. The process of starting the application includes creating a linkage to all of the required shared libraries that the application will need. The CSE is loaded and linked to the application as a part of this same process.

FIG. **3** shows that an extension library is utilized. In its native form, as it exists in the operating system kernel, a CSE uses services supplied by the operating system kernel. In order for the CSE to be migrated to user mode and operate effectively, the services that the CSE uses from the operating system kernel are replicated in user mode and contained in the shared library with the CSE itself. Services of the type referred to here include, but are not limited to, memory allocation, synchronization and device access. Preferably, as discussed above, platform specific services are not replicated, but rather are left in the operating system kernel. These will then be called by the critical system element running in user mode.

FIG. **4** shows that the invention allows for critical system elements to exist in the same context as an application. These services exported by library extensions do not replace those provided in an operating system kernel. Thus, in FIG. **4** the user process is shown to include the application itself, the regular application library, the extended library and the critical system element all of which are operating in user mode. The operating system kernel is also shown to include critical system elements. In preferred embodiments, the critical system elements which are included in user mode are replicas of elements which are still included in the operating system kernel. The term replication means that like services are supplied. As was described heretofore, it is not necessarily the case that duplicates of the same implementation found in the kernel are provided by a CSE; but essentially a same functionality is provided. As discussed previously, different applications may be provided with their own versions of the critical system elements.

Kernel Module

In some embodiments, control code is placed in kernel mode as shown in FIG. **4**. FIG. **5** shows that a kernel module is used to augment device access and interrupt notification. As a device interface the kernel module enables data exchange between a user mode CSE and a device driver in kernel mode. The exchange uses mapping of virtual memory such that data is transferred in both directions without a copy. Services exported for device interface typically include:

1. Initialization.
2. Establish a channel between a CSE in user mode and a specific device.
3. Informs the interrupt service that this CSE requires notification.
4. Write data.
5. Transfer data from a CSE to a device.
6. User mode virtual addresses are converted to kernel mode virtual addresses.
7. Read data.
8. Transfer data from a device to a CSE.
9. Kernel mode data is mapped into virtual addresses in user mode.
10. During initialization, interrupt services are informed that for specific interrupts, they should call a handler in the kernel module. The kernel module handles the interrupt by making an up call to the critical system element. Interrupts related to a device being serviced by a CSE in user mode are extended such that notification is given to the CSE in use.

As shown in FIG. **6** a handler is installed in the path of an interrupt. The handler uses an up call mechanism to inform the affected services in user mode. A user mode service enables interrupt notification through the use of an initialization function.

The general system configuration of the present invention discloses one possible implementation of the invention. In some embodiments, the 'C' programming language is used but other languages can alternatively be employed. Function overlays have been implemented through application library pre-load. A library supplied with the invention is loaded before the standard libraries, using standard services supplied by the operating system. This allows specific functions (APIs) used by an application to be overlaid or intercepted by services supplied by the invention. Access from a user mode CSE to the kernel module, for device I/O and registration of interrupt notification, is implemented by allowing the application to access the kernel module through standard device interfaces defined by the operating system. The kernel module is installed as a normal device driver. Once installed applications are able to open a device that corresponds to the module allowing effective communication as with any other device or file operation. Numerous modifications and variations of the present invention are possible in light of the above teachings without departing from the spirit and scope of the invention.

It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. A computing system for executing a plurality of software applications comprising:

a) a processor;

b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,

c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and

i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,

ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the

shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and

iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.

2. A computing system as defined in claim **1**, wherein in operation, multiple instances of an SLCSE stored in the shared library run simultaneously within the operating system.

3. A computing system according to claim **1** wherein OSCSEs corresponding to and capable of performing the same function as SLCSEs remain in the operating system kernel.

4. A computing system according to claim **1** wherein the one or more SLCSEs provided to one of the plurality of software applications having exclusive use thereof, use system calls to access services in the operating system kernel.

5. A computing system according to claim **1** wherein the operating system kernel comprises a kernel module adapted to serve as an interface between an SLCSE in the context of an application program and a device driver.

6. A computing system according to claim **5** wherein the kernel module is adapted to provide a notification of an event to an SLCSE running in the context of an application program, wherein the event is an asynchronous event and requires information to be passed to the SLCSE from outside the application.

7. A computing system according to claim **6** wherein a handler is provided for notifying the SLCSE in the context of one of the plurality of software applications through the use of an up call mechanism.

8. A computing system according to claim **7** wherein the up call mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode.

9. A computing system according to claim **2**, wherein a function overlay is used to provide one of the plurality of software applications access to operating system services.

10. A computing system according to claim **2** wherein SLCSEs stored in the shared library are linked to particular software applications of the plurality of software applications as the particular software applications are loaded such that the particular software applications have a link that provides unique access to a unique instance of a CSE.

11. A computing system according to claim **2** wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping.

12. A computing system according to claim **1**, wherein SLCSEs include services related to at least one of, network protocol processes, and the management of files.

13. A computing system according to claim **10** wherein some SLCSEs are modified for a particular one of the plurality of software applications.

14. A computing system according to claim **13** wherein the SLCSEs that are application specific, reside in user mode, while critical system elements, which are platform specific, reside in the operating system kernel.

15. A computing system according to claim **5** wherein the kernel module is adapted to enable data exchange between the SLCSEs in user mode and a device driver in kernel mode, and wherein the data exchange uses mapping of virtual memory such that data is transferred both from the SLCSEs in user mode to the device driver in kernel mode and from the device driver in kernel mode to the SLCSEs in user mode.

16. A computing system according to claim **1** wherein SLCSEs form a part of at least some of the plurality of software applications, by being linked thereto.

17. A computing system according to claim **2** wherein the SLCSEs utilize kernel services supplied by the operating system kernel for device access, interrupt delivery, and virtual memory mapping and otherwise execute without interaction from the operating system kernel.

18. A computer system as defined in claim **2** wherein SLCSEs are not copies of OSCSEs.

* * * * *

Exhibit 4

U.S. Patent No. 7,784,058 vs. Oracle

Accused Instrumentalities: Oracle products and services using user mode critical system elements as shared libraries, including without limitation Oracle Cloud Infrastructure ("OCI") and Oracle Kubernetes Engine ("OCE"), and all versions and variations thereof since the issuance of the asserted patent.

Claim 1

Claim 1	Accused Instrumentalities
[1pre] 1. A computing system for executing a plurality of software applications comprising:	To the extent the preamble is limiting, each Accused Instrumentality comprises or constitutes a computing system for executing a plurality of software applications as claimed. *See* claim limitations below. *See also, e.g.*:

Claim 1	Accused Instrumentalities
	Why Choose OKE? **Price-Performance** OKE is the lowest cost Kubernetes service amongst all hyperscalers, especially for serverless. **Autoscaling** OKE automatically adjusts compute resources based on demand, which can reduce your costs. **Efficiency** GPUs can be scarce, but OKE job scheduling makes it easy to maximize resource utilization. **Portability** OKE is consistent across clouds and on-premises, enabling portability and avoiding vendor lock-in. **Simplicity** OKE reduces the time and cost needed to manage the complexities of Kubernetes infrastructure. **Reliability** Automatic upgrades and security patching boost reliability for the control plane and worker nodes. https://www.oracle.com/cloud/cloud-native/kubernetes-engine/

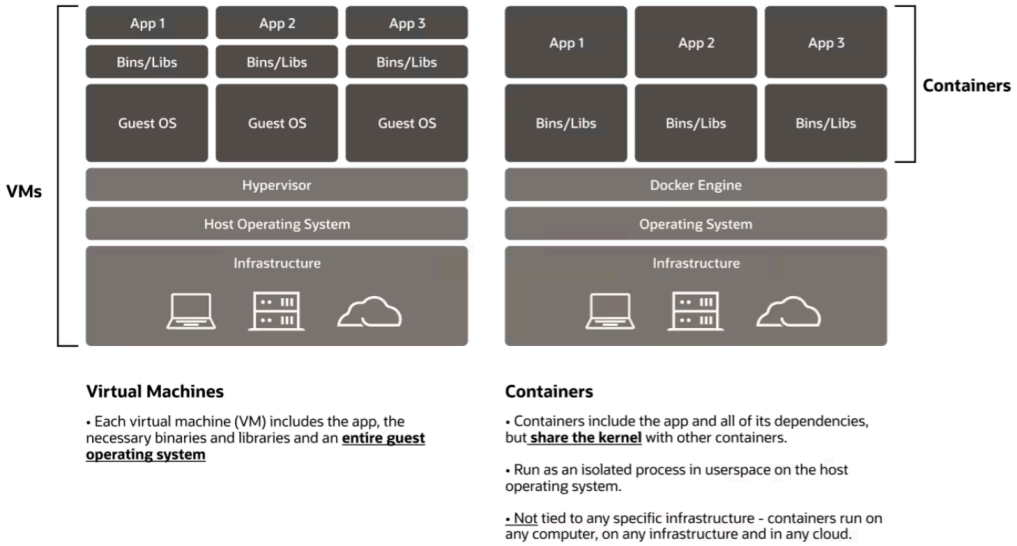
Claim 1	Accused Instrumentalities
	Welcome to Oracle Cloud Infrastructure Oracle Cloud Infrastructure (OCI) is a set of complementary cloud services that enable you to build and run a range of applications and services in a highly available hosted environment. OCI provides high-performance compute capabilities (as physical hardware instances) and storage capacity in a flexible overlay virtual network that is securely accessible from your on-premises network. https://docs.oracle.com/en-us/iaas/Content/GSG/Concepts/baremetalintro.htm **Existing applications can benefit by migrating to OCI and OKE** OKE offers lower total cost of ownership and improved time to market. OKE simplifies operations at scale in the following ways: - Lift and shift; there's no need to rearchitect - Reduce operations burden with automation - Save time on infrastructure management - Increase resource utilization and efficiency - Improve agility, flexibility, uptime, and resilience - Reduce compliance risk and enhance security https://www.oracle.com/cloud/cloud-native/kubernetes-engine/#app-migration

Claim 1	Accused Instrumentalities
	What is OCI Kubernetes Engine (OKE)? Oracle Cloud Infrastructure Kubernetes Engine (OKE) is a managed Kubernetes service that simplifies the development, deployment, and operation of containerized workloads at scale. OKE enables you to quickly create, manage, and consume Kubernetes clusters that leverage underlying OCI compute, networking, and storage services. **When should I use OKE?** You should use OKE when you want to leverage Kubernetes to deploy and manage your Kubernetes-based container applications. It allows you to combine the production-grade container orchestration of standard upstream Kubernetes with the control, security, and high, predictable performance of OCI. https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/ **How does OKE provide resiliency?** When you create an OKE cluster, OKE automatically creates and manages multiple Kubernetes control plane nodes spread across fault domains and availability domains (logical data centers). This is done to help ensure that the managed Kubernetes control plane is highly available. Control plane operations, such as upgrading to newer versions of Kubernetes, can be performed without service interruptions. Additionally, when you provision worker nodes, you can use a placement configuration to control the fault domain and availability domain where they are created. Nodes will automatically come online with labels, which you can use to schedule your workloads so they are robust and highly available. https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/ **Can I deploy private Kubernetes clusters?** Yes; with OKE, your Kubernetes clusters are integrated in your VCN. Your cluster worker nodes, load balancers, and the Kubernetes API endpoint are part of a private or public subnet of your VCN. Regular VCN routing and firewall rules control access to the Kubernetes API endpoint, making it accessible from a corporate network only, through a bastion host, or by specific platform services. https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/

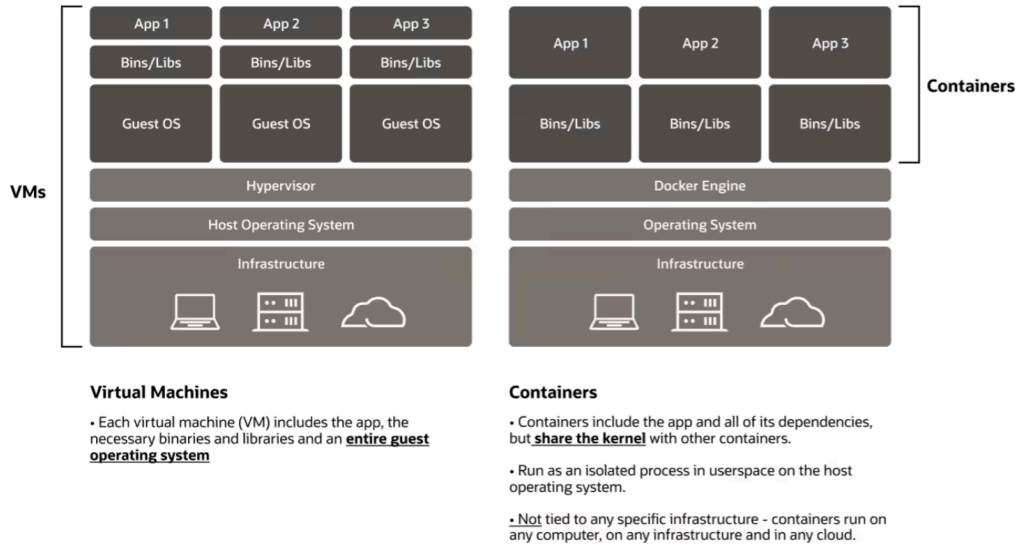
Claim 1	Accused Instrumentalities
	When should I use virtual nodes, managed nodes, or self-managed nodes? • **Virtual nodes** Virtual nodes offer a serverless Kubernetes experience. This option is ideal if you'd rather focus on your application and avoid managing the underlying infrastructure. Virtual nodes relieve you of management-related tasks such as scaling, upgrading, patching, troubleshooting, and provisioning worker nodes. • **Managed nodes** Managed nodes are a good choice for general purpose workloads. They offer an extensive list of customizable configuration options that have been tested by the OKE service. Unlike fully managed virtual nodes, you share the management of worker nodes with OCI. OKE simplifies the management process through features such as on-demand cycling to automate worker node updates, cluster self-healing upon failure detection, autoscaling, and more. • **Self-managed nodes** Self-managed nodes offer access to the underlying infrastructure, configuration options, and compute shapes that aren't currently available to managed nodes. This includes access to specialized infrastructure, such as RDMA-enabled bare metal cluster networks or confidential compute shapes. This advanced control makes self-managed nodes ideal for specialized use cases that aren't supported with managed nodes. Note that with self-managed nodes, you are fully responsible for managing the worker nodes—without the automated features provided by managed or virtual nodes. https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/ **What are the storage options for virtual nodes?** OKE virtual nodes do not yet have persistent storage capabilities. However, there are plans to introduce support for attaching persistent volumes backed by OCI Block Storage and OCI File Storage. If your Kubernetes application requires persistent storage, it's advisable to use OKE managed nodes. https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/

Claim 1	Accused Instrumentalities
	Supported Images for Managed Nodes Kubernetes Engine supports the provisioning of worker nodes (managed nodes only) using some, but not all, of the latest Oracle Linux images provided by Oracle Cloud Infrastructure. You can use these Oracle Linux images when provisioning managed nodes: • OKE Images • Platform Images • Custom Images https://docs.oracle.com/en-us/iaas/Content/ContEng/Reference/contengimagesshapes.htm **What is Docker?** A Docker container is a packaging format that packages all the code and dependencies of an application in a standard format that allows it to run quickly and reliably across computing environments. A Docker container is a popular lightweight, standalone, executable container that includes everything needed to run an application, including libraries, system tools, code, and runtime. Docker is also a software platform that allows developers to build, test, and deploy containerized applications quickly. Containers as a Service (CaaS) or Container Services are managed cloud services that manage the lifecycle of containers. Container services help orchestrate (start, stop, scale) the runtime of containers. Using container services, you can simplify, automate, and accelerate your application development and deployment lifecycle. Docker and Container Services have seen rapid adoption and have been a tremendous success over the last several years. From an almost unknown and rather technical open source technology in 2013, Docker has evolved into a standardized runtime environment now officially supported for many Oracle enterprise products. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

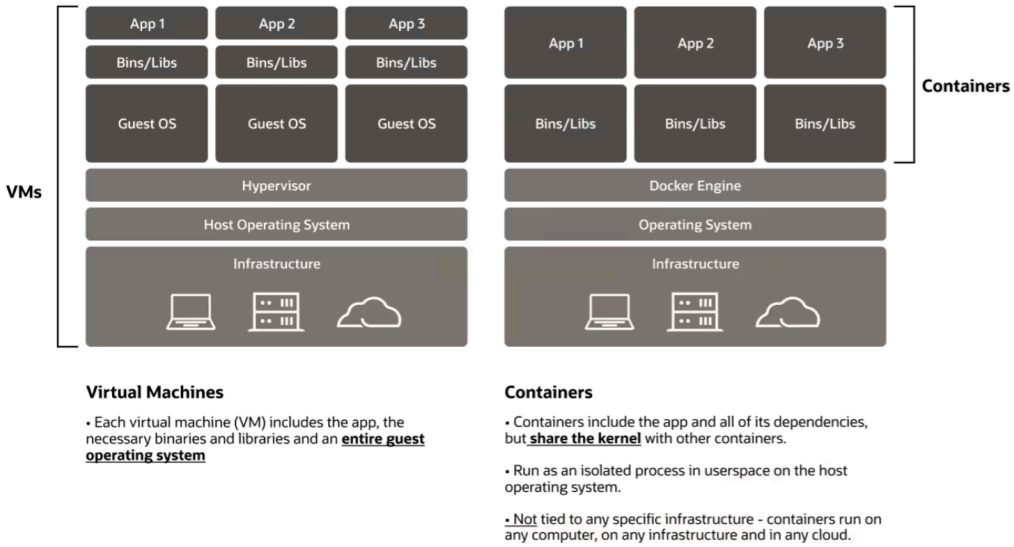
Claim 1	Accused Instrumentalities
	Container: Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the "user space." Containers share the host system's kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images. **Docker engine:** The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu. **Docker images:** Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image. **Docker Registry:** Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/ **Docker versus Kubernetes** Linux containers have existed since 2008, but they were not well known until the emergence of Docker containers in 2013. With the onset of Docker containers, came the explosion of interest in developing and deploying containerized applications. As the number of containerized applications grew to span hundreds of containers deployed across multiple servers, operating them became more complex. How do you coordinate, scale, manage, and schedule hundreds of containers? This is where Kubernetes can help. Kubernetes is an open source orchestration system that allows you to run your Docker containers and workloads. It helps you manage the operating complexities when moving to scale multiple containers deployed across multiple servers. The Kubernetes engine automatically orchestrates the container lifecycle, distributing the application containers across the hosting infrastructure. Kubernetes can quickly scale resources up or down, depending on the demand. It continually provisions, schedules, deletes, and monitors the health of the containers. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Docker Basics The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more. A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.  https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Container Cloud Services The first part of this article explained some important Docker concepts. However, in a production environment it is not enough to simply run an application in a Docker container. To setup and operate a production environment requires hardware to run the containers. Software such as Docker, along with repositories and cluster managers, must be installed, upgraded and patched. If several Docker containers communicate across hosts, a network must be created. Clustered containers should be restarted if they fail. In addition, a set of containers linked to each other should be deployable as easily as a single logical application instance. An example of this could be a load balancer, a few web servers, some Oracle WebLogic Server instances with an admin server, a managed server, and a database. To manage containerized applications at scale, requires a container orchestration system like Kubernetes or Docker Swarm. Deploying, managing, and operating orchestration systems like Kubernetes can be challenging and time-consuming. To make it easier and more efficient for developers to create containerized applications, cloud providers offer Container Cloud Services or Containers as a Service (CaaS). Container Cloud Services help developers and operations teams streamline and manage the lifecycle of containers in an automated fashion. These orchestration services, typically built using Kubernetes, make it easier for DevOps teams to manage and operate containerized applications at scale. Oracle Cloud Infrastructure Kubernetes Engine and Azure Kubernetes Service are two examples of popular container orchestration managed cloud services. Oracle Cloud Infrastructure Kubernetes Engine is a fully managed, scalable, and highly available service that you can use to deploy your containerized applications in the cloud. Use Kubernetes Engine (sometimes abbreviated to just OKE) when your development team wants to reliably build, deploy, and manage cloud native applications. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

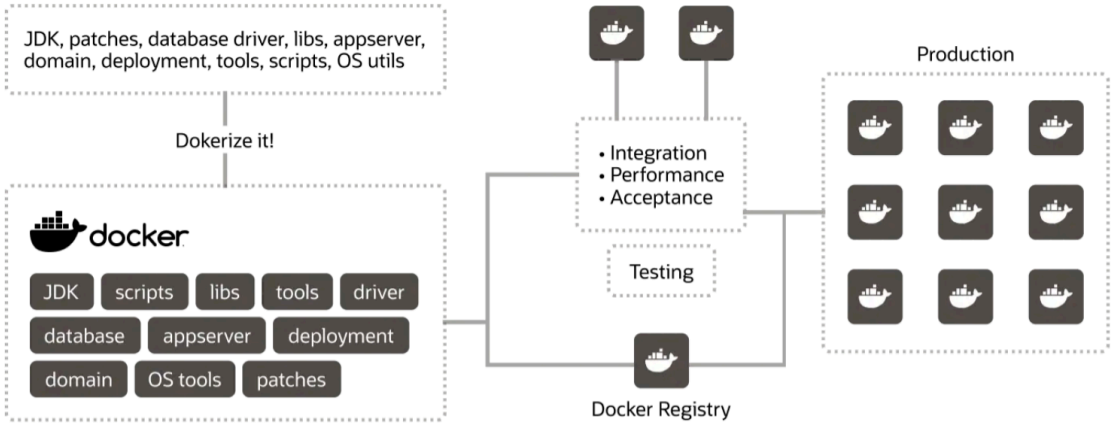
Claim 1	Accused Instrumentalities
[1a] a) a processor;	Each Accused Instrumentality comprises a processor. For example, each node/host contains at least one CPU. *See, e.g.*: **Docker Basics** The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more. A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.  https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	When should I use virtual nodes, managed nodes, or self-managed nodes? • **Virtual nodes** Virtual nodes offer a serverless Kubernetes experience. This option is ideal if you'd rather focus on your application and avoid managing the underlying infrastructure. Virtual nodes relieve you of management-related tasks such as scaling, upgrading, patching, troubleshooting, and provisioning worker nodes. • **Managed nodes** Managed nodes are a good choice for general purpose workloads. They offer an extensive list of customizable configuration options that have been tested by the OKE service. Unlike fully managed virtual nodes, you share the management of worker nodes with OCI. OKE simplifies the management process through features such as on-demand cycling to automate worker node updates, cluster self-healing upon failure detection, autoscaling, and more. • **Self-managed nodes** Self-managed nodes offer access to the underlying infrastructure, configuration options, and compute shapes that aren't currently available to managed nodes. This includes access to specialized infrastructure, such as RDMA-enabled bare metal cluster networks or confidential compute shapes. This advanced control makes self-managed nodes ideal for specialized use cases that aren't supported with managed nodes. Note that with self-managed nodes, you are fully responsible for managing the worker nodes—without the automated features provided by managed or virtual nodes. https://www.oracle.com/cloud/cloud-native/kubernetes-engine/faq/ Welcome to Oracle Cloud Infrastructure Oracle Cloud Infrastructure (OCI) is a set of complementary cloud services that enable you to build and run a range of applications and services in a highly available hosted environment. OCI provides high-performance compute capabilities (as physical hardware instances) and storage capacity in a flexible overlay virtual network that is securely accessible from your on-premises network. https://docs.oracle.com/en-us/iaas/Content/GSG/Concepts/baremetalintro.htm

Claim 1	Accused Instrumentalities
[1b] b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,	Each Accused Instrumentality comprises an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor. For example, the OSCSEs include kernel-mode functions similar to the functionalities provided by user-space libraries such as glibc. These are implemented in kernel-space to handle tasks such as (without limitation) memory management (kmalloc(), kfree(), etc,) at kernel level. *See, e.g.*: **Docker Basics** The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more. A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.  https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Kernel mode Kernel mode refers to the processor mode that enables software to have full and unrestricted access to the system and its resources. The OS kernel and kernel drivers, such as the file system driver, are loaded into protected memory space and operate in this highly privileged kernel mode. https://www.techtarget.com/searchdatacenter/definition/kernel The **GNU C Library**, commonly known as **glibc**, is the GNU Project implementation of the C standard library. It is a wrapper around the system calls of the Linux kernel for application use. Despite its name, it now also directly supports C++ (and, indirectly, other programming languages). It was started in the 1980s by the Free Software Foundation (FSF) for the GNU operating system. https://en.wikipedia.org/wiki/Glibc

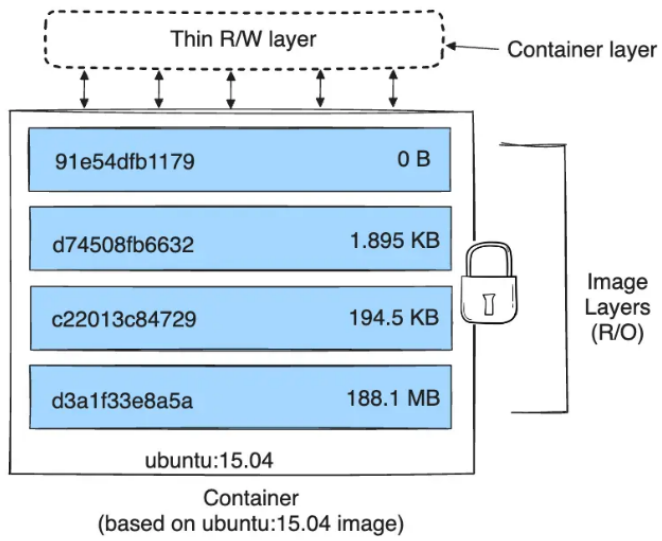
Claim 1	Accused Instrumentalities
[1c] c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and	Each Accused Instrumentality comprises a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode. For example, the shared library with SLCSEs include the runtime environment, system tools, and dependencies, such as the glibc library and other libraries that replicate OSCSEs, included in the container image (including without limitation in a base image that is included within the container image). *See, e.g.*: Figure 2 https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Docker Image—Development to Production Creating a Docker image with all of its dependencies solves the "but it worked for me on my development machine" problem. The key idea is that a Docker image is created automatically by a build pipeline from a source-code repository like Git and initially tested in a development environment. This immutable image will then be stored in a Docker registry. As shown in the Figure 4, the same image will be used for further load tests, integration tests, acceptance tests, and more. In every environment, the same image will be used. Small but necessary environmentally specific differences, such as a JDBC URL for a production database, can be fed into the container as environment variables or files.  https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Container images A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings. https://kubernetes.io/docs/concepts/containers/ Container image files are complete, static and executable versions of an application or service and differ from one technology to another. Docker images are made up of multiple layers, which start with a base image that includes all of the dependencies needed to execute code in a container. Each image has a readable/writable layer on top of static unchanging layers. Because each container has its own specific container layer that customizes that specific container, underlying image layers can be saved and reused in multiple containers. An Open Container Initiative (OCI) https://www.techtarget.com/searchitoperations/definition/container-containerization-or-container-based-virtualization

Claim 1	Accused Instrumentalities
	# About storage drivers To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way. ## Storage drivers versus Docker volumes Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer. Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance. https://docs.docker.com/storage/storagedriver/

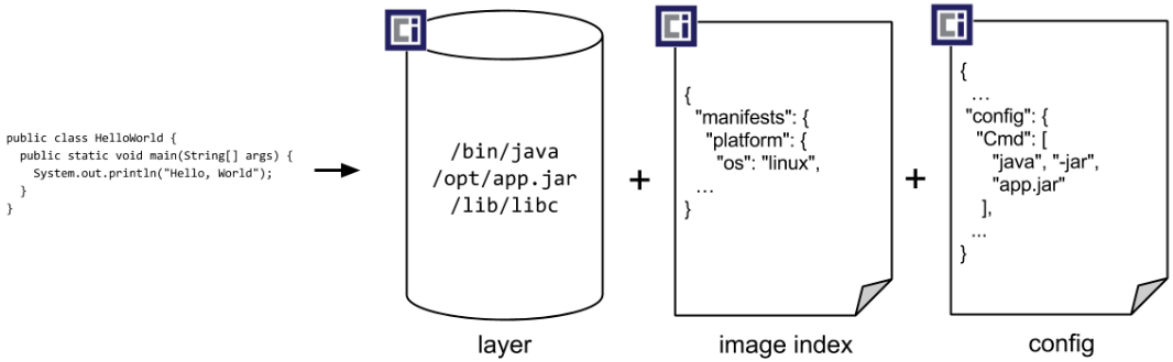
Claim 1	Accused Instrumentalities
	Images and layers A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile: ``` # syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r $HOME/.cache CMD python /app/app.py ``` This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The `FROM` statement starts out by creating a layer from the `ubuntu:22.04` image. The `LABEL` command only modifies the image's metadata, and doesn't produce a new layer. The `COPY` command adds some files from your Docker client's current directory. The first `RUN` command builds your application using the `make` command, and writes the result to a new layer. The second `RUN` command removes a cache directory, and writes the result to a new layer. Finally, the `CMD` instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer. https://docs.docker.com/storage/storagedriver/

Claim 1	Accused Instrumentalities
	Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images. The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.  https://docs.docker.com/storage/storagedriver/

Claim 1	Accused Instrumentalities
	Volumes Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts: https://kubernetes.io/docs/concepts/storage/volumes/ **Container environment** The Kubernetes Container environment provides several important resources to Containers: • A filesystem, which is a combination of an image and one or more volumes. • Information about the Container itself. • Information about other objects in the cluster. https://kubernetes.io/docs/concepts/containers/container-environment/

Claim 1	Accused Instrumentalities
	Images A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment. You typically create a container image of your application and push it to a registry before referring to it in a Pod. https://kubernetes.io/docs/concepts/containers/images/ **Volumes** On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a `Pod` and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes volume abstraction solves both of these problems. Familiarity with Pods is suggested. https://kubernetes.io/docs/concepts/storage/volumes/

Claim 1	Accused Instrumentalities
	# Open Container Initiative ## Image Format Specification This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration. The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md

Claim 1	Accused Instrumentalities
	Overview At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.  https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md

Claim 1	Accused Instrumentalities
	OCI Image Configuration An OCI *Image* is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers. This section defines the `application/vnd.oci.image.config.v1+json` media type. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md

Claim 1	Accused Instrumentalities
	Layer • Image filesystems are composed of *layers*. • Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer. • Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer. • Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. **Image JSON** • Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes. • The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers. • This JSON is considered to be immutable, because changing it would change the computed ImageID. • Changing it means creating a new derived image, instead of changing the existing image. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md

Claim 1	Accused Instrumentalities
	The **GNU C Library**, commonly known as **glibc**, is the GNU Project implementation of the C standard library. It is a wrapper around the system calls of the Linux kernel for application use. Despite its name, it now also directly supports C++ (and, indirectly, other programming languages). It was started in the 1980s by the Free Software Foundation (FSF) for the GNU operating system. https://en.wikipedia.org/wiki/Glibc

Claim 1	Accused Instrumentalities
[1d] i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications,	In each Accused Instrumentality, some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications. For example, a base image serves as a self-contained unit that encompasses all the necessary components for an application to run, including the application code, runtime environment, system tools, and dependencies (i.e., SLCSEs). The images are based on existing Linux distributions, such as Debian and Ubuntu, including essential system elements (i.e., functional replicas of OSCSEs). Each container image is based on a specific base image, which contains the application code, and dependencies, including system libraries or shared library critical system elements (SLCSEs). The base image forms a part of the container image according to the "layer" model described in the documentation below. When the container runs the image, it creates a runtime instance of that container image. In turn, when one or more applications executes within the container runtime environment, it dynamically links to the SLCSEs stored in the runtime environment, which thereby become a part of the application(s). *See, e.g.*: Container images A container image is a ready-to-run software package containing everything needed to run an application: the code and any runtime it requires, application and system libraries, and default values for any essential settings. https://kubernetes.io/docs/concepts/containers/

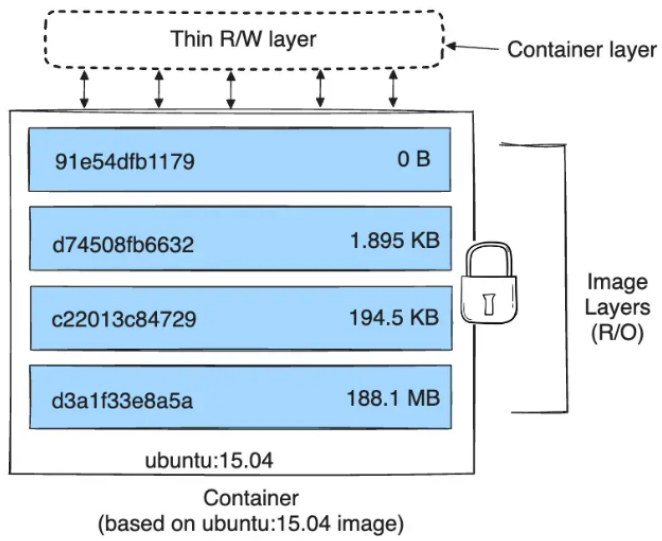
Claim 1	Accused Instrumentalities
	ubuntu Updated 15 days ago Ubuntu is a Debian-based Linux operating system based on free software. Linux IBM Z 386 riscv64 x86-64 ARM ARM 64 PowerPC 64 LE 1B+ · 10K+ debian Updated 35 minutes ago Debian is a Linux distribution that's composed entirely of free and open-source software. Linux riscv64 x86-64 ARM ARM 64 386 mips64le PowerPC 64 LE IBM Z 1B+ · 4.9K https://hub.docker.com/search?image_filter=official&type=image&q=

Platform	x86_64 / amd64	arm64 / aarch64	arm (32-bit)	ppc64le	s390x
CentOS	✅	✅		✅	
Debian	✅	✅	✅	✅	
Fedora	✅	✅		✅	
Raspberry Pi OS (32-bit)			✅		
RHEL (s390x)					✅
SLES					✅
Ubuntu	✅	✅	✅	✅	✅
Binaries	✅	✅	✅		

https://docs.docker.com/engine/install/

Claim 1	Accused Instrumentalities
	Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container. https://www.techtarget.com/searchitoperations/definition/Docker-image **About storage drivers** To use storage drivers effectively, it's important to know how Docker builds and stores images, and how these images are used by containers. You can use this information to make informed choices about the best way to persist data from your applications and avoid performance problems along the way. **Storage drivers versus Docker volumes** Docker uses storage drivers to store image layers, and to store data in the writable layer of a container. The container's writable layer doesn't persist after the container is deleted, but is suitable for storing ephemeral data that is generated at runtime. Storage drivers are optimized for space efficiency, but (depending on the storage driver) write speeds are lower than native file system performance, especially for storage drivers that use a copy-on-write filesystem. Write-intensive applications, such as database storage, are impacted by a performance overhead, particularly if pre-existing data exists in the read-only layer. Use Docker volumes for write-intensive data, data that must persist beyond the container's lifespan, and data that must be shared between containers. Refer to the volumes section to learn how to use volumes to persist data and improve performance. https://docs.docker.com/storage/storagedriver/

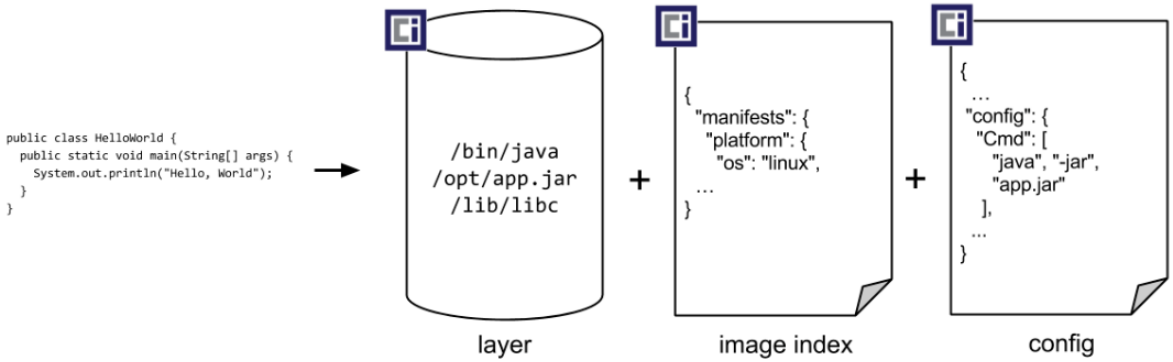
Claim 1	Accused Instrumentalities
	Images and layers A Docker image is built up from a series of layers. Each layer represents an instruction in the image's Dockerfile. Each layer except the very last one is read-only. Consider the following Dockerfile: ``` # syntax=docker/dockerfile:1 FROM ubuntu:22.04 LABEL org.opencontainers.image.authors="org@example.com" COPY . /app RUN make /app RUN rm -r $HOME/.cache CMD python /app/app.py ``` This Dockerfile contains four commands. Commands that modify the filesystem create a layer. The `FROM` statement starts out by creating a layer from the `ubuntu:22.04` image. The `LABEL` command only modifies the image's metadata, and doesn't produce a new layer. The `COPY` command adds some files from your Docker client's current directory. The first `RUN` command builds your application using the `make` command, and writes the result to a new layer. The second `RUN` command removes a cache directory, and writes the result to a new layer. Finally, the `CMD` instruction specifies what command to run within the container, which only modifies the image's metadata, which doesn't produce an image layer. https://docs.docker.com/storage/storagedriver/

Claim 1	Accused Instrumentalities
	Each layer is only a set of differences from the layer before it. Note that both *adding*, and *removing* files will result in a new layer. In the example above, the `$HOME/.cache` directory is removed, but will still be available in the previous layer and add up to the image's total size. Refer to the Best practices for writing Dockerfiles and use multi-stage builds sections to learn how to optimize your Dockerfiles for efficient images. The layers are stacked on top of each other. When you create a new container, you add a new writable layer on top of the underlying layers. This layer is often called the "container layer". All changes made to the running container, such as writing new files, modifying existing files, and deleting files, are written to this thin writable container layer. The diagram below shows a container based on an `ubuntu:15.04` image.  https://docs.docker.com/storage/storagedriver/

Claim 1	Accused Instrumentalities
	Volumes Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker. Volumes have several advantages over bind mounts: https://kubernetes.io/docs/concepts/storage/volumes/ **Container environment** The Kubernetes Container environment provides several important resources to Containers: • A filesystem, which is a combination of an image and one or more volumes. • Information about the Container itself. • Information about other objects in the cluster. https://kubernetes.io/docs/concepts/containers/container-environment/

Claim 1	Accused Instrumentalities
	Images A container image represents binary data that encapsulates an application and all its software dependencies. Container images are executable software bundles that can run standalone and that make very well defined assumptions about their runtime environment. You typically create a container image of your application and push it to a registry before referring to it in a Pod. https://kubernetes.io/docs/concepts/containers/images/ **Volumes** On-disk files in a container are ephemeral, which presents some problems for non-trivial applications when running in containers. One problem occurs when a container crashes or is stopped. Container state is not saved so all of the files that were created or modified during the lifetime of the container are lost. During a crash, kubelet restarts the container with a clean state. Another problem occurs when multiple containers are running in a `Pod` and need to share files. It can be challenging to setup and access a shared filesystem across all of the containers. The Kubernetes volume abstraction solves both of these problems. Familiarity with Pods is suggested. https://kubernetes.io/docs/concepts/storage/volumes/

Claim 1	Accused Instrumentalities
	# Open Container Initiative ## Image Format Specification This specification defines an OCI Image, consisting of an image manifest, an image index (optional), a set of filesystem layers, and a configuration. The goal of this specification is to enable the creation of interoperable tools for building, transporting, and preparing a container image to run. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md

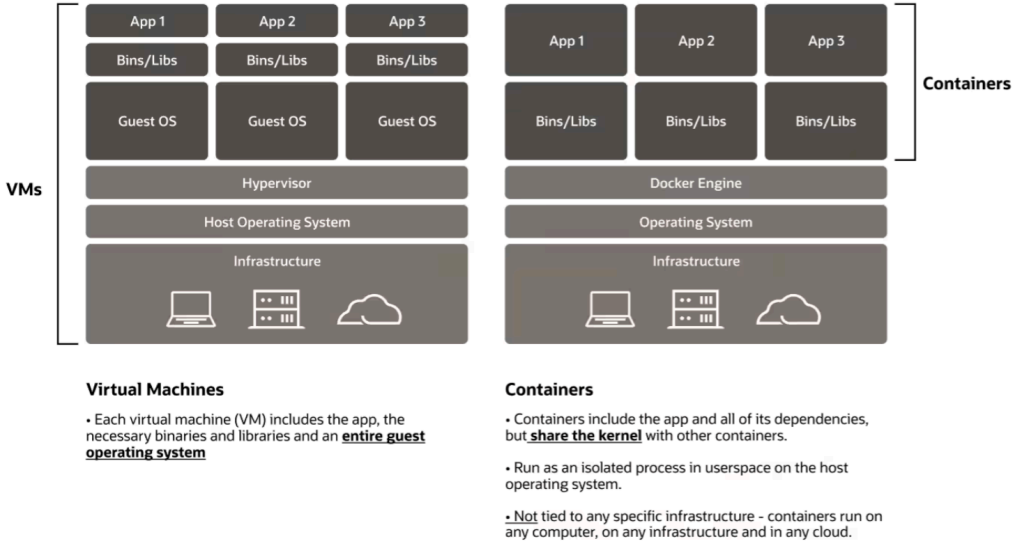
Claim 1	Accused Instrumentalities
	Overview At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.  https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md

Claim 1	Accused Instrumentalities
	OCI Image Configuration An OCI *Image* is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime. This specification outlines the JSON format describing images for use with a container runtime and execution tool and its relationship to filesystem changesets, described in Layers. This section defines the `application/vnd.oci.image.config.v1+json` media type. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md

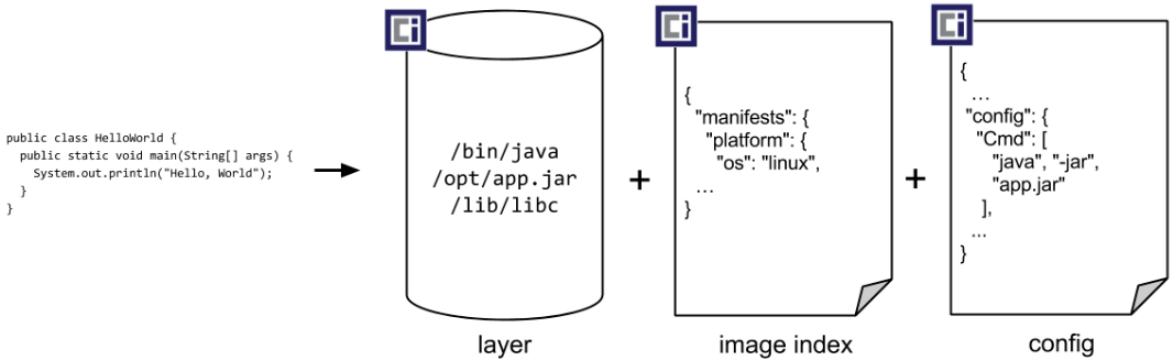
Claim 1	Accused Instrumentalities
	Layer • Image filesystems are composed of *layers*. • Each layer represents a set of filesystem changes in a tar-based layer format, recording files to be added, changed, or deleted relative to its parent layer. • Layers do not have configuration metadata such as environment variables or default arguments - these are properties of the image as a whole rather than any particular layer. • Using a layer-based or union filesystem such as AUFS, or by computing the diff from filesystem snapshots, the filesystem changeset can be used to present a series of image layers as if they were one cohesive filesystem. **Image JSON** • Each image has an associated JSON structure which describes some basic information about the image such as date created, author, as well as execution/runtime configuration like its entrypoint, default arguments, networking, and volumes. • The JSON structure also references a cryptographic hash of each layer used by the image, and provides history information for those layers. • This JSON is considered to be immutable, because changing it would change the computed ImageID. • Changing it means creating a new derived image, instead of changing the existing image. https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/config.md Containers only have access to resources that are defined in the image, https://www.hpe.com/us/en/what-is/docker.html

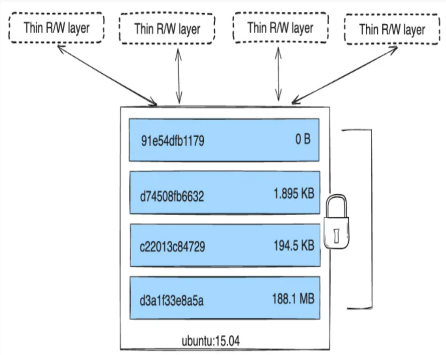
Claim 1	Accused Instrumentalities
	DESCRIPTION top `The programs ld.so and ld-linux.so* find and load the shared objects (shared libraries) needed by a program, prepare the program to run, and then run it.` https://man7.org/linux/man-pages/man8/ld.so.8.html
[1e] ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function, and	In each Accused Instrumentality, an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing same function. When a Docker or Kubernetes image is used to create a container in the Accused Instrumentalities, it creates a separate and isolated instance of a runtime environment which is independent of other containers running on the same host. Each container has its own instance of base images and its own data. The containers run in isolation, ensuring that the SLCSEs stored in the shared library are accessible to the software applications running in their respective containers. The image includes essential system files, libraries, and dependencies required to run the software application within the container. The containers can share common dependencies and components using layered images. This means that multiple containers utilize the same base image to create an instance. When an instance of SLCSE is provided from the base image (i.e., from the shared library) to an individual container including application software, it is not shared with other containers. This ensures that each container has its own isolated context. Docker or Kubernetes containers in the Accused Instrumentalities can share common dependencies and components using layered images. This means that multiple containers can utilize the same base image. Therefore, each container, containing the application software running under the operating system, utilizes a unique instance of the corresponding critical system element to execute the respective application software for performing a same function. *See, e.g.*:

Claim 1	Accused Instrumentalities
	Cgroups and Namespaces History The underlying Linux kernel features that Docker uses are cgroups and namespaces. In 2008 cgroups were introduced to the Linux kernel based on work previously done by Google developers[1]. Cgroups limit and account for the resource usage of a set of operating system processes. The Linux kernel uses namespace to isolate the system resources of processes from each other. The first namespace, i.e. the mount namespace, was introduced as early as 2002.[2] https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/ **Container:** Unlike a VM which provides hardware virtualization, a container provides lightweight, operating-system-level virtualization by abstracting the "user space." Containers share the host system's kernel with other containers. A container, which runs on the host operating system, is a standard software unit that packages code and all its dependencies, so applications can run quickly and reliably from one environment to another. Containers are nonpersistent and are spun up from images. **Docker engine:** The open source host software building and running the containers. Docker Engines act as the client-server application supporting containers on various Windows servers and Linux operating systems, including Oracle Linux, CentOS, Debian, Fedora, RHEL, SUSE, and Ubuntu. **Docker images:** Collection of software to be run as a container that contains a set of instructions for creating a container that can run on the Docker platform. Images are immutable, and changes to an image require to build a new image. **Docker Registry:** Place to store and download images. The registry is a stateless and scalable server-side application that stores and distributes Docker images. https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	Docker Basics The core concepts of Docker are images and containers. A Docker image contains everything that is needed to run your software: the code, a runtime (for example, Java Virtual Machine (JVM), drivers, tools, scripts, libraries, deployments, and more. A Docker container is a running instance of a Docker image. However, unlike in traditional virtualization with a type 1 or type 2 hypervisor, a Docker container runs on the kernel of the host operating system. Within a Docker image there is no separate operating system, as illustrated in Figure 1.  https://www.oracle.com/in/cloud/cloud-native/container-registry/what-is-docker/

Claim 1	Accused Instrumentalities
	# Setting Up Storage for Kubernetes Clusters *Find out how to define and apply persistent volume claims to clusters you've created using Kubernetes Engine (OKE). With Oracle Cloud Infrastructure as the underlying IaaS provider, you can provision persistent volume claims by attaching volumes from the Block Volume service or by mounting file systems from the File Storage service.* Container storage via a container's root file system is ephemeral, and can disappear upon container deletion and creation. To provide a durable location to prevent data from being lost, you can create and use persistent volumes to store data outside of containers. A persistent volume offers persistent storage that enables your data to remain intact, regardless of whether the containers to which the storage is connected are terminated. A persistent volume claim (PVC) is a request for storage, which is met by binding the PVC to a persistent volume (PV). A PVC provides an abstraction layer to the underlying storage. With Oracle Cloud Infrastructure, you can provision persistent volume claims: • By attaching volumes from the Oracle Cloud Infrastructure Block Volume service. The volumes are connected to clusters created by Kubernetes Engine using CSI (Container Storage Interface) or FlexVolume volume plugins deployed on the clusters. Oracle recommends the CSI volume plugin since the upstream Kubernetes project deprecates the FlexVolume volume plugin in Kubernetes version 1.23. See Provisioning PVCs on the Block Volume Service. • By mounting file systems in the Oracle Cloud Infrastructure File Storage service. The File Storage service file systems are mounted inside containers running on clusters created by Kubernetes Engine using a CSI (Container Storage Interface) volume plugin deployed on the clusters. See Provisioning PVCs on the File Storage Service. https://docs.oracle.com/en-us/iaas/Content/ContEng/Tasks/contengcreatingpersistentvolumeclaim.htm

Claim 1	Accused Instrumentalities
	Overview At a high level the image manifest contains metadata about the contents and dependencies of the image including the content-addressable identity of one or more filesystem layer changeset archives that will be unpacked to make up the final runnable filesystem. The image configuration includes information such as application arguments, environments, etc. The image index is a higher-level manifest which points to a list of manifests and descriptors. Typically, these manifests may provide different implementations of the image, possibly varying by platform or other attributes.  https://github.com/opencontainers/image-spec/blob/a6af2b480dcfc001ba975f44de53001c873cb0ef/spec.md

Claim 1	Accused Instrumentalities
	Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.  https://docs.docker.com/storage/storagedriver/ Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container. https://www.techtarget.com/searchitoperations/definition/Docker-image
[1f] iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.	In each Accused Instrumentality, a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously. For example, in Docker or Kubernetes containers used within the Accused Instrumentalities, each container operates independently, and a base image includes essential system files, libraries, and dependencies (i.e., SLCSEs) required to run the software application within the container. Based on information and belief, each element, such as system files, libraries, and dependencies (i.e., SLCSE) is associated with an execution of a predetermined function related to the application. When an image is used to create a container in the Accused Instrumentality, an instance of the SLCSE is provided to a

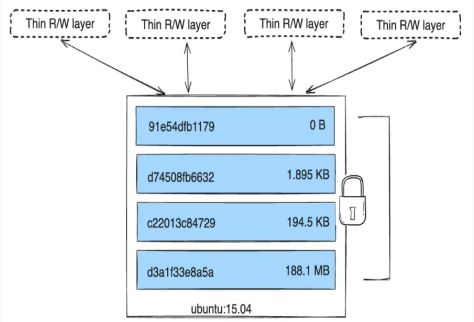
Claim 1	Accused Instrumentalities
	software application. Therefore, different instances of the SLCSE are provided to different applications for performing a same function, simultaneously. *See, e.g.*: Docker is used to create, run and deploy applications in containers. A Docker image contains application code, libraries, tools, dependencies and other files needed to make an application run. When a user runs an image, it can become one or many instances of a container. https://www.techtarget.com/searchitoperations/definition/Docker-image A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state. https://docs.docker.com/get-started/overview/ Because each container has its own writable container layer, and all changes are stored in this container layer, multiple containers can share access to the same underlying image and yet have their own data state. The diagram below shows multiple containers sharing the same Ubuntu 15.04 image.  https://docs.docker.com/storage/storagedriver/

Exhibit 5

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

NOTICE OF ALLOWANCE AND FEE(S) DUE

27975 7590 12/10/2008

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

EXAMINER	
BAUM, RONALD	
ART UNIT	PAPER NUMBER
2439	

DATE MAILED: 12/10/2008

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/939,903	09/13/2004	Donn Rochette	78802 (120-1 US)	5216

TITLE OF INVENTION: SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS

APPLN. TYPE	SMALL ENTITY	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	YES	$755	$300	$0	$1055	03/10/2009

THE APPLICATION IDENTIFIED ABOVE HAS BEEN EXAMINED AND IS ALLOWED FOR ISSUANCE AS A PATENT. PROSECUTION ON THE MERITS IS CLOSED. THIS NOTICE OF ALLOWANCE IS NOT A GRANT OF PATENT RIGHTS. THIS APPLICATION IS SUBJECT TO WITHDRAWAL FROM ISSUE AT THE INITIATIVE OF THE OFFICE OR UPON PETITION BY THE APPLICANT. SEE 37 CFR 1.313 AND MPEP 1308.

THE ISSUE FEE AND PUBLICATION FEE (IF REQUIRED) MUST BE PAID WITHIN THREE MONTHS FROM THE MAILING DATE OF THIS NOTICE OR THIS APPLICATION SHALL BE REGARDED AS ABANDONED. THIS STATUTORY PERIOD CANNOT BE EXTENDED. SEE 35 U.S.C. 151. THE ISSUE FEE DUE INDICATED ABOVE DOES NOT REFLECT A CREDIT FOR ANY PREVIOUSLY PAID ISSUE FEE IN THIS APPLICATION. IF AN ISSUE FEE HAS PREVIOUSLY BEEN PAID IN THIS APPLICATION (AS SHOWN ABOVE), THE RETURN OF PART B OF THIS FORM WILL BE CONSIDERED A REQUEST TO REAPPLY THE PREVIOUSLY PAID ISSUE FEE TOWARD THE ISSUE FEE NOW DUE.

HOW TO REPLY TO THIS NOTICE:

I. Review the SMALL ENTITY status shown above.

If the SMALL ENTITY is shown as YES, verify your current SMALL ENTITY status:

A. If the status is the same, pay the TOTAL FEE(S) DUE shown above.

B. If the status above is to be removed, check box 5b on Part B - Fee(s) Transmittal and pay the PUBLICATION FEE (if required) and twice the amount of the ISSUE FEE shown above, or

If the SMALL ENTITY is shown as NO:

A. Pay TOTAL FEE(S) DUE shown above, or

B. If applicant claimed SMALL ENTITY status before, or is now claiming SMALL ENTITY status, check box 5a on Part B - Fee(s) Transmittal and pay the PUBLICATION FEE (if required) and 1/2 the ISSUE FEE shown above.

II. PART B - FEE(S) TRANSMITTAL, or its equivalent, must be completed and returned to the United States Patent and Trademark Office (USPTO) with your ISSUE FEE and PUBLICATION FEE (if required). If you are charging the fee(s) to your deposit account, section "4b" of Part B - Fee(s) Transmittal should be completed and an extra copy of the form should be submitted. If an equivalent of Part B is filed, a request to reapply a previously paid issue fee must be clearly made, and delays in processing may occur due to the difficulty in recognizing the paper as an equivalent of Part B.

III. All communications regarding this application must give the application number. Please direct all communications prior to issuance to Mail Stop ISSUE FEE unless advised to the contrary.

IMPORTANT REMINDER: Utility patents issuing on applications filed on or after Dec. 12, 1980 may require payment of maintenance fees. It is patentee's responsibility to ensure timely payment of maintenance fees when due.

PTOL-85 (Rev. 08/07) Approved for use through 08/31/2010.

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail **Mail Stop ISSUE FEE**
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450
or Fax (571)-273-2885

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 5 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Use Block 1 for any change of address)

27975 7590 12/10/2008

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission

I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO (571) 273-2885, on the date indicated below.

(Depositor's name)
(Signature)
(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/939,903	09/13/2004	Donn Rochette	78802 (120-1 US)	5216

TITLE OF INVENTION: SYSTEM FOR CONTAINERIZATION OF APPLICATION SETS

APPLN. TYPE	SMALL ENTITY	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	YES	$755	$300	$0	$1055	03/10/2009

EXAMINER	ART UNIT	CLASS-SUBCLASS
BAUM, RONALD	2439	713-167000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).
❑ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.
❑ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list
(1) the names of up to 3 registered patent attorneys or agents OR, alternatively,
(2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.
1_____
2_____
3_____

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)

PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. If an assignee is identified below, the document has been filed for recordation as set forth in 37 CFR 3.11. Completion of this form is NOT a substitute for filing an assignment.

(A) NAME OF ASSIGNEE (B) RESIDENCE: (CITY and STATE OR COUNTRY)

Please check the appropriate assignee category or categories (will not be printed on the patent) : ❑ Individual ❑ Corporation or other private group entity ❑ Government

4a. The following fee(s) are submitted:
❑ Issue Fee
❑ Publication Fee (No small entity discount permitted)
❑ Advance Order - # of Copies _____

4b. Payment of Fee(s): **(Please first reapply any previously paid issue fee shown above)**
❑ A check is enclosed.
❑ Payment by credit card. Form PTO-2038 is attached.
❑ The Director is hereby authorized to charge the required fee(s), any deficiency, or credit any overpayment, to Deposit Account Number _____ (enclose an extra copy of this form).

5. **Change in Entity Status** (from status indicated above)
❑ a. Applicant claims SMALL ENTITY status. See 37 CFR 1.27.
❑ b. Applicant is no longer claiming SMALL ENTITY status. See 37 CFR 1.27(g)(2).

NOTE: The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

Authorized Signature _____ Date _____

Typed or printed name _____ Registration No. _____

This collection of information is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313-1450.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

Oracle Ex. 1113, p. 162
Oracle v. VirtaMove
IPR2025-00982

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/939,903	09/13/2004	Donn Rochette	78802 (120-1 US)	5216

27975 7590 12/10/2008

ALLEN, DYER, DOPPELT, MILBRATH & GILCHRIST P.A.
1401 CITRUS CENTER 255 SOUTH ORANGE AVENUE
P.O. BOX 3791
ORLANDO, FL 32802-3791

EXAMINER
BAUM, RONALD

ART UNIT	PAPER NUMBER
2439	

DATE MAILED: 12/10/2008

Determination of Patent Term Adjustment under 35 U.S.C. 154 (b)
(application filed on or after May 29, 2000)

The Patent Term Adjustment to date is 933 day(s). If the issue fee is paid on the date that is three months after the mailing date of this notice and the patent issues on the Tuesday before the date that is 28 weeks (six and a half months) after the mailing date of this notice, the Patent Term Adjustment will be 933 day(s).

If a Continued Prosecution Application (CPA) was filed in the above-identified application, the filing date that determines Patent Term Adjustment is the filing date of the most recent CPA.

Applicant will be able to obtain more detailed information by accessing the Patent Application Information Retrieval (PAIR) WEB site (http://pair.uspto.gov).

Any questions regarding the Patent Term Extension or Adjustment determination should be directed to the Office of Patent Legal Administration at (571)-272-7702. Questions relating to issue and publication fee payments should be directed to the Customer Service Center of the Office of Patent Publication at 1-(888)-786-0101 or (571)-272-4200.

PTOL-85 (Rev. 08/07) Approved for use through 08/31/2010.

Notice of Allowability	Application No.	Applicant(s)
	10/939,903	ROCHETTE ET AL.
	Examiner	**Art Unit**
	RONALD BAUM	2439

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. ☒ This communication is responsive to *03 September 2008*.

2. ☒ The allowed claim(s) is/are *1-34*.

3. ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
   a) ☐ All b) ☐ Some* c) ☐ None of the:
   1. ☐ Certified copies of the priority documents have been received.
   2. ☐ Certified copies of the priority documents have been received in Application No. _____ .
   3. ☐ Copies of the certified copies of the priority documents have been received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

   * Certified copies not received: _____.

Applicant has THREE MONTHS FROM THE "MAILING DATE" of this communication to file a reply complying with the requirements noted below. Failure to timely comply will result in ABANDONMENT of this application.
THIS THREE-MONTH PERIOD IS NOT EXTENDABLE.

4. ☐ A SUBSTITUTE OATH OR DECLARATION must be submitted. Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL PATENT APPLICATION (PTO-152) which gives reason(s) why the oath or declaration is deficient.

5. ☐ CORRECTED DRAWINGS (as "replacement sheets") must be submitted.
   (a) ☐ including changes required by the Notice of Draftsperson's Patent Drawing Review (PTO-948) attached
   1) ☐ hereto or 2) ☐ to Paper No./Mail Date _____.
   (b) ☐ including changes required by the attached Examiner's Amendment / Comment or in the Office action of Paper No./Mail Date _____.

   Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the drawings in the front (not the back) of each sheet. Replacement sheet(s) should be labeled as such in the header according to 37 CFR 1.121(d).

6. ☐ DEPOSIT OF and/or INFORMATION about the deposit of BIOLOGICAL MATERIAL must be submitted. Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Attachment(s)

1. ☒ Notice of References Cited (PTO-892)
2. ☐ Notice of Draftperson's Patent Drawing Review (PTO-948)
3. ☐ Information Disclosure Statements (PTO/SB/08), Paper No./Mail Date _____
4. ☐ Examiner's Comment Regarding Requirement for Deposit of Biological Material
5. ☐ Notice of Informal Patent Application
6. ☐ Interview Summary (PTO-413), Paper No./Mail Date _____ .
7. ☐ Examiner's Amendment/Comment
8. ☒ Examiner's Statement of Reasons for Allowance
9. ☐ Other _____.

DETAILED ACTION

Examiner's Statement of Reasons for Allowance

1. Claims 1-34 are allowed over prior art.

2. This action is in reply to applicant's correspondence of 03 September 2008.

3. The following is an examiner's statement of reasons for the indication of allowable claimed subject matter.

4. As per claims 1 and 17 generally, prior art of record, Forbes et al, U.S. Patent 6,381,742 B2, fails to anticipate, disclose, teach or suggest alone, or in combination, at the time of the invention, the features as set forth in the claims in this application as allowed, and not necessarily as summarized and/or characterized by the examiner, whether or not as italicized, as discussed and remarked upon in the response of 03 September 2008 to office action of 03 June 2008.

Specifically, (as per claim 1, for example) prior art dealing with various aspects of systems that run multiple applications (often for a single user running unrelated applications scenarios), across independent systems/disparate workstations, utilizing multiple, often different operating systems (i.e., scenarios that require system architectures that require system virtualization with high degrees of both isolation and efficiency (e.g., Hypervisors), alternatives utilizing resource containers and security containers applied to general-purpose, time-shared operating systems exist, such as the Linux-VServer; Soltesz, S., et al, 'Container-based Operating System Virtualization:A Scalable, High-performance Alternative to Hypervisors', SIGOPS Oper. Syst. Rev., Vol. 41, No. 3. (June 2007), pp. 275-287, entire article, http://delivery.acm.org/10.1145/1280000/1273025/p275-soltesz.pdf?key1=1273025&key2=0279528221&coll=GUIDE&dl=GUIDE&CFID=13091697&CFTOKEN=46675559), is generally known per se. Nowhere in the prior art is found collectively the *italicized* claim

elements (i.e., the various aspects of applications software not being sharable between the plurality of secure (and isolated) containers of application software, and unique root file systems different from an operating system's root file system, so as to allow for different versions of the same operating system running on the same system/server environment), at the ***time*** *of the invention*, serving to patently distinguish the invention from said prior art;

"1. In a system

having *a* ***plurality*** *of servers* with

operating systems ***that differ***,

operating in

disparate *computing environments*,

wherein each server includes

a processor and

an operating system including

a kernel

a set of associated local system files

compatible with the processor,

a method of providing at least some of the servers in the system with

secure, executable, ***applications***

related to ***a service***,

wherein the applications

are executed in

a secure environment,

wherein the *applications* ***each*** *include*

an object *executable by*

at least ***some of the different*** *operating systems*

for performing a task

related to the service,

the method comprising:

storing in memory ***accessible*** *to at least some of the servers*

a *plurality of secure containers of application software*,

each *container* comprising

one or more of the *executable applications* and

a set of *associated system files*

required to execute the one or more *applications*,

for use with ***a local kernel***

residing ***permanently on***

one of the servers;

wherein the set of *associated system files*

are *compatible with*

a local kernel of

at least *some of the* ***plurality*** *of* ***different*** *operating systems*,

the ***containers of***

application software

excluding a kernel,

wherein some or all of the *associated* ***system files***

within *a container* stored in memory

are *utilized* ***in place of***

the associated ***local system files***

that ***remain resident*** *on the server*,

wherein *said associated system files*

utilized in place of

the associated *local system files*

are ***copies or modified copies*** *of*

the *associated* ***local*** *system files*

that ***remain resident on the server***, and

wherein the ***application*** *software*

cannot be shared between

the plurality of ***secure*** *containers of* ***application software***, and

wherein ***each*** *of the containers has*

a unique *root file system*

that is ***different*** *from*

an operating system's root file system."

5. Dependent claims 2-16 and 18-34 are allowable by virtue of their dependencies.

Conclusion

6. Any inquiry concerning this communication or earlier communications from examiner should be directed to Ronald Baum, whose telephone number is (571) 272-3861, and whose unofficial Fax number is (571) 273-3861 and unofficial email is Ronald.baum@uspto.gov. The examiner can normally be reached Monday through Thursday from 8:00 AM to 5:30 PM.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kambiz Zand, can be reached at (571) 272-3811. The Fax number for the organization where this application is assigned is **571-273-8300**.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. For more information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Ronald Baum

Patent Examiner

/R. B./

Examiner, Art Unit 2439

/Kambiz Zand/

Supervisory Patent Examiner, Art Unit 2434

Exhibit 6

Notice of Allowability	Application No.	Applicant(s)
	10/946,536	ROCHETTE ET AL.
	Examiner	**Art Unit**
	SYED RONI	2194

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address--

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. ☒ This communication is responsive to *01/15/2010*.

2. ☒ The allowed claim(s) is/are *1 - 5 and 7 - 19*.

3. ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
   a) ☐ All b) ☐ Some* c) ☐ None of the:
      1. ☐ Certified copies of the priority documents have been received.
      2. ☐ Certified copies of the priority documents have been received in Application No. _____ .
      3. ☐ Copies of the certified copies of the priority documents have been received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

   * Certified copies not received: _____.

Applicant has THREE MONTHS FROM THE "MAILING DATE" of this communication to file a reply complying with the requirements noted below. Failure to timely comply will result in ABANDONMENT of this application.
THIS THREE-MONTH PERIOD IS NOT EXTENDABLE.

4. ☐ A SUBSTITUTE OATH OR DECLARATION must be submitted. Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL PATENT APPLICATION (PTO-152) which gives reason(s) why the oath or declaration is deficient.

5. ☐ CORRECTED DRAWINGS (as "replacement sheets") must be submitted.
   (a) ☐ including changes required by the Notice of Draftsperson's Patent Drawing Review (PTO-948) attached
      1) ☐ hereto or 2) ☐ to Paper No./Mail Date _____.
   (b) ☐ including changes required by the attached Examiner's Amendment / Comment or in the Office action of Paper No./Mail Date _____.

   Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the drawings in the front (not the back) of each sheet. Replacement sheet(s) should be labeled as such in the header according to 37 CFR 1.121(d).

6. ☐ DEPOSIT OF and/or INFORMATION about the deposit of BIOLOGICAL MATERIAL must be submitted. Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Attachment(s)

1. ☐ Notice of References Cited (PTO-892)
2. ☐ Notice of Draftperson's Patent Drawing Review (PTO-948)
3. ☐ Information Disclosure Statements (PTO/SB/08), Paper No./Mail Date _____
4. ☐ Examiner's Comment Regarding Requirement for Deposit of Biological Material
5. ☐ Notice of Informal Patent Application
6. ☒ Interview Summary (PTO-413), Paper No./Mail Date *20100423* .
7. ☒ Examiner's Amendment/Comment
8. ☒ Examiner's Statement of Reasons for Allowance
9. ☐ Other _____.

DETAILED ACTION

EXAMINER'S AMENDMENT

An examiner's amendment to the record appears below. Should the changes and/or additions be unacceptable to applicant, an amendment may be filed as provided by 37 CFR 1.312. To ensure consideration of such an amendment, it MUST be submitted no later than the payment of the issue fee.

Authorization for this examiner's amendment was given in a telephone interview David S. Carus on 04/23/2010.

Claim 1, (Currently Amended) A computing system for executing a plurality of software applications comprising:

a) a processor;

~~a~~ b) an operating system having an operating system kernel having OS critical system elements (OSCSEs) for running in kernel mode using said processor; and,

~~b~~ c) a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and

i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications, ~~and~~

ii) wherein an instance of a ~~an~~ SLCSE provided to at least a first ~~one or more~~ of the plurality of software applications from the shared library is run in a context of said at least first ~~one or more~~ of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second ~~one or more other~~ of the plurality of software applications running under the operating system have use of a unique instance of a corresponding critical system element for performing ~~essentially~~ same function, and

iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously.

Claim 3, (Currently Amended) A computing system according to claim 1 wherein OSCSEs corresponding to and capable of performing ~~essentially~~ the same function as SLCSEs remain in the operating system kernel.

Claim 6, (Canceled)

Claim 9, (Currently Amended) A computing system according to claim 8 ~~claim 7~~ wherein the up call mechanism in operation, executes instructions from an SLCSE resident in user mode space, in kernel mode.

Claim 19, (Currently Amended) A computer system as defined in claim 2 wherein SLCSEs are not copies of OSCSEs ~~OSLCEs~~.

Claim 20, (Canceled)

Reasons for Allowance

Claims 1 – 5 and 7 – 19 are allowed.

The following is an Examiner's statement of reasons for allowance. None of Elnozahy et al. (hereinafter Elnozahy) (US 7,499,966 B2) and Wong et al. (hereinafter Wong) (US 2004/0216145 A1) on the record discloses "a shared library having shared library critical system elements (SLCSEs) stored therein for use by the plurality of software applications in user mode and

i) wherein some of the SLCSEs stored in the shared library are functional replicas of OSCSEs and are accessible to some of the plurality of software applications and when one of the SLCSEs is accessed by one or more of the plurality of software applications it forms a part of the one or more of the plurality of software applications, and

ii) wherein an instance of a SLCSE provided to at least a first of the plurality of software applications from the shared library is run in a context of said at least first of the plurality of software applications without being shared with other of the plurality of software applications and where at least a second of the plurality of software applications running under the operating system have use of

a unique instance of a corresponding critical system element for performing same function.

iii) wherein a SLCSE related to a predetermined function is provided to the first of the plurality of software applications for running a first instance of the SLCSE, and wherein a SLCSE for performing a same function is provided to the second of the plurality of software applications for running a second instance of the SLCSE simultaneously" (Claim 1). Instead, Elnozahy discloses kernel extension device driver that are user space extensions of operating system code to minimize kernel calls by web server and Wong discloses user mode accessible copies of kernel-mode memory to facilitate a device driver to execute in user-mode while the graphics engine remains in kernel mode.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to SYED RONI whose telephone number is (571)270-7806. The examiner can normally be reached on M - F (8:30 am - 5:00 pm).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Hyung Sub Sough (Sam) can be reached on (571) 272 - 6799. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/SYED RONI/
Examiner, Art Unit 2194

/Hyung S. Sough/
Supervisory Patent Examiner, Art Unit 2194
04/26/10