(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 110914809 B
(45)授权公告日 2023.08.29

(21)申请号 201880048011.3
(22)申请日 2018.06.18
(65)同一申请的已公布的文献号
申请公布号 CN 110914809 A
(43)申请公布日 2020.03.24
(30)优先权数据
15/653,676 2017.07.19 US
(85)PCT国际申请进入国家阶段日
2020.01.17
(86)PCT国际申请的申请数据
PCT/IB2018/054458 2018.06.18
(87)PCT国际申请的公布数据
WO2019/016628 EN 2019.01.24
(73)专利权人 国际商业机器公司
地址 美国纽约阿芒克
(72)发明人 C•M•亚当 M•武科维奇
黄珍镐 S•纳德高达
N•亚尼罗西斯
(74)专利代理机构 北京市金杜律师事务所 11256
专利代理师 酆迅 辛鸣
(51)Int.Cl.
G06F 11/30(2006.01)
G06F 15/177(2006.01)
(56)对比文件
CN 104813331 A,2015.07.29
CN 105637833 A,2016.06.01
CN 1669033 A,2005.09.14
US 2006021035 A1,2006.01.26
US 2011112973 A1,2011.05.12
US 2013219156 A1,2013.08.22
US 2016373289 A1,2016.12.22
US 8090974 B1,2012.01.03
审查员 林松岭

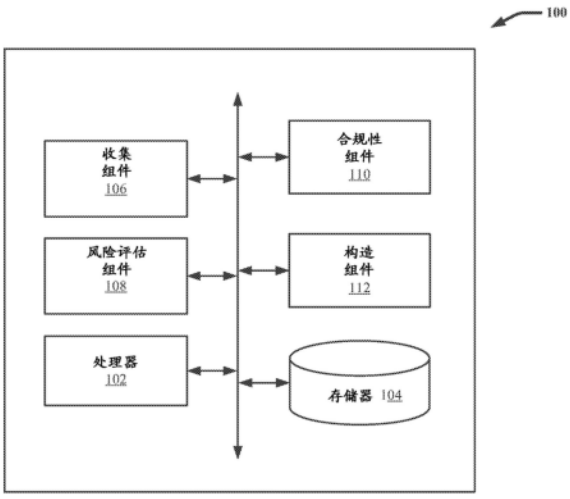
权利要求书2页 说明书16页 附图13页

(54)发明名称

基于应用程序模式和风险评估的合规感知的运行时生成

(57)摘要

提供了促进容器的合规感知的运行时生成的系统、计算机实现的方法和/或计算机程序产品。在一个实施例中，一种计算机实现的方法包括：由可操作地耦合到处理器的系统识别目标应用程序用于容器化的信息；确定在一个或多个已定义阈值内所述信息是否存在一个或多个风险违规；确定所述信息中是否存在合规性或安全性违规，其中，基于风险评估组件对不存在一个或多个风险违规性的确定，来执行由合规性组件进行的确定；以及生成与允许所述目标应用程序在没有底层操作系统的情况下执行的所述目标应用程序的定义组件对应的新容器，其中所述生成基于在所述信息中不存在合规性或安全性违规的确定。



CN 110914809 B

1.一种系统,包括:

存储计算机可执行组件的内存;

处理器,可操作地耦合到存储器,并执行存储在存储器中的计算机可执行组件,其中,计算机可执行组件包括:

收集组件,用于通过使用chroot和strace命令在单独的分区环境或单独的分区文件夹中运行目标应用程序来识别所述目标应用程序用于容器化的信息;

风险评估组件,用于确定在一个或多个已定义阈值内所述信息是否存在一个或多个风险违规;

合规性组件,其确定所述信息中是否存在合规性或安全性违规,其中,基于风险评估组件对不存在一个或多个风险违规性的确定,来执行由合规性组件进行的确定;和

构造组件,其基于在所述信息中不存在合规性或安全性违规的确定,生成与允许所述目标应用程序在没有底层操作系统的情况下执行的所述目标应用程序的定义组件对应的新容器。

2.根据权利要求1所述的系统,进一步包括自动将一个或多个特征映射到一个或多个系统操作或一个或多个库的映射组件。

3.根据权利要求2所述的系统,其中,一个或多个特征到一个或多个系统操作或一个或多个库的映射包括一个或多个合规性相关特征的映射。

4.根据权利要求2所述的系统,其中,一个或多个特征到一个或多个系统操作或一个或多个库的映射包括一个或多个安全性相关特征的映射。

5.根据权利要求1所述的系统,其进一步包含模式组件,所述模式组件生成用于自动创建一个或多个容器映像的一个或多个模式。

6.根据权利要求5所述的系统,其中,所述一个或多个模式的生成是基于风险评估组件对所述信息存在一个或多个风险违规的确定或基于合规性组件对所述信息中存在合规性或安全性违规的确定而执行的。

7.根据权利要求1所述的系统,其中,所述系统提供动态生成包含有助于服务的执行的目标代码的一个或多个运行时。

8.根据权利要求1所述的系统,进一步包含停止所述操作系统的过程的操作组件。

9.一种计算机实现的方法,包含:

由可操作地耦合到处理器的系统通过使用chroot和strace命令在单独的分区环境或单独的分区文件夹中运行目标应用程序来识别所述目标应用程序用于容器化的信息;

由所述系统确定在一个或多个已定义阈值内所述信息是否存在一个或多个风险违规;

由所述系统确定所述信息中是否存在合规性或安全性违规,其中,基于风险评估组件对不存在一个或多个风险违规性的确定,来执行由合规性组件进行的确定;和

由所述系统生成与允许所述目标应用程序在没有底层操作系统的情况下执行的所述目标应用程序的定义组件对应的新组件容器,其中所述生成基于在所述信息中不存在合规性或安全性违规的确定。

10.根据权利要求9所述的计算机实现的方法,进一步包括由所述系统自动将一个或多个特征映射到一个或多个系统操作或一个或多个库的映射组件。

11.根据权利要求10所述的计算机实现的方法,其中,所述自动将一个或多个特征到一

个或多个系统操作或一个或多个库的映射包括自动映射一个或多个合规性相关特征的映射。

12.根据权利要求10所述的计算机实现的方法,其中,所述自动将一个或多个特征到一个或多个系统操作或一个或多个库的映射包括自动映射一个或多个安全性相关特征的映射。

13.根据权利要求12所述的计算机实现的方法,仅进一步包括生成一个或多个模式,所述生成一个或多个模式基于风险评估组件对所述信息存在一个或多个风险违规的确定或基于合规性组件对所述信息中存在合规性或安全性违规的确定。

14.一种计算机可读存储介质,该计算机可读存储介质体现有程序指令,所述程序指令可由处理器执行,以使处理器:

通过使用chroot和strace命令在单独的分区环境或单独的分区文件夹中运行目标应用程序来识别所述目标应用程序用于容器化的信息;

确定在一个或多个已定义阈值内所述信息是否存在一个或多个风险违规;

确定所述信息中是否存在合规性或安全性违规,其中,对合规性或安全性违规的确定基于风险评估组件对不存在一个或多个风险违规性的确定;和

生成与允许所述目标应用程序在没有底层操作系统的情况下执行的所述目标应用程序的定义组件对应的新组件容器。

15.根据权利要求14所述的计算机可读存储介质,其中,所述新容器的生成基于在所述信息中不存在合规性或安全性违规的确定。

16.根据权利要求14所述的计算机可读存储介质,其中所述程序指令进一步可执行以使处理器:

自动映射一个或多个特征到一个或多个系统操作或一个或多个库的映射组件。

17.根据权利要求16所述的计算机可读存储介质,其中所述自动映射一个或多个特征到一个或多个系统操作或一个或多个库的映射包括映射一个或多个合规性相关特征。

18.根据权利要求17所述的计算机可读存储介质,其中所述自动映射一个或多个特征到一个或多个系统操作或一个或多个库的映射包括映射一个或多个安全性相关特征。

19.根据权利要求14所述的计算机可读存储介质,其中,所述程序指令进一步可执行以使处理器:

生成用来自动地创建一个或多个容器映像的一个或多个新模式。

20.根据权利要求19所述的计算机可读存储介质,其中,所述一个或多个新模式的生成是基于所述信息存在一个或多个风险违规的第一确定或基于所述信息中存在合规性或安全性违规的第二确定而执行的。

基于应用程序模式和风险评估的合规感知的运行时生成

技术领域

[0001] 本发明涉及促进容器的合规感知的运行时生成。

发明内容

[0002] 以下给出了提供对本发明的一个或多个实施例的基本理解的概述。该概述并非旨在标识关键或重要元素，也不旨在描绘本发明的特定实施例的任何范围或权利要求的任何范围。其唯一目的是以简化的形式呈现构思，作为稍后呈现的更详细描述的前序。在一个或多个实施例中，描述了促进容器的合规感知的运行时的生成的设备、系统、计算机实现的方法、装置和/或计算机程序产品。

[0003] 根据本发明的一个实施例，提供了一种系统。该系统可以包括存储计算机可执行组件的存储器。该系统还可以包括处理器，该处理器可操作地耦合到存储器，并且可以执行存储在存储器中的计算机可执行组件。计算机可执行组件可以包括收集组件，该收集组件可以识别目标应用程序用于容器化的信息。计算机可执行组件可以进一步包括风险评估组件，该风险评估组件可以确定在一个或多个已定义阈值内所述信息是否存在一个或多个风险违规。所述计算机可执行组件还可以包括合规性组件，其可以确定所述信息中是否存在合规性或安全性违规，其中，基于风险评估组件对不存在一个或多个风险违规性的确定，来执行由合规性组件进行的确定。计算机可执行组件还可包括构造组件，该构造组件基于在信息中不存在合规性或安全性违规的确定，生成与所述目标应用程序的定义组件相对应的、允许目标应用程序在没有底层操作系统的情况下执行的新容器。

[0004] 按照本发明另一个实施例，提供一种计算机实现的方法。该计算机实现的方法可包括由可操作地耦合到处理器的系统识别目标应用程序用于容器化的信息。该计算机实现的方法可进一步包括由所述系统确定在一个或多个已定义阈值内所述信息是否存在一个或多个风险违规。该计算机实现的方法可进一步包括由所述系统确定所述信息中是否存在合规性或安全性违规，其中，基于风险评估组件对不存在一个或多个风险违规性的确定，来执行由合规性组件进行的确定。该计算机实现的方法可进一步包括由所述系统生成与允许所述目标应用程序在没有底层操作系统的情况下执行的所述目标应用程序的定义组件对应的新组件容器，其中所述生成基于在所述信息中不存在合规性或安全性违规的确定。

[0005] 按照本发明另一个实施例，提供一种用于促进容器的合规感知的运行时生成的计算机程序产品。该计算机程序产品可包括计算机可读存储介质，该计算机可读存储介质具有利用其而被体现的程序指令。该程序指令可由处理器可执行以使得处理器识别目标应用程序用于容器化的信息。该程序指令进一步可由处理器可执行以使得处理器确定在一个或多个已定义阈值内所述信息是否存在一个或多个风险违规。该程序指令进一步可由处理器可执行以使得处理器确定所述信息中是否存在合规性或安全性违规，其中，关于合规性或安全性违规的确定基于风险评估组件对不存在一个或多个风险违规性的确定。该程序指令进一步可由处理器可执行以使得处理器生成与允许所述目标应用程序在没有底层操作系统的情况下执行的所述目标应用程序的定义组件对应的新组件容器。

附图说明

[0006] 图1是根据本发明的一个或多个实施例的促进容器的合规感知的运行时的生成的示例系统的框图。

[0007] 图2示出了根据本文描述的一个或多个实施例的包括映射组件的促进容器的合规感知的运行时的生成的示例性非限制性系统的框图。

[0008] 图3示出了根据本文描述的一个或多个实施例的包括模式组件的促进容器的合规感知的运行时的生成的示例性非限制性系统的框图。

[0009] 图4示出了根据本文描述的一个或多个实施例的包括操作组件的促进容器的合规感知的运行时的生成的示例性非限制性系统的框图。

[0010] 图5示出了根据本文所述的一个或多个实施例的促进容器的合规感知的运行时的生成的示例性非限制性计算机实现的方法。

[0011] 图6示出了根据本文描述的一个或多个实施例的促进合规性和安全性分类的示例性非限制性计算机实现的方法。

[0012] 图7、8、9和10示出了根据本文描述的一个或多个实施例的促进容器的合规感知的运行时的生成的示例性非限制性计算机实现的方法

[0013] 图11示出了本文所述的一个或多个实施例可在其中实现的示例非限制性操作环境的框图。

[0014] 图12示出了本文所述的一个或多个实施例可在其中实现的云计算环境。

[0015] 图13描绘了可以促进本文描述的一个或多个实施例的抽象模型层。

具体实施方式

[0016] 以下详细描述仅是说明性的，并不旨在限制本发明的实施例和/或本发明的实施例的应用或使用。此外，前面的发明背景或发明内容部分或下文的具体实施方式部分中提供的任何明示或暗示的信息不意味着对本发明的限制。

[0017] 现在参考附图描述本发明的一个或多个实施例，其中，贯穿全文的相似的参考标号用于指代相似的元件。在下面的描述中，出于解释的目的，阐述了许多具体细节以便提供对本发明的更透彻的理解。然而，很明显，在各种情况下，可以在没有这些具体细节的情况下实施本发明。

[0018] 在此描述的本发明的一个或多个实施例可以自动生成合规感知的运行时容器，其可以允许应用程序(例如，可执行文件、服务、服务实例、应用程序实例等)在没有底层操作系统的情况下运行。如本文所使用的，术语“应用程序”、“可执行文件”、“服务”、“服务实例”、“过程”和/或“应用程序实例”在本文中可以互换使用。

[0019] 图1示出了一个系统100的框图，该系统可以促进容器的合规感知的运行时生成。在本公开中解释的系统(例如，系统100等)、装置或过程的各方面可以构成体现在一个或多个机器中(例如，体现在与一个或多个机器关联的一个或多个计算机可读介质中)的一个或多个机器可执行组件。当由诸如计算机、计算设备、虚拟机等的一个或多个机器执行时，这样的组件可以使机器执行所描述的操作。

[0020] 在本发明的各种实施例中，系统100可以是包括处理器的任何类型的组件、机器、设备、设施、装置和/或仪器。在一些实施例中，系统100能够与有线和/或无线网络进行有效

和/或可操作的通信。可以包括系统100的组件、机器、装置、设备、设施和/或工具可以包括但不限于平板计算设备、手持设备、服务器类计算机器和/或数据库、膝上型计算机、笔记本计算机、台式计算机、蜂窝电话、智能电话、消费类设备和/或仪器、工业和/或商用设备、数字助理、支持多媒体因特网的电话、多媒体播放器等。

[0021] 如图1所示，系统100可以包括处理器102、存储器104、收集组件106、风险评估组件108、合规性组件110和/或构造组件112。在本发明的一些实施例中，处理器102、存储器104、收集组件106、风险评估组件108、合规性组件110和/或构造组件112的中的一个或多个可以在通信上和/或在操作上彼此耦合，以执行系统100的一项或多项功能。

[0022] 在系统100的一个或多个实施例中，预测分析可用于自动生成一个或多个合规感知的运行时容器。例如，自动生成可以基于模式知识库中保留的信息。如本文所使用的，术语“知识库”可以是可以存储一种或多种类型的信息的数据库或其他存储位置或存储库。本发明的所有这样的实施例都在设想内。

[0023] 模式知识库可以包括与一个或多个应用程序有关的信息。在本发明的一些实施例中，与一个或多个应用有关的信息可以随着时间的流逝而被收集并保留在模式知识库中。在一些实施例中，收集的信息可以包括用于目标应用程序的风险、合规性和/或安全违规。基于获得的信息，当启动应用程序时或之后，系统100可以评估模式知识库(或多个模式知识库)并生成一个或多个模式，和/或可以将关于目标应用程序的已知信息映射到关于其他应用程序的已知信息。系统100的预测分析可以确定，如果目标应用程序的信息与一个或多个其他应用程序相似，则可以利用相似应用程序的合规感知的运行时容器来自动执行目标应用程序。

[0024] 本文描述的计算机处理系统、计算机实现的方法、装置和/或计算机程序产品可以采用硬件和/或软件来生成本质上是高度技术性的、不是抽象的并且不能被人类作为一系列精神行为来执行的合规感知的运行时容器。例如，本发明的一个或多个实施例可以对可用信息进行冗长的解释和分析，以确定应将一个或多个运行时容器中的哪些运行时容器用于目标应用程序。在另一示例中，即使在没有关于目标应用程序的详细知识的情况下，本发明的一个或多个实施例也可以对大量数据执行预测分析，以自动促进具有高度准确性的容器的合规感知的运行时生成。可以通过将训练集与测试集进行比较来评估准确性。在采用训练集训练一个模型之后，可以通过计算由运行在与预测目标匹配的训练集元素上的模型生成的输出百分比，用测试集来计算准确性。

[0025] 在本发明的各种实施例中，收集组件106可以识别要部署的目标应用程序目标所采用的系统信息，例如但不限于动态库(例如，从属文件和库)和系统调用(例如，对所执行的服务的过程的请求)。例如，在现代的基于Unix的操作系统中，收集组件106可以执行列表动态依赖项(ldd)和strace命令来跟踪依赖项(例如，打开的文件或运行新应用程序启动时或启动之后进行的系统调用)并查找错误(例如版本不匹配、缺少文件或库等)。更具体地说，收集组件106可以用已在其内部部署的服务保留docker容器(docker container)中允许应用程序的部署的代码。例如，收集组件106可以从docker枢纽(docker hub)下载用于目标应用程序的docker映像。

[0026] 此外，在现代的基于Unix的操作系统中，收集组件106还可以使用chroot和strace命令在单独的分区环境或单独的分区文件夹中运行目标应用程序，以捕获所有的(或者，在

某些实施例中，一个或多个更多)正在运行的用于发现错误的代码。收集组件106可以通过将用于运行该应用程序的文件存放在一个文件夹中并执行chroot命令，在单独的分区环境中运行该应用程序。chroot命令可以将根文件系统切换到单独的分区目录。如果由于某个错误指示缺少文件或库而在执行chroot和strace命令后某个操作有风险，那么如果可以找到缺失文件，则收集组件106可以从docker映像中复制缺失的文件或库。通过将用于运行已经安装在操作系统上的应用程序实例的文件和库复制和/或存放到单独的分区文件夹中，系统100可以迭代地进行该过程，直到目标应用程序在单独的分区文件夹中运行。

[0027] 本发明的一个或多个实施例可以大幅度减少通常为了符合合规性准则和/或策略而使用的正在运行的文件的数量。随着文件数量的减少，漏洞问题也可以减少。由于映像大小的减小，在一个或多个实施例中，可以减少网络流量。另外，由于减少了映像占用空间，应用程序部署和/或启动也可以更快。

[0028] 应用程序通常分为两类：可以直接加载一组系统库的可执行文件和使用语言运行时(language runtime)的应用程序。对于具有可以直接加载一组系统库的可执行文件的应用程序，上述计算机实现的方法可以生成合规感知的运行时容器。这些无需运行时即可直接运行的应用程序的示例包括但不限于nginx、MySQL、Redis等。对于使用语言运行时的应用程序(例如Ruby、Python、Java、Erlang等)，上述计算机实现的方法还可以构建运行时映像，并且其他逻辑可以确定可以安装在运行时映像上的特定于语言的包(例如Ruby Gems、Python Wheels、Java JAR等)。

[0029] 在本发明的一些实施例中，系统100可以通过将已经安装在操作系统上的应用程序作为输入来生成合规感知的运行时容器。为服务产生的运行时环境可以包含服务用于部署的代码，而没有其他内容。可以评估合规感知的运行时容器中包括的每个文件和库(和/或在一些实施例中，至少一个或多个文件和/或库)的脆弱性和/或合规性。可以在创建容器映像时或之后适当评估风险，以使容器不包含潜在的易受攻击的库。例如，风险评估组件108可以捕获在应用程序在其本机环境中运行时打开和/或访问的文件和/或库，并分析文件和/或库中的漏洞，以及分析配置文件的合规性。在本发明的一些实施例中，风险评估组件108可以对应用程序属性执行动态和/或静态分析，以确定是否将额外的信息(例如文件、库、代码等)用于容器来部署目标应用程序。风险评估组件108可以分析并确定调整的可行性。例如，风险评估组件108可以确定与在具有减少的文件和库的集合的生成的容器中运行应用程序相关的一个或多个问题。

[0030] 风险评估组件108可以确定是否在一个或多个定义的阈值内信息存在一个或多个风险违规。例如，可以采用风险或影响函数来衡量执行操作或部署服务的风险量。如果风险值等于或大于特定定义的阈值，则可能指示系统100可能无法运行目标应用程序。可以评估由风险评估组件108捕获的信息以提供在生成容器时可以利用的风险分析和/或模式形成。如果风险评估组件108指示某个操作具有大于或等于定义的阈值的伴随风险，则可以在构造新容器之前复制该应用程序和/或模拟一个或多个操作。例如，如果某个操作的伴随风险大于或等于定义的阈值，则在执行chroot和strace命令(例如，通过收集组件106)后，由于错误指示文件或库丢失，则如果缺失的文件或库在docker映像中，则系统100可以从docker映像中复制缺失的文件或库。在目标应用程序可以在单独的分区文件夹中运行之前，这可能是一个反复的过程。

[0031] 在本发明的一些实施例中，合规性组件110可以确定所述信息中是否存在合规性或安全性违规。合规性组件110的确定可以基于风险评估组件108的确定不存在一个或多个定义的风险违规。例如，当风险评估组件108可以确定操作具有不大于且不等于定义的阈值的伴随风险时(因此该操作被认为是无风险的)，合规组件110可以通过采用主动学习算法确定是否存在合规性或安全性违规。如果发现合规性或安全性违规，则可以将该信息存储在模式知识库中。可以通过分析一个或多个应用程序的两种模式以及未发现的安全问题来执行该学习。可以根据需要为给定的应用程序创建两个以上的不同容器。通过使用当前活动来训练学习算法(例如，支持向量机(SVM))中的数据，可以通过分析差异，模式能被学习和/或能被采用，以来加快合规性和/或安全性评估。此外，系统100的机器学习过程可用于分析生成合规感知的运行时容器的步骤并加快该过程。例如，机器学习过程可以识别模式和/或可以评估一段代码是冗余的、不必要的还是不相关的。可以采用模式和/或模式识别来使收集组件106收集系统信息的过程更有效，从而节省资源和时间。如果不存在合规性或安全性违规，则可以生成一个合规感知的运行时容器。

[0032] 在一些实施例中，构造组件112可以生成允许目标应用程序在没有底层操作系统的情况下执行的新容器。合规感知的运行时容器的生成可以基于在信息(例如，用于运行目标应用程序的文件、库、代码等)中不存在合规性或安全性违规的确定。容器中缺少操作系统可以消除对执行合规性评估(例如密码管理和操作系统资源)的需要。这还可以自动阻止用户在容器上安装其他服务，例如安全套接字壳(SSH)—其可能使容器面临其他潜在的安全漏洞。

[0033] 更具体地，构造组件112可以生成容器，并且在一些实施例中，可以仅包括由容器用来运行特定服务或应用程序的文件和库。在(例如，通过风险评估组件108和合规性组件110)确定不存在合规性或安全性违规并且(例如，通过收集组件106)目标应用程序在单独的分区文件夹中运行之后，构建组件112可以将用于运行应用程序的文件和库复制到dockerfile中。因此，在本发明的一些实施例中，构造组件112可以利用dockerfile来生成具有合规性保证和最小的安全风险的容器映像。用于生成这些合规保证的容器映像的相同组件和/或过程可以用于所有运行中的容器，以评估风险、合规性和/或安全违规。

[0034] 在本发明的一些实施例中，这些合规感知的运行时生成的容器也是不可变的。在一些实施例中，因为没有壳(shells)，所以不能输入和/或修改不可变容器。这样，最初部署的映像可以是将要运行的映像，并且没有简便的方法来更改此不可变的、合规感知的运行时容器的结构。这些生成的容器可以使应用程序无需底层操作系统就能运行。没有底层操作系统的容器还伴随有降低的攻击面风险，因为没有底层操作系统，要修改(例如，攻击)的文件和库就更少。较小的攻击面又可以减少合规成本并增强安全性。通过将重点转移到应用程序本身的合规性上并消除配置和保护复杂的底层操作系统的成本，结果可以大大降低运营服务的成本。

[0035] 图2示出了促进包括映射组件202的容器的合规感知的运行时生成的系统200的框图。系统200还可以包括映射组件202。映射组件202可以将一个或多个特征自动地映射到一个或多个系统操作或库。该一个或多个系统操作或库的映射可以包括一个或多个与合规性和安全性相关的特征的映射。例如，映射组件202可以映射与学习到的由风险评估组件108和合规性组件110评估的合规感知的运行时相关联的系统信息。

[0036] 图3是促进包括模式组件的容器的合规感知的运行时生成的系统300的框图。模式组件302可以生成一种或多种模式，用于自动构建一个或多个合规感知的运行时容器映像。可以基于风险评估组件108对信息存在一个或多个风险违规的确定和/或基于合规性组件110对信息中存在合规性或安全性违规的确定，来执行一个或多个模式的生成。例如，如果风险评估组件108确定某操作具有等于或大于定义的阈值的风险值，则模式组件302可以生成可用来构造一个或多个容器映像的一个或多个模式。此外，如果合规性组件110确定信息中存在合规性或安全性违规，则模式组件302可以生成用来构造一个或多个容器映像的一个或多个模式。

[0037] 图4是促进包括操作组件402的容器的合规感知的运行时生成的系统400的框图。操作组件402可以停止操作系统的过程。在一些实施例中，操作组件402可以基于由模式组件302生成的模式来确定何时停止操作系统的过程。参考由模式组件302生成的模式—其可以基于(例如，通过风险评估组件108)对存在一个或多个风险违规的确定或者基于(例如，通过合规性组件110)对信息中存在合规性和安全违规的确定而生成，如果存在合规性或安全性违规，则可以停止操作系统。更具体地，操作组件402可以访问由模式组件302分析的风险、合规性和/或安全性违规信息，以警告用户某操作具有的伴随风险大于或等于定义的阈值，具有合规性违规或具有安全性违规。

[0038] 系统100、200、300和/或400的组件可以在一起相互通信以生成合规感知的容器和/或收集通过该过程学习的信息以供将来使用。收集组件106可以识别目标应用程序使用的信息，并且该信息可以由风险评估组件108和合规性组件110用于评估与风险、合规性和安全性有关的问题。为了应用程序属性的动态和静态分析，可以连续执行风险、合规性和/或安全性评估。由风险组件108和合规性组件110分析的信息可以由映射组件202用来生成映射图(maps)，并且可以由模式组件302用来生成模式。操作组件402可以采用基于风险\合规性和安全违规而生成的模式来停止操作系统的过程。

[0039] 图5示出了促进容器的合规感知的运行时生成的计算机实现的方法500。方法500使用nginx作为可直接加载一组系统库的可执行文件或无需运行时即可直接运行的可执行文件的示例。在510，方法500可以包括(例如，经由收集组件106)识别要容器化的目标应用程序。

[0040] 在520，方法500可以包括(例如，经由收集组件106)收集诸如用于目标应用程序的动态库和系统调用的系统信息。可以通过执行ldd命令以确定可执行文件的直接依赖项，来确定系统信息。该识别还可以包括从Docker枢纽下载目标应用程序的docker映像，并执行chroot和strace命令以捕获正在运行的所有代码以查找错误。这个过程可以通过通过执行chroot命令创建一个可以在其中运行可执行文件的单独的分区环境或单独的分区文件夹来完成。在单独的分区文件夹中运行的可执行文件将就像它是文件系统的根目录一样来响应。如果在执行chroot和strace命令后，错误信息指示文件或库缺失，则从docker映像中复制缺失的文件或库—如果缺失的文件或库在那里的话。这可以是一个迭代过程，可以一直执行，直到可执行文件可以在单独的分区文件夹中运行。结果是一个单独的分区文件夹，其仅具有用于运行代码的一组文件和库而无其它。在所使用的文件或库在单独的分区文件夹内运行的时刻，方法500可包括将用于运行目标应用程序的文件和库复制到新dockerfile中。该新dockerfile可被用来生成具有合规感知的运行时的新docker映像。在530，方法500

可包括(例如,经由风险评估组件108)对库和系统调用执行风险评估。可以用一个风险或影响函数来评估用减少的文件和库集来运行应用程序的潜在风险。

[0041] 样本风险或影响函数的非限制性示例可以是$R(\theta,\delta)=E_{\theta}L(\theta,\delta(X))=\int_{X}L(\theta,\delta(X))dP_{\theta}(X)$,其中风险函数R是从0到1变化的值,θ是固定值,并且可能是未知的自然状态。X是从例如先前的应用程序或服务器配置文件、一系列相关的升级操作、版本,可用的服务管理功能等的总体中随机抽取的观察值的向量。E_{θ}是对X的总体值的期望。dP_{θ}是用θ参数化的事件空间X上的概率度量,该积分在X的整个支持范围内被评估。如果风险值超过定义的阈值,则可以复制应用程序和/或可以模拟操作以进行安全检查或风险分析。

[0042] 在540,方法500可以包括(例如,经由风险评估组件108)确定是否发现了风险违规。可以通过将风险值与定义的阈值进行比较来确定风险分析。如果发现了风险违规,则在542,方法500可以包括(例如,经由风险评估组件)在模式知识库中输入风险违规信息。如果未发现风险违规,则在550,方法500可以包括(例如,经由合规性组件110)对库和系统调用执行合规性和安全性评估。可以通过采用如在下面的图6中详细描述的主动学习算法来确定合规性和安全性。然后,在560,方法500可以包括(例如,经由合规性组件110)确定是否以预定阈值找到了合规性和安全性违规。如果找到合规性和安全性违规,则在542,计算机实现的方法500可以包括(例如,经由合规性组件110)在模式知识库中输入合规性和安全性违规信息。如果未找到合规性和安全性违规,则在570,方法500可以包括(例如,经由构造组件112)生成合规感知的容器。然后在580,方法500可以包括执行部署测试。

[0043] (例如,与自动评估合规性和安全违规有关的)各个方面可以采用各种基于人工智能的方案来执行其各个方面。例如,可以利用用于评估目标应用程序的一个或多个参数的过程来预测对评估的一个或多个响应,而无需来自目标应用程序的交互,这可以通过主动学习算法来实现。支持向量机(SVM)是可以使用的分类器的示例。SVM通过在可能的输入空间中找到一个超曲面来进行操作,该超曲面试图将触发标准与非触发事件分开。凭直觉,这使分类适用于可能与训练数据相似但不一定相同的测试数据。可以采用提供不同独立性模式的其他有向和无向模型分类方法(例如朴素贝叶斯、贝叶斯网络、决策树、神经网络、模糊逻辑模型和概率分类模型)。如本文所使用的分类可以包括用于开发优先级模型的统计回归。

[0044] 图6示出了促进合规性和安全性分类的计算机实现的方法600。方法600可以是促进合规性和安全性分类的基于多标签SVM的主动学习算法。在本发明的一些实施例中,方法600每次迭代可以将标记集D_l、未标记集D_u、多个步骤T以及多个示例作为输入。方法600可以基于训练数据D_l来训练多标签SVM分类器f。例如,对于未标记集合D_u中的每个实例x,方法600可以使用基于损失减少(LR)的预测方法来预测其标签矢量y,该预测方法具有用于以最大置信度进行最大损失减少的方程。方法600可以用最确定的标签矢量y来计算预期的损失减少;对D_u中所有x的得分x降序排序,并选择一组得分最高的示例。该选择可以基于主题专家(SME)的输入。函数$f_i(x)$是与类i相关的SVM分类器。数据点$x_1,\cdots x_n$表示每个x(库、系统调用、操作系统信息、配置信息、应用程序信息等)的特征向量以及合规性和安全性兼容性。

[0045] 图7示出了计算机实现的方法700。在702,方法700可以包括:由可操作地耦合至处理器的系统(例如,经由收集组件)识别由目标应用程序用来进行容器化的信息。可以执行ldd命令来确定立即或初始依赖项。可以执行chroot和strace命令以捕获和分析正在运行

的代码以查找错误。在704，方法700可以包括由该系统(例如，经由风险评估组件108)确定在一个或多个定义的阈值内所述信息是否存在一个或多个风险违规。可以通过采用风险或影响函数来执行风险分析。如果风险值大于阈值，则可以替换服务实例。在706，方法700可以包括由该系统(例如，通过合规性组件110)确定所述信息中是否存在合规性或安全性违规，其中基于风险评估组件对不存在一个或多个风险违规的确定来确定是否存在合规性或安全性违规。可以将基于多标签SVM的学习算法应用于捕获合规性和安全性相关的问题。在708，方法700可以包括由该系统(例如，通过构建组件112)生成与目标应用程序的定义的组件相对应的组件的新容器，该新容器允许目标应用程序在没有底层操作系统的情况下执行，其中，该生成基于的是对信息中不存在安全合规性和安全性违规的确定。合规感知的运行时生成的容器可以允许目标应用程序在没有底层操作系统的情况下执行。在本发明的一些实施例中，该容器不具有壳，从而也使其成为不可变的容器。

[0046] 图8示出了计算机实现的方法800。方法800被提供用于被分类为可以直接加载一组系统库的可执行文件的示例应用程序。这里的示例程序利用了nginx。在802，方法800可以包括(例如，经由收集组件106)创建用于运行应用程序的单独的分区目录和其他目录。在804，方法800可以包括(例如，经由收集组件106)创建执行chroot命令所需的目录。在806，方法800可以包括(例如，经由收集组件106)从docker库中下载最新映像并在本地运行。在808，方法800可以包括(例如，经由收集组件106)在docker文件中找到用于运行容器的命令。在810，方法800可以包括(例如，经由收集组件106)使用ldd命令来建立可执行文件的初始依赖项。在812，方法800可以包括(例如，经由收集组件106)确定在启动应用程序时是否遇到缺失文件的错误。如果是，则方法800可包括重复步骤812的以下步骤814、816、818和820。如果否，则方法800可包括以下步骤822、824和826以完成并结束该过程。如果遇到缺失文件的错误，则在814对于找到的每个依赖项(或者，在一些实施例中，对于找到的一个或多个依赖项)，方法800可以包括(例如，通过收集组件106)在单独的分区中创建其目录，并在单独的分区中从容器复制库。在816，方法800可以包括(例如，通过收集组件106)对不是库、但仍为应用程序所需要的文件，诸如配置文件、证书文件、安全套接字壳(SSH)密钥等的文件，执行与上述相同的操作。在818，方法800可以包括(例如，通过收集组件106)使用chroot和strace命令在单独的分区目录中启动应用程序，并且解析strace输出以找到任何缺失文件或库。在820，方法800可以包括(例如，经由收集组件106)在docker容器中定位缺失文件和库。如果在容器中找到文件，则方法800可以包括将文件添加到依赖项列表以复制到单独的分区目录中。然后，方法800可以包括重复步骤812、814、816、818和820，直到在启动应用程序时没有遇到缺失文件的错误为止。如果在启动应用程序时没有遇到缺失文件的错误，则在822，方法800可以包括(例如，经由收集组件106)从头开始生成新的dockerfile，并在映像根目录中复制单独的分区目录的内容。在824，方法800可以包括(例如，经由收集组件106)解析原始dockerfile中如暴露端口、定义卷之类的其它命令，并将它们添加到新的dockerfile中。在826，方法800可以包括：在以稳定状态运行容器之前，运行用于初始化或配置(例如，初始化、迁移和/或播种应用程序的数据库)的任何其他命令。

[0047] 图9和图10示出了示例方法900和1000。方法900和1000包括使用语言运行时(language runtime)的示例应用。对于这些类型的应用程序，采用略有不同的过程来确保或提高解释器可以在容器中运行的可能性，并且应用程序具有所有(或在某些实施例中，一

个或多个)用于部署的文件和/或库。语言运行时开始运行后,可以部署应用程序。Python应用程序通常具有一个需求文件,其具有可安装的pythonwheel和包(package)。Python通常具有自己的安装程序,例如pip。这些特定于应用程序的本地包可以被安装在合规感知的运行时容器上。其他使用语言运行时的应用程序也可以具有类似的过程。对于Ruby,可以使用gem而不是包,并且Ruby也可以采用构建Ruby运行时。通过在应用程序运行时环境中运行gem列表(list)命令,可以获得应用程序使用的gemfile中列出的gem的完整集合。对于Java,可以使用jar代替gem或包。

[0048] 图9示出了方法900。计算机实现的方法900是python运行时应用程序的示例。在910,方法900可以包括(例如,经由收集组件106)为python运行python应用程序而构建合规感知的运行时。在912,方法900可以包括(例如,经由收集组件106)确定在系统上可用的所有(或者在本发明的一些实施例中,一个或多个)python包的列表。在914,方法900可以包括(例如,经由收集组件106)确定由目标应用程序访问的所有(或在本发明的一些实施例中,一个或多个)python包的列表。在916,方法900可以包括(例如,经由收集组件106)确定是否指定了一个requirements.txt文件。如果是(指定了requirements.txt文件),则在918,方法900可以包括(例如,经由收集组件106)解析该需求(requirement)文件以确定python包依赖项的列表。然后在920,方法900可以包括(例如,经由收集组件106)将应用程序脚本复制到单独的分区目录。此外,在922,方法900可以包括(例如,经由收集组件106)将定义的(或者,在本发明的一些实施例中,采用的或要求的)python包复制到单独的分区目录。如果未指定requirements.txt文件,则在924,计算机实现的方法900可以包括(例如,经由收集组件106)获取用于该应用程序的.py文件的列表。然后在926,方法900可包括(例如,经由收集组件106)从.py文件解析导入语句。在928,方法900可以包括(例如,经由收集组件106)使用pyflakes和pylint工具来移除任何未使用的库。在930,方法900可以包括(例如,经由收集组件106)收集应用程序使用的所有唯一性python库。例如,可以通过在应用程序运行时环境中运行pip freeze命令来获取应用程序使用的python包的完整集合。

[0049] 图10示出了方法1000。图10示出了用于Java运行时应用程序的示例过程1000。在1002,方法1000可以包括(例如,经由收集组件106)为诸如Java运行时环境(JRE)之类的Java虚拟机(JVM)建立最小运行时(minimum runtime)以运行Java应用程序。在1004,方法1000可以包括(例如,经由收集组件106)确定系统上可用的所有java库的列表。在1006,方法1000可以包括(例如,经由收集组件106)读取项目构建文件以找到绑定到应用程序的所有库。在1008,方法1000可以包括(例如,经由收集组件106)将.jar文件复制到单独的分区目录。

[0050] 为了提供所公开主题的各个方面的背景,图11以及以下讨论旨在提供对其中可以实现所公开的主题的各个方面的合适环境的一般描述。图11示出了其中可以促进本发明的一个或多个实施例的示例性非限制性操作环境的框图。

[0051] 参照图11,用于实现本发明的各个方面的合适的操作环境1100还可以包括计算机1112。计算机1112还可以包括处理单元1114、系统存储器1116和系统总线1118。系统总线1118耦合系统组件—包括但不限于—将系统存储器1116耦合到处理单元1114。处理单元1114可以是各种可用处理器中的任何一种。也可采用双微处理器和其他多处理器架构来作为处理单元1114。系统总线1118可以是几种类型的总线结构中的任何一种,包括存储器总

线或存储器控制器、外围总线或外部总线、和/或使用任何可用总线架构的本地总线，这些总线架构包括但不限于工业标准架构(ISA)、微通道架构(MSA)、扩展ISA(EISA)、智能驱动电子(IDE)、VESA本地总线(VLB)、外围组件互连(PCI)、卡总线、通用串行总线(USB)、高级图形端口(AGP)、火线(IEEE 1394)和小型计算机系统接口(SCSI)。

[0052] 系统存储器1116还可以包括易失性存储器1120和非易失性存储器1122。基本输入/输出系统(BIOS)包含诸如在启动期间在计算机1112内各元素之间传递信息的基本例程。作为示例而非限制，非易失性存储器1122可以包括只读存储器(ROM)、可编程ROM(PROM)、电可编程ROM(EPROM)、电可擦除可编程ROM(EEPROM)、闪存。易失性存储器1120还可以包括用作外部高速缓冲存储器的随机存取存储器(RAM)。例如但非限制，RAM有各种可用的形式，诸如静态RAM(SRAM)、动态RAM(DRAM)、同步DRAM(SDRAM)、双倍数据速率SDRAM(DDR SDRAM)、增强型SDRAM(ESDRAM)、同步链接DRAM(SLDRAM)、直接Rambus RAM(DRRAM)、直接Rambus动态RAM(DRDRA M)和Rambus动态RAM。

[0053] 计算机1112还可以包括可移动/不可移动、易失性/非易失性计算机存储介质。图11示出了例如磁盘存储器1124。磁盘存储器1124还可以包括但不限于诸如磁盘驱动器、软盘驱动器、磁带驱动器、Jaz驱动器、Zip驱动器、LS-100驱动器、闪存、存储卡或记忆棒等的设备。磁盘存储器1124还可以单独地或与其他存储介质结合地使用的存储介质，所述其他介质包括但不限于诸如光盘ROM设备(CD-ROM)、可记录CD的驱动器(CD-R Drive)、CD可擦写驱动器(CD-RW驱动器)或数字多功能磁盘ROM驱动器(DVD-ROM)之类的光盘驱动器。为了促进磁盘存储器1124到系统总线1118的连接，通常使用可移动或不可移动接口，例如接口1126。图11还描绘了充当用户与在适当的操作环境1100中描述的基本计算机资源之间的中介的软件。这样的软件还可以包括例如操作系统1128。操作系统1128可以存储在磁盘存储器1124上，用于控制和分配计算机1112的资源。

[0054] 系统应用程序1130利用操作系统1128通过(例如，存储在系统存储器1116或磁盘存储1124中的)程序模块1132和程序数据1134对资源的管理。应该理解，本公开可以用各种操作系统或操作系统组合来实现。用户通过输入设备1136向计算机1112输入命令或信息。输入设备1136包括但不限于诸如鼠标、轨迹球、触控笔、触摸板、键盘、麦克风、操纵杆、游戏垫、卫星天线、扫描仪、电视调谐卡、数码相机、数码摄像机、网络摄像机等指示设备。这些和其他输入设备经由接口端口1138通过系统总线1118连接到处理单元1114。接口端口1138包括例如串行端口、并行端口、游戏端口和通用串行总线(USB)。输出设备1140使用一些与输入设备1136相同类型的端口。因此，例如，USB端口可用于向计算机1112提供输入，并将信息从计算机1112输出到输出设备1140。提供输出适配器1142以说明存在一些需要特殊的适配器的输出设备1140，例如监视器、扬声器和打印机，以及其他输出设备1140。作为说明而非限制，输出适配器1142包括视频和声卡，其提供了在输出设备1140和系统总线1118之间的连接手段。应当注意，其他设备和/或设备系统同时提供输入和输出功能，例如远程计算机1144。

[0055] 计算机1112可以在使用到一个或多个远程计算机(例如一个或多个远程计算机1144)的逻辑连接的联网环境中操作。一个或多个远程计算机1144可以是计算机、服务器、路由器、网络PC、工作站、基于微处理器的设备、对等设备或其他公共网络节点等，并且通常还可以包括相对于计算机1112描述的许多或所有元素。为了简洁起见，仅与远程计算机

1144一起示出内存存储器1146。远程计算机1144通过网络接口1148逻辑地连接到计算机1112，然后经由通信连接1150物理地连接。网络接口1148包括有线和/或无线通信网络，诸如局域网(LAN)、广域网(WAN)、蜂窝网络等。LAN技术包括光纤分布式数据接口(FDDI)、铜缆分布式数据接口(CDDI)、以太网、令牌环等等。WAN技术包括但不限于点对点链接、电路交换网络(如集成服务数字网络(ISDN))及其上的变体、分组交换网络和数字用户线(DSL)。通信连接1150是指用来将网络接口1148连接到系统总线1118的硬件/软件。尽管为了说明清楚起见，在计算机1112内部示出了通信连接1150，但是它也可以在计算机1112的外部。仅用于示例性目的，用于连接到网络接口1148的软件还可以包括内部和外部技术，诸如包括常规电话级调制解调器、电缆调制解调器和DSL调制解调器的调制解调器，ISDN适配器和以太网卡。

[0056] 首先应当理解，尽管本公开包括关于云计算的详细描述，但其中记载的技术方案的实现却不限于云计算环境，而是能够结合现在已知或以后开发的任何其它类型的计算环境而实现。

[0057] 云计算是一种服务交付模式，用于对共享的可配置计算资源池进行方便、按需的网络访问。可配置计算资源是能够以最小的管理成本或与服务提供者进行最少的交互就能快速部署和释放的资源，例如可以是网络、网络带宽、服务器、处理、内存、存储、应用、虚拟机和服务。这种云模式可以包括至少五个特征、至少三个服务模型和至少四个部署模型。

[0058] 特征包括：

[0059] 按需自助式服务：云的消费者在无需与服务提供者进行人为交互的情况下能够单方面自动地按需部署诸如服务器时间和网络存储等的计算能力。

[0060] 广泛的网络接入：计算能力可以通过标准机制在网络上获取，这种标准机制促进了通过不同种类的瘦客户机平台或厚客户机平台(例如移动电话、膝上型电脑、个人数字助理PDA)对云的使用。

[0061] 资源池：提供者的计算资源被归入资源池并通过多租户(multi-tenant)模式服务于多重消费者，其中按需将不同的实体资源和虚拟资源动态地分配和再分配。一般情况下，消费者不能控制或甚至并不知晓所提供的资源的确切位置，但可以在较高抽象程度上指定位置(例如国家、州或数据中心)，因此具有位置无关性。

[0062] 迅速弹性：能够迅速、有弹性地(有时是自动地)部署计算能力，以实现快速扩展，并且能迅速释放来快速缩小。在消费者看来，用于部署的可用计算能力往往显得是无限的，并能在任意时候都能获取任意数量的计算能力。

[0063] 可测量的服务：云系统通过利用适于服务类型(例如存储、处理、带宽和活跃用户帐号)的某种抽象程度的计量能力，自动地控制和优化资源效用。可以监测、控制和报告资源使用情况，为服务提供者和消费者双方提供透明度。

[0064] 服务模型如下：

[0065] 软件即服务(SaaS)：向消费者提供的能力是使用提供者在云基础架构上运行的应用。可以通过诸如网络浏览器的瘦客户机接口(例如基于网络的电子邮件)从各种客户机设备访问应用。除了有限的特定于用户的应用配置设置外，消费者既不管理也不控制包括网络、服务器、操作系统、存储、乃至单个应用能力等的底层云基础架构。

[0066] 平台即服务(PaaS)：向消费者提供的能力是在云基础架构上部署消费者创建或获

得的应用，这些应用利用提供者支持的程序设计语言和工具创建。消费者既不管理也不控制包括网络、服务器、操作系统或存储的底层云基础架构，但对其部署的应用具有控制权，对应用托管环境配置可能也具有控制权。

[0067] 基础架构即服务(IaaS)：向消费者提供的能力是消费者能够在其中部署并运行包括操作系统和应用的任意软件的处理、存储、网络和其他基础计算资源。消费者既不管理也不控制底层的云基础架构，但是对操作系统、存储和其部署的应用具有控制权，对选择的网络组件(例如主机防火墙)可能具有有限的控制权。

[0068] 部署模型如下：

[0069] 私有云：云基础架构单独为某个组织运行。云基础架构可以由该组织或第三方管理并且可以存在于该组织内部或外部。

[0070] 共同体云：云基础架构被若干组织共享并支持有共同利害关系(例如任务使命、安全要求、政策和合规考虑)的特定共同体。共同体云可以由共同体内的多个组织或第三方管理并且可以存在于该共同体内部或外部。

[0071] 公共云：云基础架构向公众或大型产业群提供并由出售云服务的组织拥有。

[0072] 混合云：云基础架构由两个或更多部署模型的云(私有云、共同体云或公共云)组成，这些云依然是独特的实体，但是通过使数据和应用能够移植的标准化技术或私有技术(例如用于云之间的负载平衡的云突发流量分担技术)绑定在一起。

[0073] 云计算环境是面向服务的，特点集中在无状态性、低耦合性、模块性和语意的互操作性。云计算的核心是包含互连节点网络的基础架构。

[0074] 现在参照图12，描绘了说明性的云计算环境1250。如图所示，云计算环境1250包括一个或多个云计算节点1210，由云消费者使用的本地计算设备，诸如个人数字助理(PDA)或蜂窝电话1254A、台式计算机1254B、膝上型计算机1254C、和/或汽车计算机系统1254N，可以与云计算节点通信。节点1210可以彼此通信。可以在一个或多个网络(如上文所述的私有云、共同体云、公共云或混合云)或其组合中对它们进行物理或虚拟分组(未显示)。这允许云计算环境1250提供基础设施、平台和/或软件即服务，云消费者不需要为其维护本地计算设备上的资源。应该理解的是，图12中所示的计算设备1254A-N的类型仅仅是示例性的，计算节点1210和云计算环境1250可以通过任何类型的网络和/或网络可寻址连接(例如，使用网络浏览器)与任何类型的计算机化设备通信。

[0075] 现在参照图13，示出了由云计算环境1250(图12)提供的一组功能抽象层。应当预先理解，图13中所示的组件、层和功能仅仅是示例性的，本发明的实施例不限于此。如图所示，提供了以下的层和相应的功能：

[0076] 硬件和软件层1360包括硬件和软件组件。硬件组件的示例包括：大型机1361；基于RISC(精简指令集计算机)体系结构的服务器1362；服务器1363；刀片服务器1364；存储设备1365；以及网络和联网组件1366。在一些实施例中，软件组件包括网络应用服务器软件1367和数据库软件1368。

[0077] 虚拟化层1370提供抽象层，从该抽象层可以提供虚拟实体的以下示例：虚拟服务器1371；虚拟存储器1372；虚拟网络1373，包括虚拟专用网络；虚拟应用程序和操作系统1374；以及虚拟客户端1375。

[0078] 在一个示例中，管理层1380可提供以下描述的功能。资源供应1381提供对计算资

源和用于在云计算环境内执行任务的其他资源的动态采购。当在云计算环境中利用资源时，计量和定价1382提供成本跟踪，并为这些资源的消耗开具帐单或发票。在一个示例中，这些资源可以包括应用软件许可证。安全性为云使用者和任务提供身份验证，并为数据和其他资源提供保护。用户门户1383为消费者和系统管理员提供对云计算环境的访问。服务级别管理1384提供云计算资源分配和管理，以便满足所需的服务水平。服务水平协议(SLA)规划和实现1385按照SLA提供对预期未来需求的云计算资源的预先安排和采购。

[0079] 工作负载层1390提供可以利用云计算环境的功能的示例。可以从该层提供的工作负载和功能的非限制性示例包括：映射和导航1391；软件开发和生命周期管理1392；虚拟课堂教学交付1393；数据分析处理1394；交易处理1395；和交易模型软件1396。

[0080] 本发明可以是处于任何可能的技术细节集成水平的系统，方法，装置和/或计算机程序产品。该计算机程序产品可以包括其上具有用于使处理器执行本发明的方面的计算机可读程序指令的计算机可读存储介质。计算机可读存储介质可以是可以保持和存储由指令执行设备使用的指令的有形设备。计算机可读存储介质例如可以是－但不限于－电存储设备、磁存储设备、光存储设备、电磁存储设备、半导体存储设备或者上述的任意合适的组合。计算机可读存储介质的更具体的例子(非穷举的列表)也可包括：便携式计算机盘、硬盘、随机存取存储器(RAM)、只读存储器(ROM)、可擦式可编程只读存储器(EPROM或闪存)、静态随机存取存储器(SRAM)、便携式压缩盘只读存储器(CD-ROM)、数字多功能盘(DVD)、记忆棒、软盘、机械编码设备、例如其上存储有指令的打孔卡或凹槽内凸起结构、以及上述的任意合适的组合。这里所使用的计算机可读存储介质不被解释为瞬时信号本身，诸如无线电波或者其他自由传播的电磁波、通过波导或其他传输媒介传播的电磁波(例如，通过光纤电缆的光脉冲)、或者通过电线传输的电信号。

[0081] 这里所描述的计算机可读程序指令可以从计算机可读存储介质下载到各个计算/处理设备，或者通过网络、例如因特网、局域网、广域网和/或无线网下载到外部计算机或外部存储设备。网络可以包括铜传输电缆、光纤传输、无线传输、路由器、防火墙、交换机、网关计算机和/或边缘服务器。每个计算/处理设备中的网络适配卡或者网络接口从网络接收计算机可读程序指令，并转发该计算机可读程序指令，以供存储在各个计算/处理设备中的计算机可读存储介质中。用于执行本发明操作的计算机程序指令可以是汇编指令、指令集架构(ISA)指令、机器指令、机器相关指令、微代码、固件指令、状态设置数据、集成电路配置数据或者以一种或多种编程语言的任意组合编写的源代码或目标代码，所述编程语言包括面向对象的编程语言—诸如Smalltalk、C++等，以及过程式编程语言—诸如“C”语言或类似的编程语言。计算机可读程序指令可以完全地在用户计算机上执行、部分地在用户计算机上执行、作为一个独立的软件包执行、部分在用户计算机上部分在远程计算机上执行、或者完全在远程计算机或服务器上执行。在涉及远程计算机的情形中，远程计算机可以通过任意种类的网络—包括局域网(LAN)或广域网(WAN)—连接到用户计算机，或者，可以连接到外部计算机(例如利用因特网服务提供商来通过因特网连接)。在一些实施例中，通过利用计算机可读程序指令的状态信息来个性化定制电子电路，例如可编程逻辑电路、现场可编程门阵列(FPGA)或可编程逻辑阵列(PLA)，该电子电路可以执行计算机可读程序指令，从而实现本发明的各个方面。

[0082] 这里参照根据本发明实施例的方法、装置(系统)和计算机程序产品的流程图和/

或框图描述了本发明的各个方面。应当理解，流程图和/或框图的每个方框以及流程图和/或框图中各方框的组合，都可以由计算机可读程序指令实现。这些计算机可读程序指令可以提供给通用计算机、专用计算机或其它可编程数据处理装置的处理器，从而生产出一种机器，使得这些指令在通过计算机或其它可编程数据处理装置的处理器执行时，产生了实现流程图和/或框图中的一个或多个方框中规定的功能/操作的装置。也可以把这些计算机可读程序指令存储在计算机可读存储介质中，这些指令使得计算机、可编程数据处理装置和/或其他设备以特定方式工作，从而，存储有指令的计算机可读介质则包括一个制造品，其包括实现流程图和/或框图中的一个或多个方框中规定的功能/操作的各个方面的指令。也可以把计算机可读程序指令加载到计算机、其它可编程数据处理装置、或其它设备上，使得在计算机、其它可编程数据处理装置或其它设备上执行一系列操作步骤，以产生计算机实现的过程，从而使得在计算机、其它可编程数据处理装置、或其它设备上执行的指令实现流程图和/或框图中的一个或多个方框中规定的功能/操作。

[0083] 附图中的流程图和框图显示了根据本发明的多个实施例的系统、方法和计算机程序产品的可能实现的体系架构、功能和操作。在这点上，流程图或框图中的每个方框可以代表一个模块、程序段或指令的一部分，所述模块、程序段或指令的一部分包含一个或多个用于实现规定的逻辑功能的可执行指令。在有些作为替换的实现中，方框中所标注的功能也可以以不同于附图中所标注的顺序发生。例如，两个连续的方框实际上可以基本并行地执行，它们有时也可以按相反的顺序执行，这依所涉及的功能而定。也要注意的是，框图和/或流程图中的每个方框、以及框图和/或流程图中的方框的组合，可以用执行规定的功能或操作的专用的基于硬件的系统来实现，或者可以用专用硬件与计算机指令的组合来实现。

[0084] 尽管以上已经在运行在一个或多个计算机上的计算机程序产品的计算机可执行指令的一般上下文中描述了本发明，但是本领域技术人员将认识到，本公开内容也可以其他程序模块或可结合其他程序模块来实现。通常，程序模块包括执行特定任务和/或实现特定抽象数据类型的例程、程序、组件、数据结构等。此外，本领域技术人员将理解，可以用其他计算机系统配置来实施本发明的计算机实现的方法，包括使用单处理器或多处理器计算机系统、小型计算设备、大型计算机以及手持式计算设备(例如PDA，电话)、基于微处理器或可编程的消费类或工业电子产品等。所示出的各方面也可以在分布式计算环境中实施，其中，由通过通信网络链接的远程处理设备来执行任务。然而，本公开的一些(如果不是全部的话)方面可以在独立计算机上实施。在分布式计算环境中，程序模块可以位于本地和远程存储设备中。

[0085] 如在本申请中使用的，术语“组件”、“系统”、“平台”、“接口”等可以指代和/或可以包括计算机相关的实体或与具有一个或多个特定功能的可操作机器相关的实体。本文公开的实体可以是硬件、硬件和软件的组合、软件或执行中的软件。例如，组件可以是但不限于在处理器对象、可执行文件、执行线程、程序和/或计算机上运行的进程。作为示例，在服务器上运行的应用程序和服务器都可以是组件。一个或多个组件可以驻留在执行的进程和/或执行线程中，并且一个组件可以位于一台计算机上和/或分布在两台或更多台计算机之间。在另一个示例中，各个组件可以从其上存储有各种数据结构的各种计算机可读介质执行。组件可以例如按照具有一个或多个数据分组的信号(例如，来自一个与本地系统、分布式系统中和/或跨网络的另一组件交互的组件的数据)经由本地和/或远程过程进行通信。

作为另一示例，组件可以是具有由电气或电子电路操作的机械部件提供的特定功能的装置，该机械部件由处理器执行的软件或固件应用程序操作。在这种情况下，处理器可以在装置的内部或外部，并且可以执行软件或固件应用程序的至少一部分。作为又一个示例，组件可以是通过不具有机械部件的电子组件提供特定功能的装置，其中电子组件可以包括处理器或其他装置，以执行至少部分地赋予电子组件功能的软件或固件。在一方面，组件可以例如经由云计算系统内的虚拟机来仿真电子组件。

[0086] 另外，术语“或”旨在表示包括性的“或”而不是排他性的“或”。也就是说，除非另有说明，或者从上下文可以清楚地看出，否则“X使用A或B”旨在表示任何自然的包容性排列。也就是说，如果X使用A；X使用B；或X使用A和B两者，则在任何上述情况下均满足“X使用A或B”。此外，在本说明书和附图中使用的冠词“一”和“一个”通常应当被解释为意指“一个或多个”，除非另有说明或上下文中清楚地指向单数形式。如本文所使用的，术语“示例”和/或“示例性”用于表示用作示例、实例或说明。为了避免疑问，本文所公开的主题不受这些示例的限制。另外，本文中描述为“示例”和/或“示例性”的任何方面或设计不必被解释为比其他方面或设计优选或有利，也不意味着排除本领域普通技术人员已知的等效示例性结构和技术。

[0087] 如在本说明书中所采用的术语“处理器”，基本上可以指任何计算处理单元或设备，包括但不限于单核处理器、具有软件多线程执行能力的单处理器、多核处理器、具有软件多线程执行能力的多核处理器、具有硬件多线程技术的多核处理器、并行平台、以及具有分布式共享内存的并行平台。另外，处理器可以指集成电路、专用集成电路(ASIC)、数字信号处理器(DSP)、现场可编程门阵列(FPGA)、可编程逻辑控制器(PLC)、复杂的可编程逻辑设备(CPLD)、分立的栅极或晶体管逻辑、分立的硬件组件或为执行本文所述的功能而设计的上述的任何租户。此外，处理器可以利用为优化空间使用或增强用户设备的性能的纳米级架构，例如但不限于基于分子和量子点的晶体管、开关和门。处理器也可以被实现为计算处理单元的组合。在本公开中，诸如“储存器”、“存储器”、“数据储存器”、“数据存储器”、“数据库”以及与组件的操作和功能有关的基本上任何其他信息存储组件的术语被用来指代“存储器组件”、体现在“存储器”中的实体或包含存储器的组件。应当理解，本文所述的存储器和/或存储器组件可以是易失性存储器或非易失性存储器，或者可以包括易失性和非易失性存储器两者。作为说明而非限制，非易失性存储器可以包括只读存储器(ROM)、可编程ROM(PROM)、电可编程ROM(EPROM)、电可擦除ROM(EEPROM)，闪存或非易失性随机存取存储器(RAM)(例如铁电RAM(FeRAM)。易失性存储器可以包括RAM，其可以例如用作外部高速缓存。通过说明而非限制，RAM有多种形式，诸如同步RAM(SRAM)、动态RAM(DRAM)、同步DRAM(SDRAM)、双倍数据速率SDRAM(DDR SDRAM)、强型SDRAM(ESDRAM)、Synchlink DRAM(SLDRAM)、直接Rambus RAM(DRRAM)、直接Rambus动态RAM(DRDRAM)和Rambus动态RAM(RDRAM)。另外，本文公开的系统或计算机实现的方法的存储器组件旨在包括但不限于包括这些以及任何其他合适类型的存储器。

[0088] 上面已经描述的内容仅包括系统和计算机实现的方法的示例。当然，为了描述本发明的目的，不可能描述组件或计算机实现的方法的每种可能的组合，但是本领域的普通技术人员可以认识到，本公开的许多进一步的组合和置换是可能的。此外，在说明书、权利要求、附录和附图中使用了术语“包括”、“有”，“具有”，这些术语旨在是包含性的，类似于在

权利要求中被用作过渡词的术语“包括”的解释。

[0089] 已经出于说明的目的给出了对本发明的各种实施例的描述，但是这些描述并不旨在是穷举的或将本发明限制于此。在不脱离本发明范围的情况下，许多修改和变化对于本领域普通技术人员将是显而易见的。选择这里使用的术语是为了最好地解释本发明的实施例的原理，对市场上发现的技术的实际应用或对技术的改进，或者使本领域的其他普通技术人员能够理解本发明。

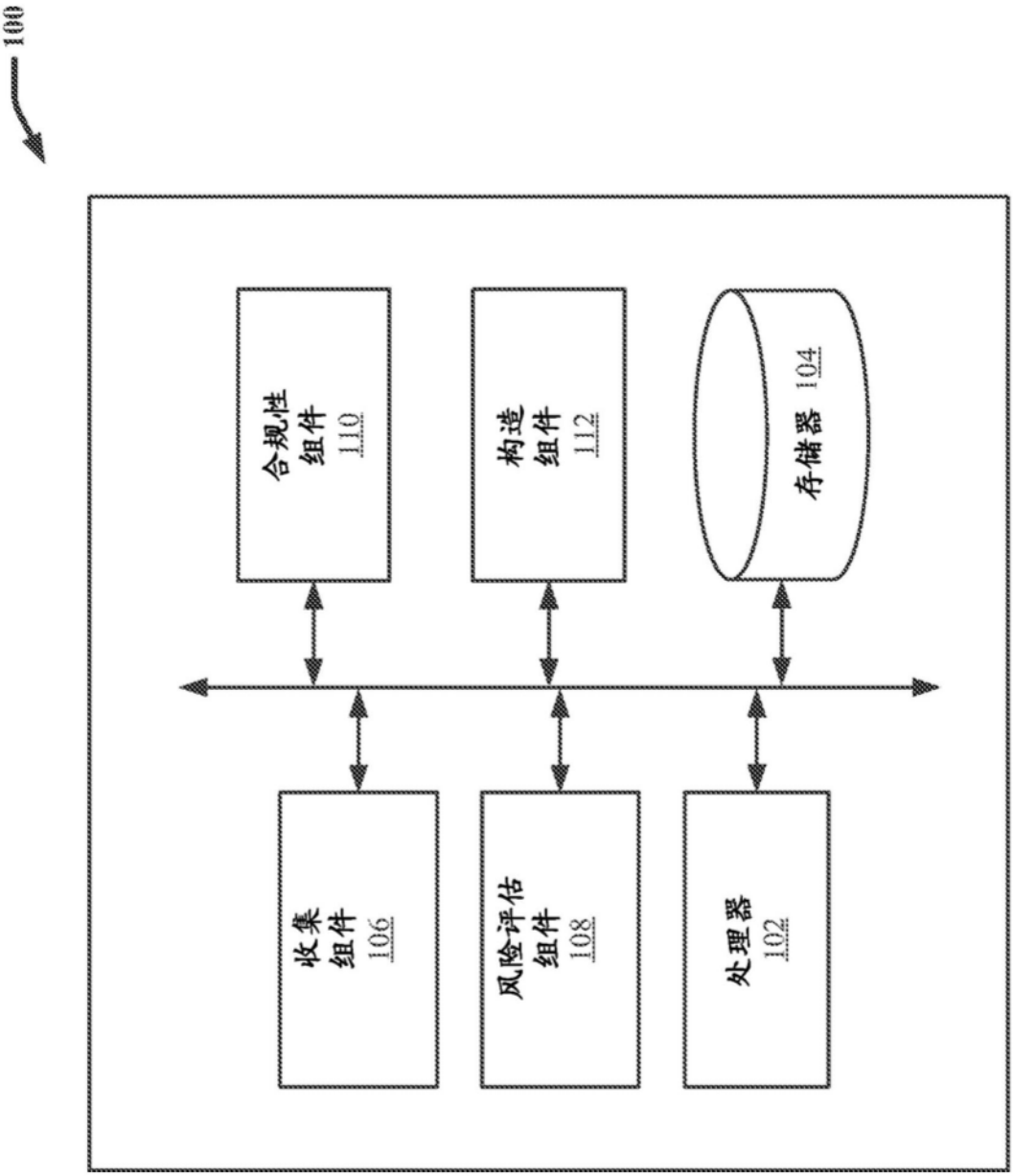


图1

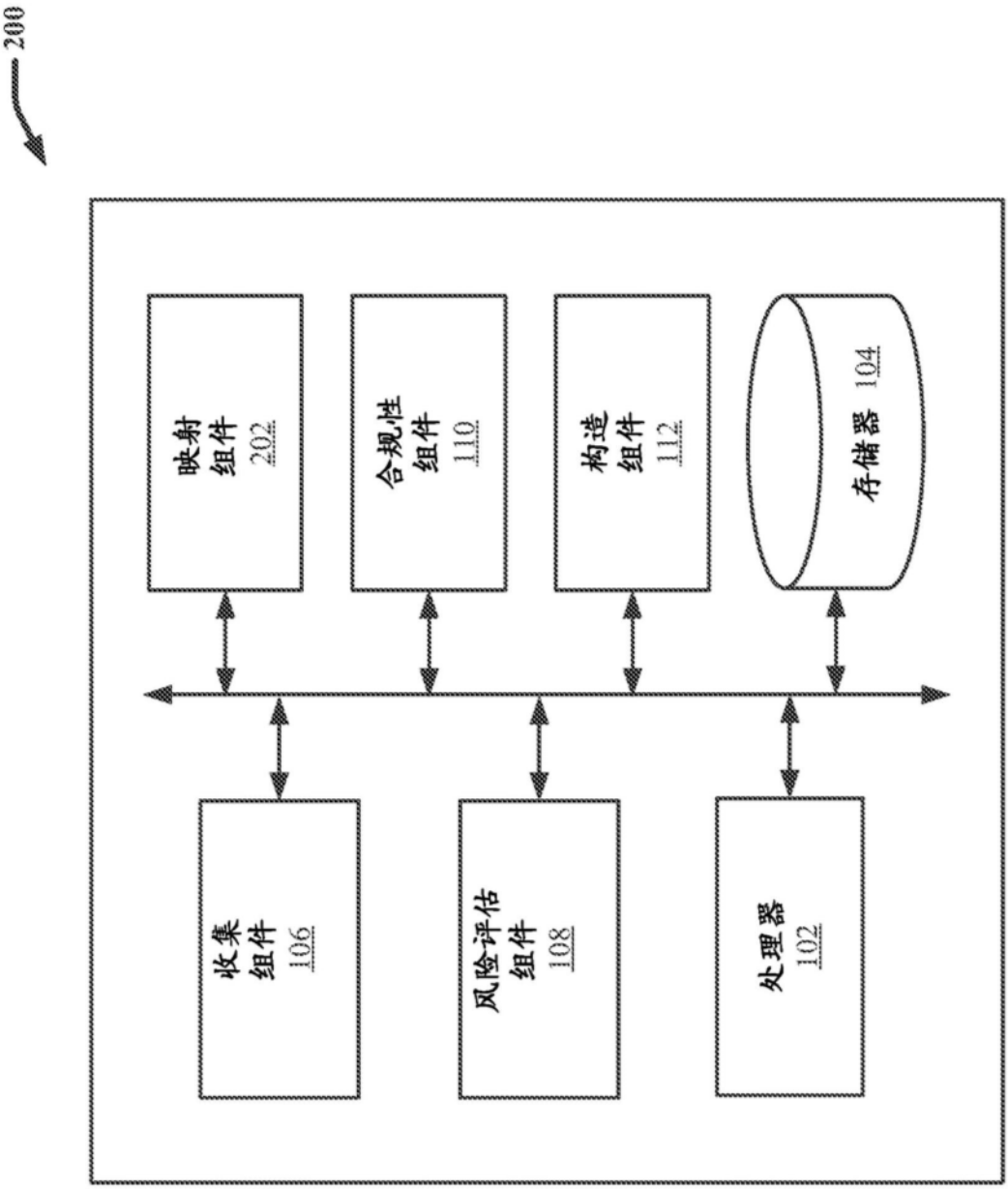

200
映射组件 202
合规性组件 110
构造组件 112
存储器 104
收集组件 106
风险评估组件 108
处理器 102

图2

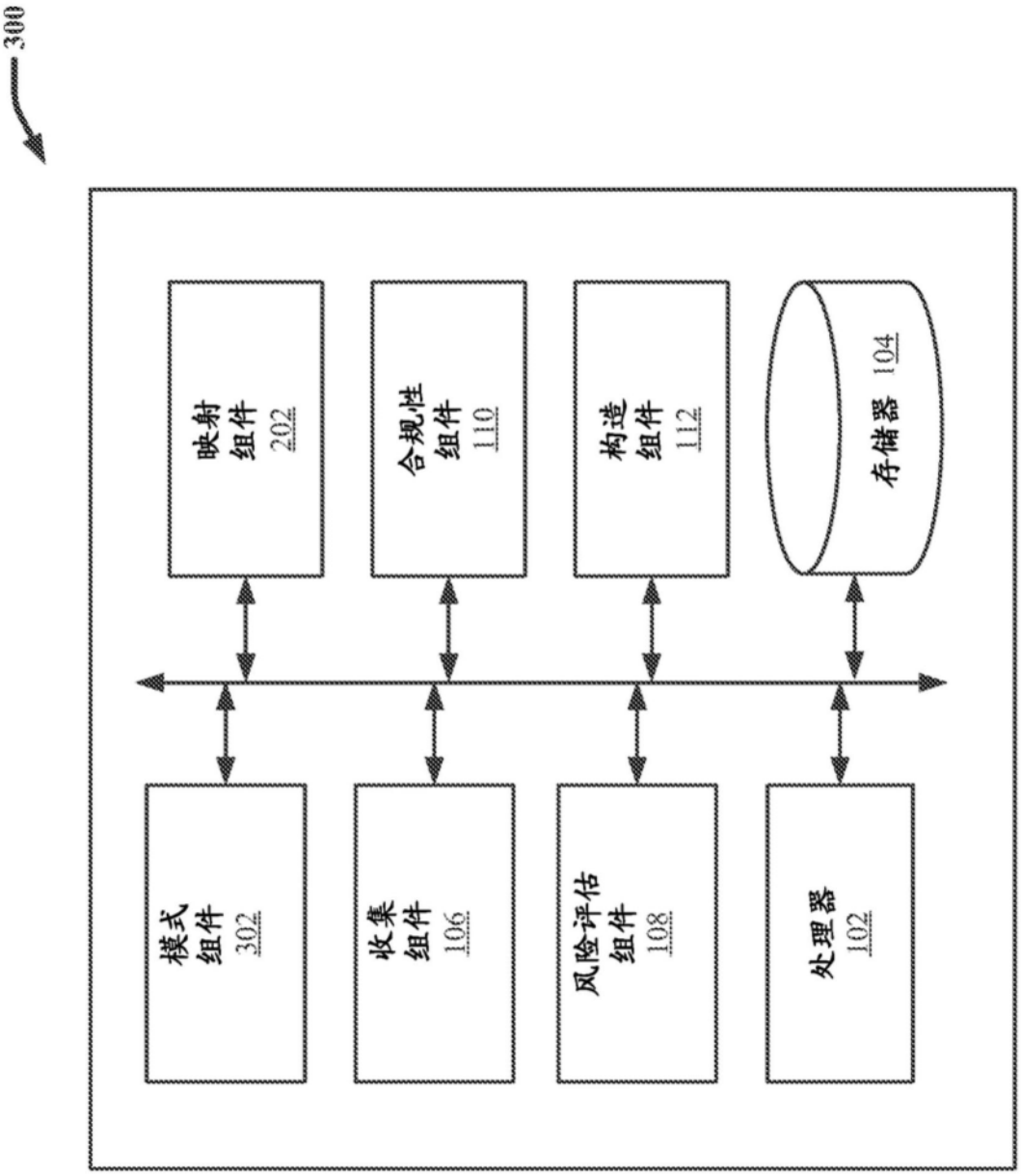

300
映射组件 202
合规性组件 110
构造组件 112
存储器 104
模式组件 302
收集组件 106
风险评估组件 108
处理器 102

图3

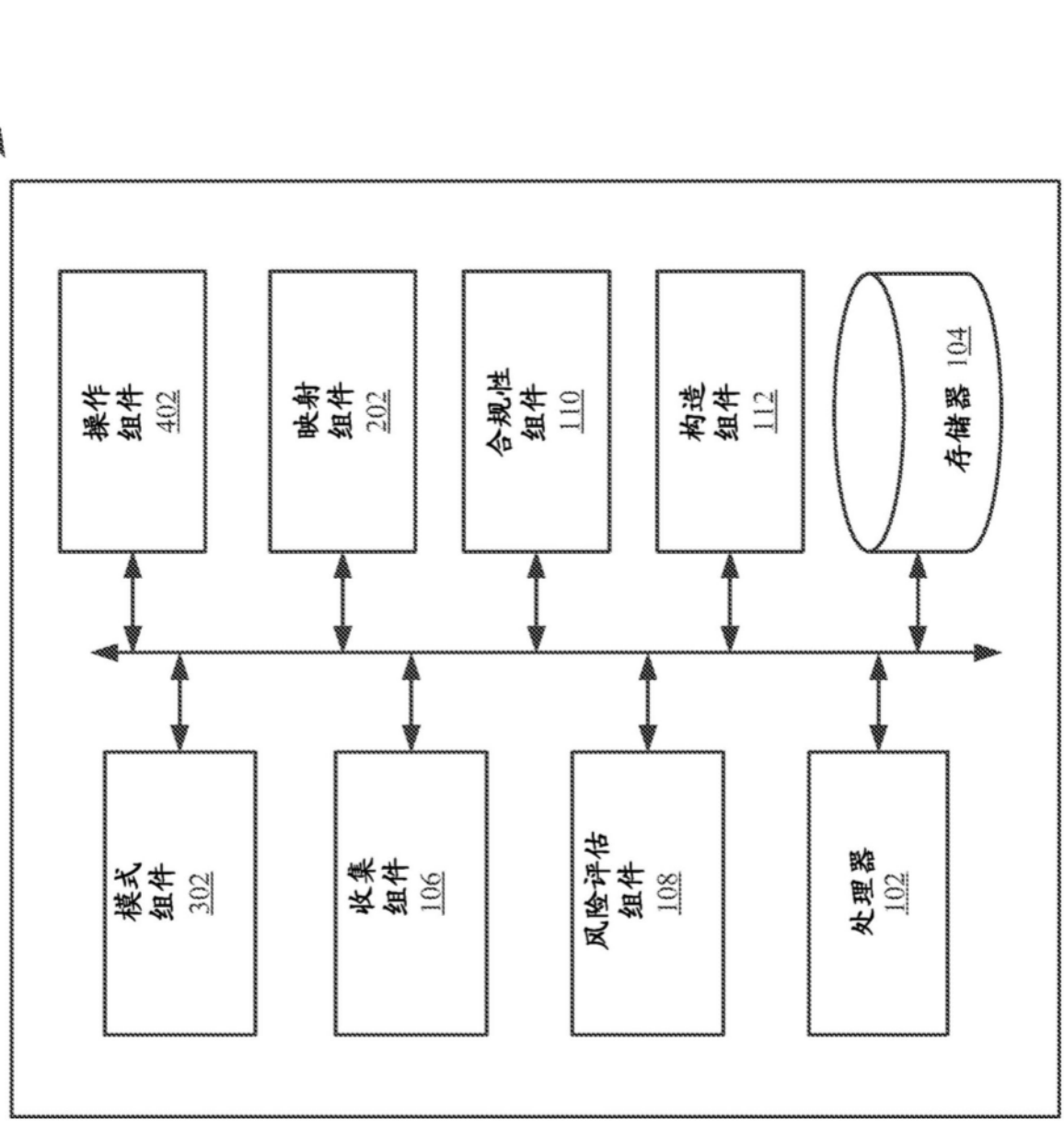

400
操作组件 402
映射组件 202
合规性组件 110
构造组件 112
存储器 104
模式组件 302
收集组件 106
风险评估组件 108
处理器 102

图4

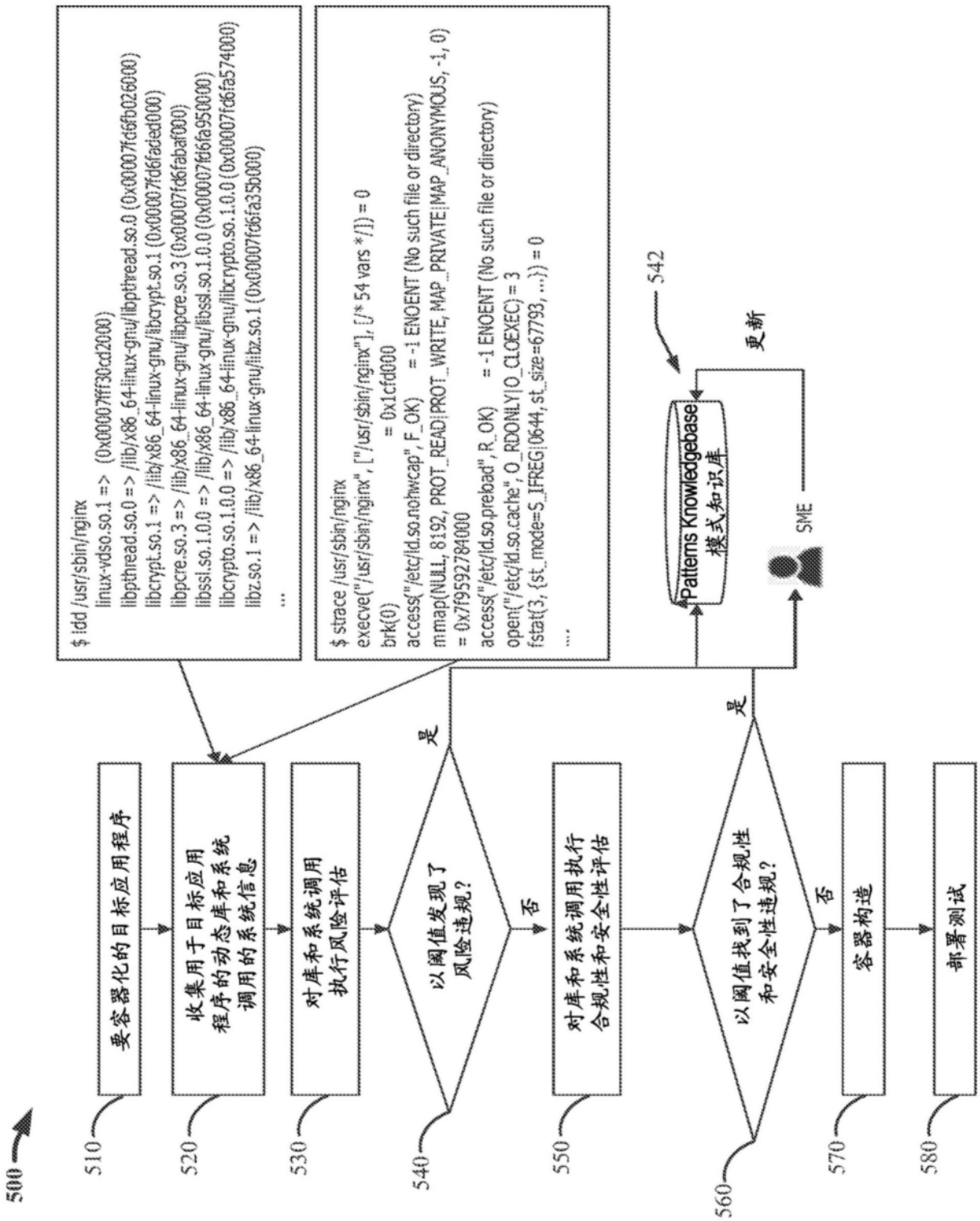

500
510
要容器化的目标应用程序
520
收集用于目标应用程序的动态库和系统调用的系统信息
530
对库和系统调用执行风险评估
540
以阈值发现了风险违规?
是
否
550
对库和系统调用执行合规性和安全性评估
560
以阈值找到了合规性和安全性违规?
是
否
570
容器构造
580
部署测试
$ ldd /usr/sbin/nginx
linux-vdso.so.1 => (0x00007fff30cd2000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007fd6fb026000)
libcrypt.so.1 => /lib/x86_64-linux-gnu/libcrypt.so.1 (0x00007fd6faded000)
libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007fd6fabaf000)
libssl.so.1.0.0 => /lib/x86_64-linux-gnu/libssl.so.1.0.0 (0x00007fd6fa950000)
libcrypto.so.1.0.0 => /lib/x86_64-linux-gnu/libcrypto.so.1.0.0 (0x00007fd6fa574000)
libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007fd6fa35b000)
...
$ strace /usr/sbin/nginx
execve("/usr/sbin/nginx", ["/usr/sbin/nginx"], [/* 54 vars */]) = 0
brk(0) = 0x1cfd000
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f9592784000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=67793, ...}) = 0
...
542
Patterns Knowledgebase
模式知识库
更新
SME

图5

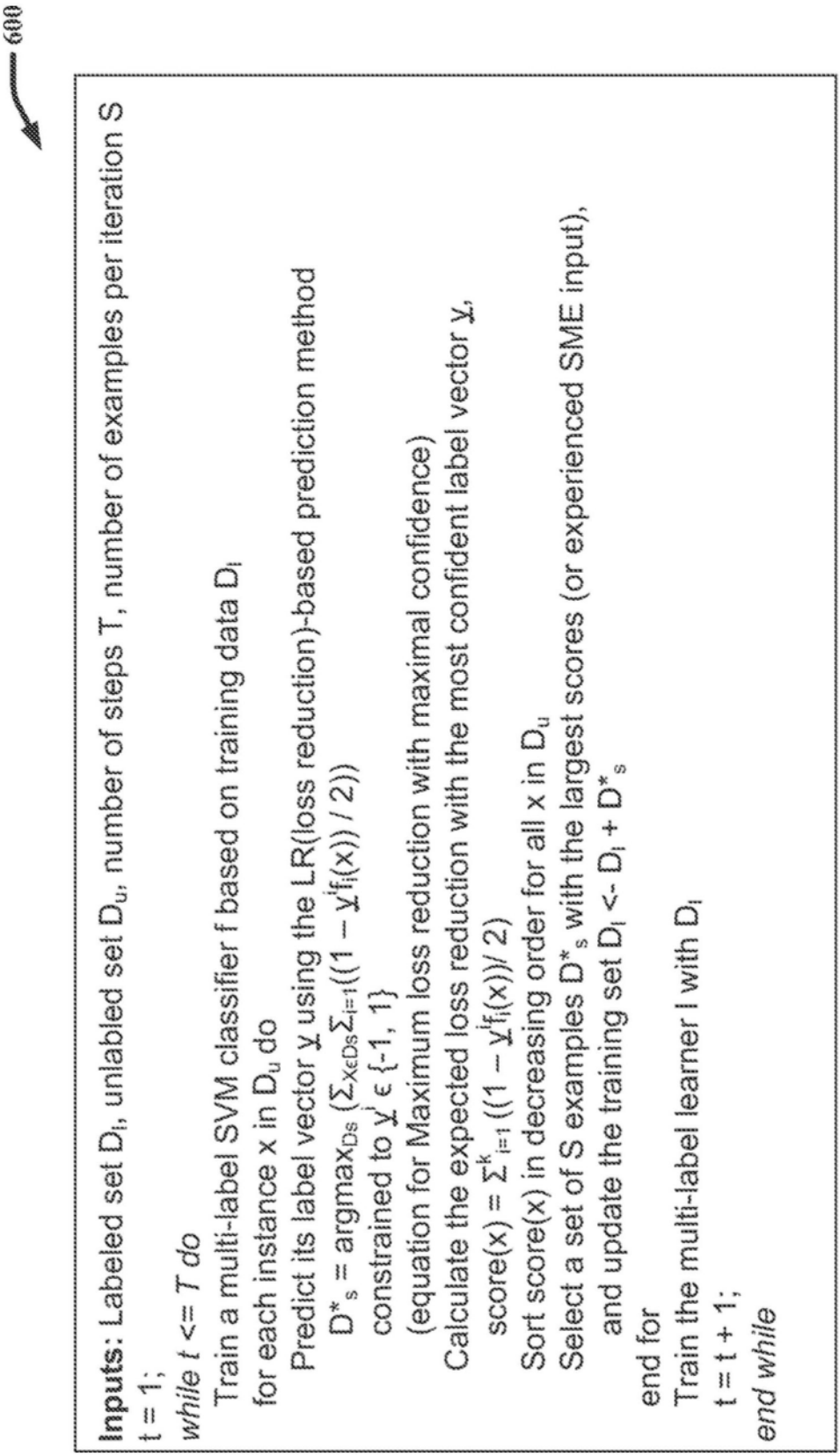

600
Inputs: Labeled set D_l, unlabled set D_u, number of steps T, number of examples per iteration S
t = 1;
while t <= T do
Train a multi-label SVM classifier f based on training data D_l
for each instance x in D_u do
Predict its label vector $\mathcal{Y}$ using the LR(loss reduction)-based prediction method
$D^*_s = \text{argmax}_{D_s} (\Sigma_{X \epsilon D_s} \Sigma_{i=1} ((1 - \mathcal{Y}^i f_i(x)) / 2))$
constrained to $\mathcal{Y}^i \epsilon \{-1, 1\}$
(equation for Maximum loss reduction with maximal confidence)
Calculate the expected loss reduction with the most confident label vector $\mathcal{Y}$,
$score(x) = \Sigma^k_{i=1} ((1 - \mathcal{Y}^i f_i(x))/ 2)$
Sort score(x) in decreasing order for all x in D_u
Select a set of S examples D^*_s with the largest scores (or experienced SME input),
and update the training set D_l <- D_l + D^*_s
end for
Train the multi-label learner l with D_l
t = t + 1;
end while

图6

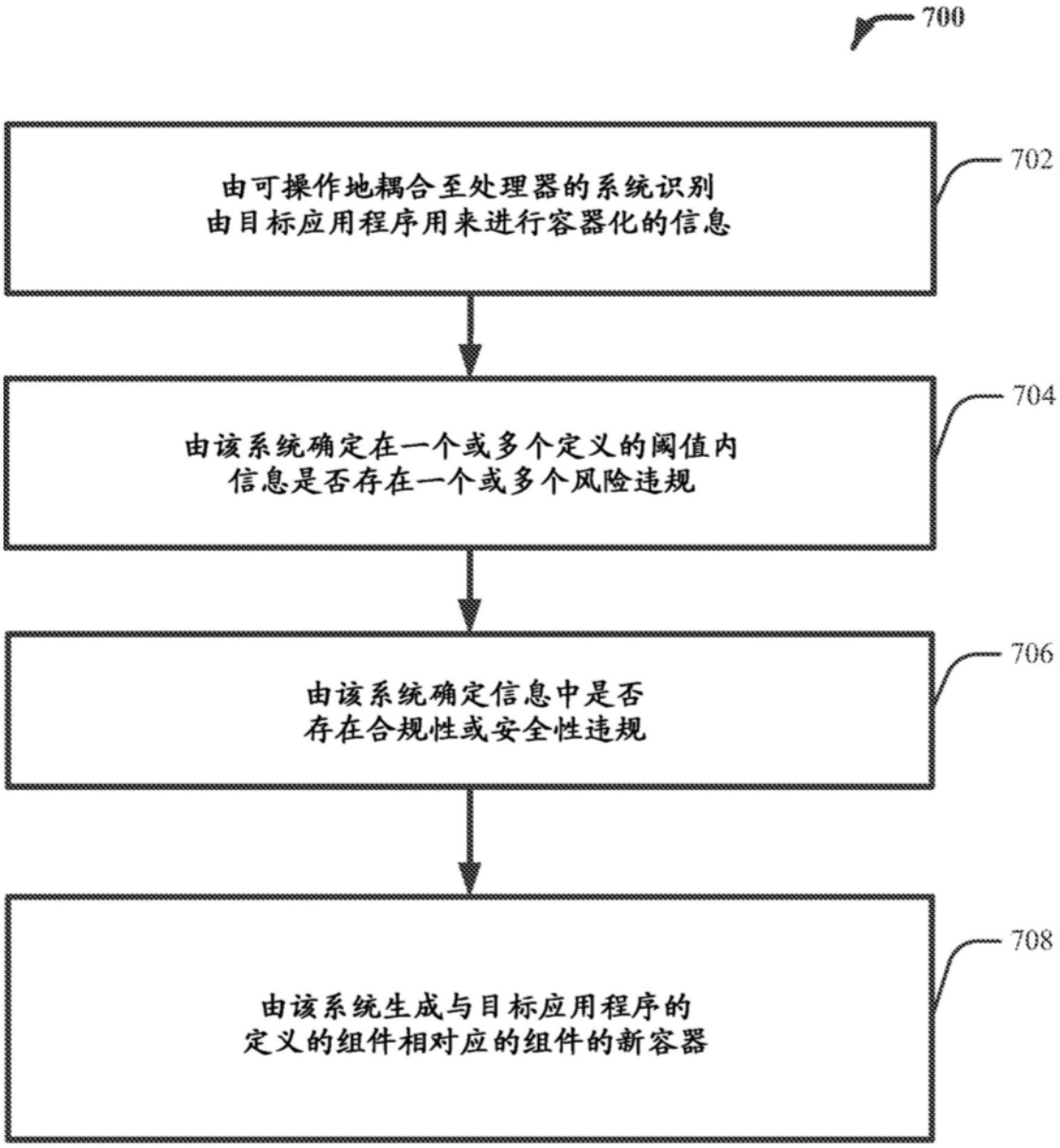

700
由可操作地耦合至处理器的系统识别
由目标应用程序用来进行容器化的信息
702
由该系统确定在一个或多个定义的阈值内
信息是否存在一个或多个风险违规
704
由该系统确定信息中是否
存在合规性或安全性违规
706
由该系统生成与目标应用程序的
定义的组件相对应的组件的新容器
708

图7

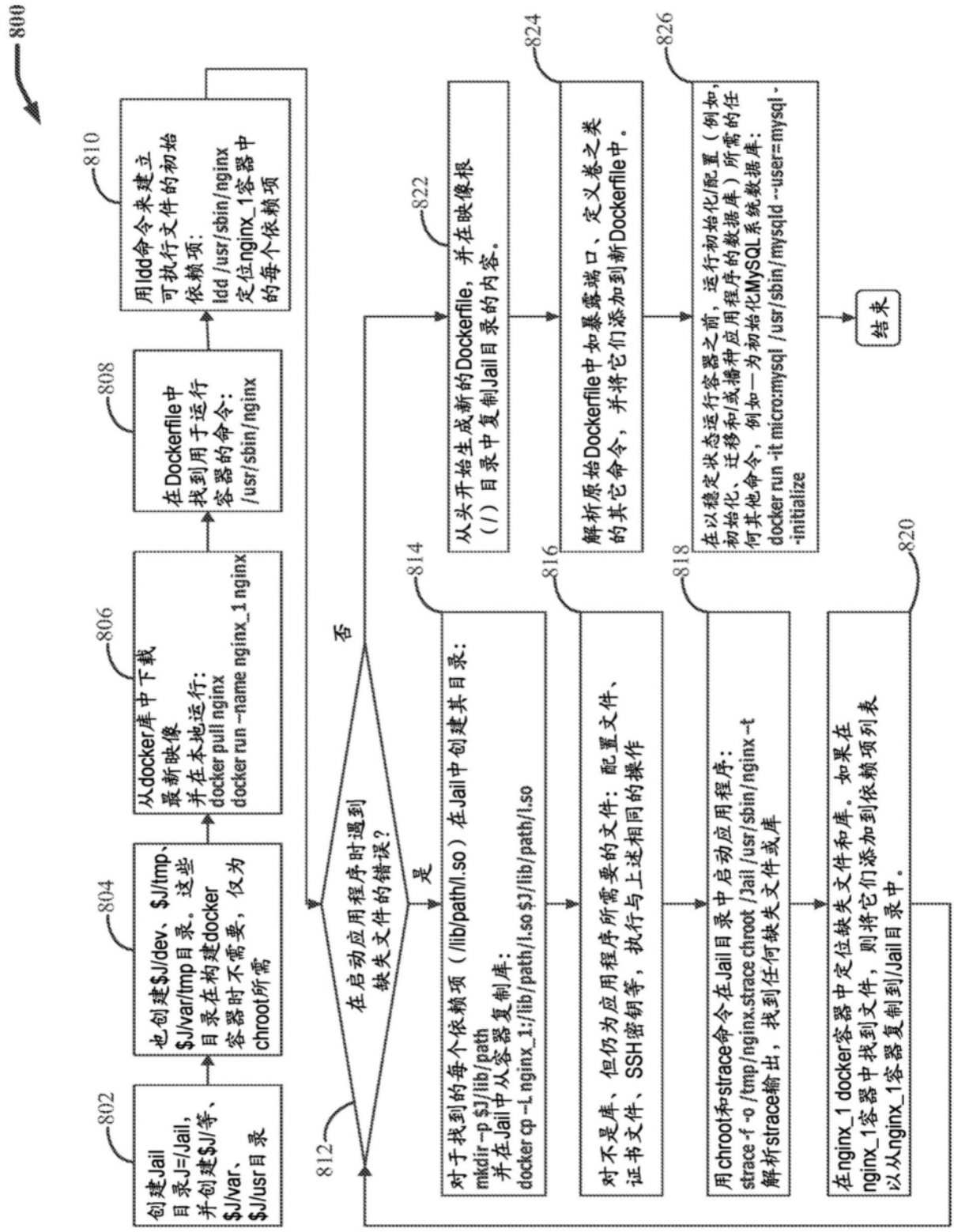

800
802
创建Jail
目录J=/Jail，
并创建$J/等、
$J/var、
$J/usr目录
804
也创建$J/dev、$J/tmp、
$J/var/tmp目录。这些
目录在构建docker
容器时不需要，仅为
chroot所需
806
从docker库中下载
最新映像
并在本地运行：
docker pull nginx
docker run --name nginx_1 nginx
808
在Dockerfile中
找到用于运行
容器的命令：
/usr/sbin/nginx
810
用ldd命令来建立
可执行文件的初始
依赖项：
ldd /usr/sbin/nginx
定位nginx_1容器中
的每个依赖项
812
在启动应用程序时遇到
缺失文件的错误？
否
是
814
对于找到的每个依赖项（/lib/path/l.so）在Jail中创建其目录：
mkdir -p $J/lib/path
并在Jail中从容器复制库：
docker cp -L nginx_1:/lib/path/l.so $J/lib/path/l.so
816
对不是库、但仍为应用程序所需要的文件：配置文件、
证书文件、SSH密钥等，执行与上述相同的操作
818
用chroot和strace命令在Jail目录中启动应用程序：
strace -f -o /tmp/nginx.strace chroot /Jail /usr/sbin/nginx -t
解析strace输出，找到任何缺失文件或库
820
在nginx_1 docker容器中定位缺失文件和库。如果在
nginx_1容器中找到文件，则将它们添加到依赖项列表
以从nginx_1容器复制到/Jail目录中。
822
从头开始生成新的Dockerfile，并在映像根
（/）目录中复制Jail目录的内容。
824
解析原始Dockerfile中如暴露端口、定义卷之类
的其它命令，并将它们添加到新Dockerfile中。
826
在以稳定状态运行容器之前，运行初始化/配置（例如，
初始化、迁移和/或播种应用程序的数据库）所需的任
何其他命令，例如一为初始化MySQL系统数据库：
docker run -it micro:mysql /usr/sbin/mysqld --user=mysql
--initialize
结束

图8

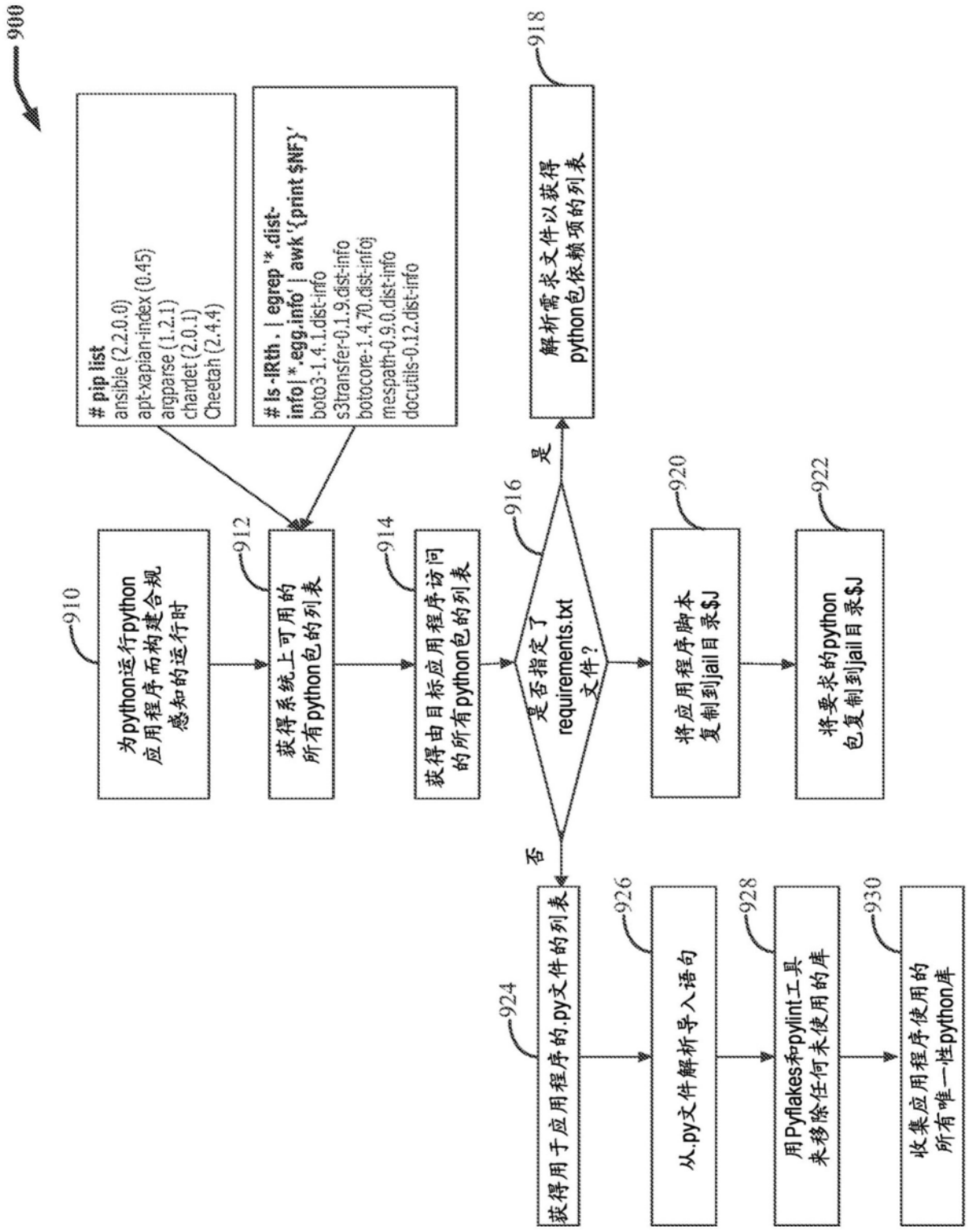

900
910
为python运行python应用程序而构建合规感知的运行时
912
获得系统上可用的所有python包的列表
914
获得由目标应用程序访问的所有python包的列表
916
是否指定了requirements.txt文件?
是
否
918
解析需求文件以获得python包依赖项的列表
920
将应用程序脚本复制到jail目录$J
922
将要求的python包复制到jail目录$J
924
获得用于应用程序的.py文件的列表
926
从.py文件解析导入语句
928
用Pyflakes和pylint工具来移除任何未使用的库
930
收集应用程序使用的所有唯一性python库
pip list
ansible (2.2.0.0)
apt-xapian-index (0.45)
argparse (1.2.1)
chardet (2.0.1)
Cheetah (2.4.4)
ls -lRth . | egrep '*.dist-info|*.egg.info' | awk '{print $NF}'
boto3-1.4.1.dist-info
s3transfer-0.1.9.dist-info
botocore-1.4.70.dist-info
mespath-0.9.0.dist-info
docutils-0.12.dist-info

图9

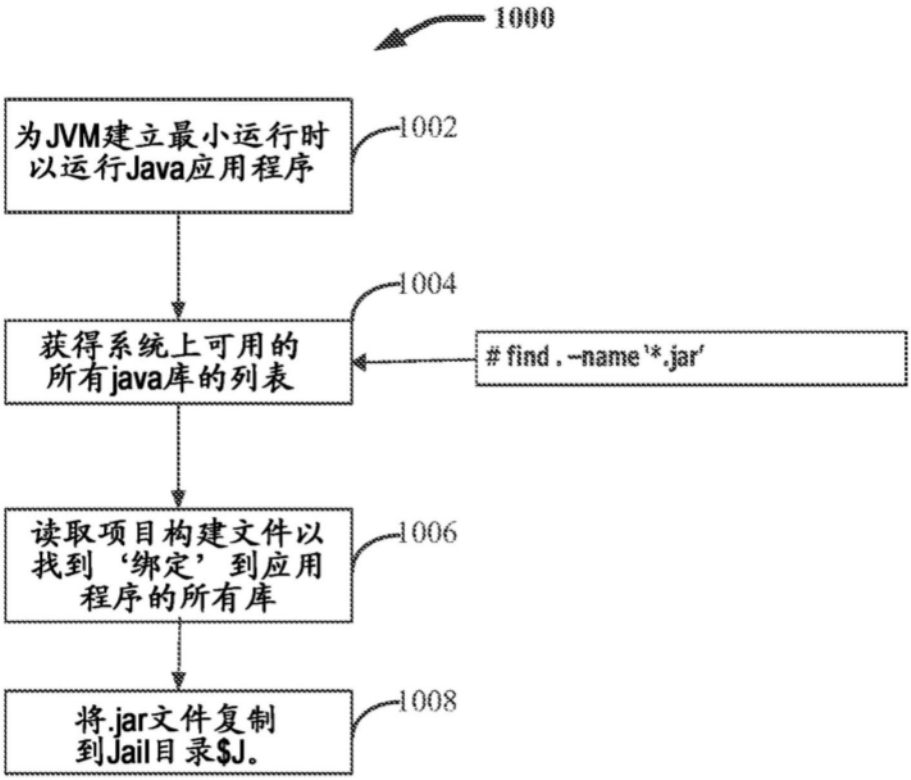

1000
为JVM建立最小运行时以运行Java应用程序
1002
获得系统上可用的所有java库的列表
1004
find . -name '*.jar'
读取项目构建文件以找到‘绑定’到应用程序的所有库
1006
将.jar文件复制到Jail目录$J。
1008

图10

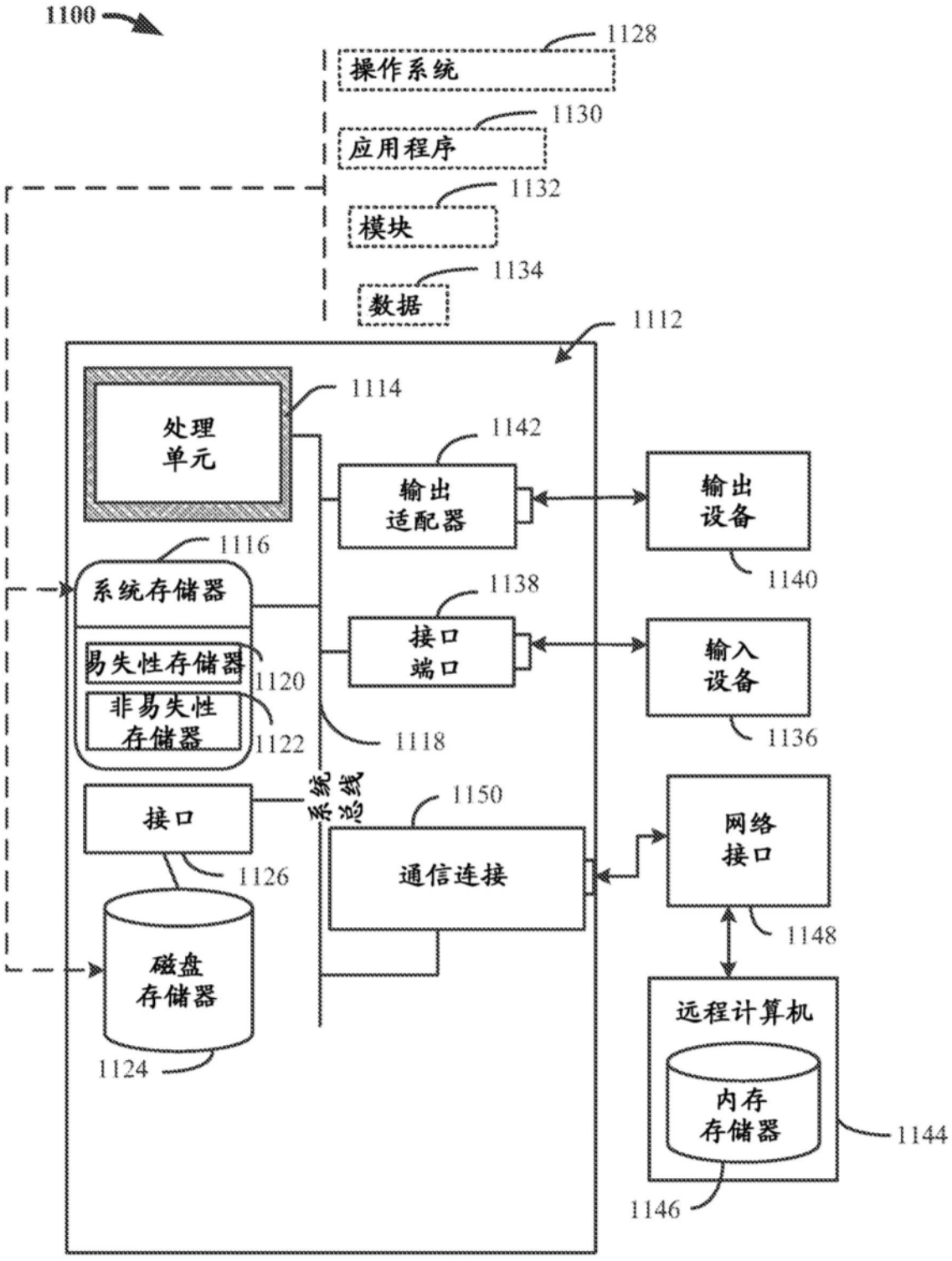

1100
1128
操作系统
1130
应用程序
1132
模块
1134
数据
1112
1114
处理单元
1142
输出适配器
输出设备
1140
1116
系统存储器
1138
接口端口
输入设备
1136
易失性存储器
1120
非易失性存储器
1122
1118
系统总线
接口
1150
通信连接
网络接口
1148
1126
磁盘存储器
1124
远程计算机
内存存储器
1144
1146

图11

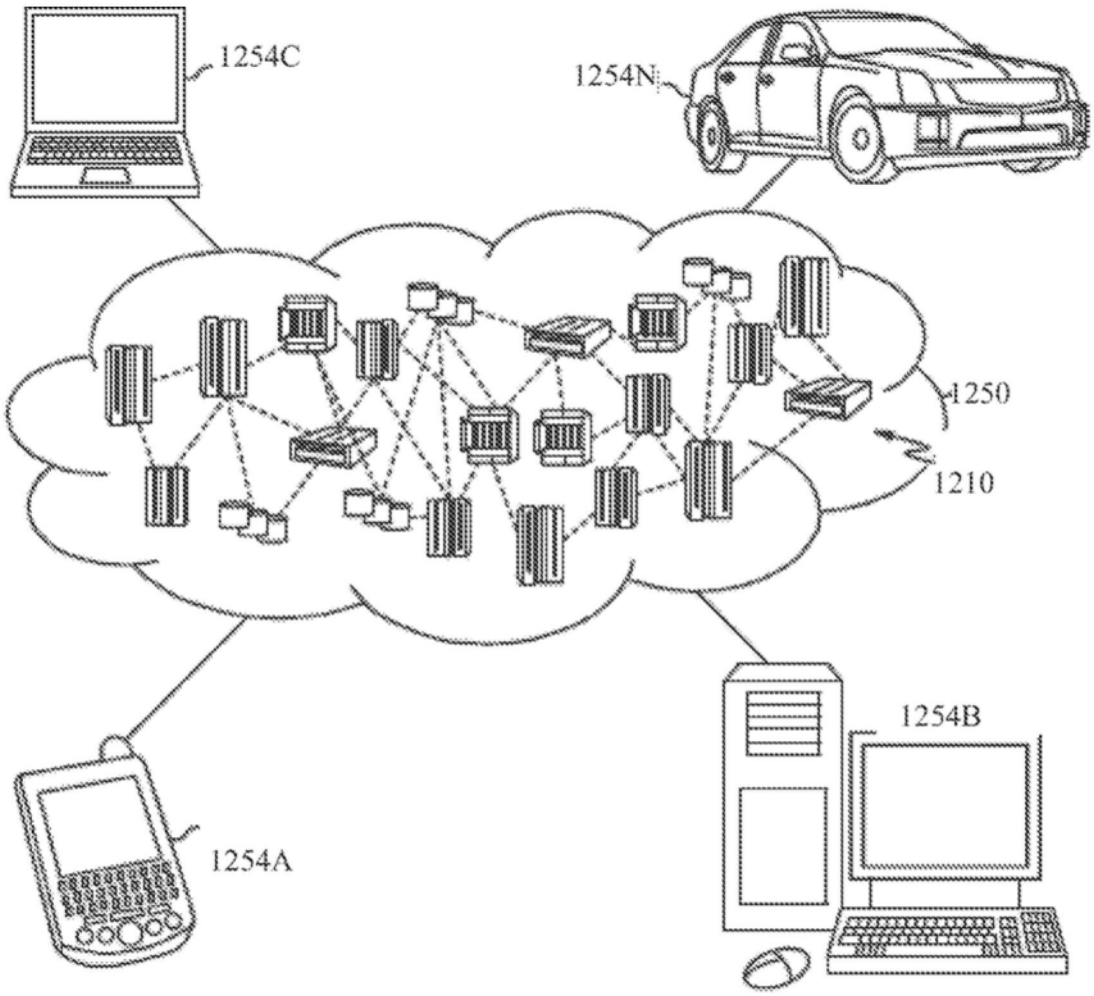


图12

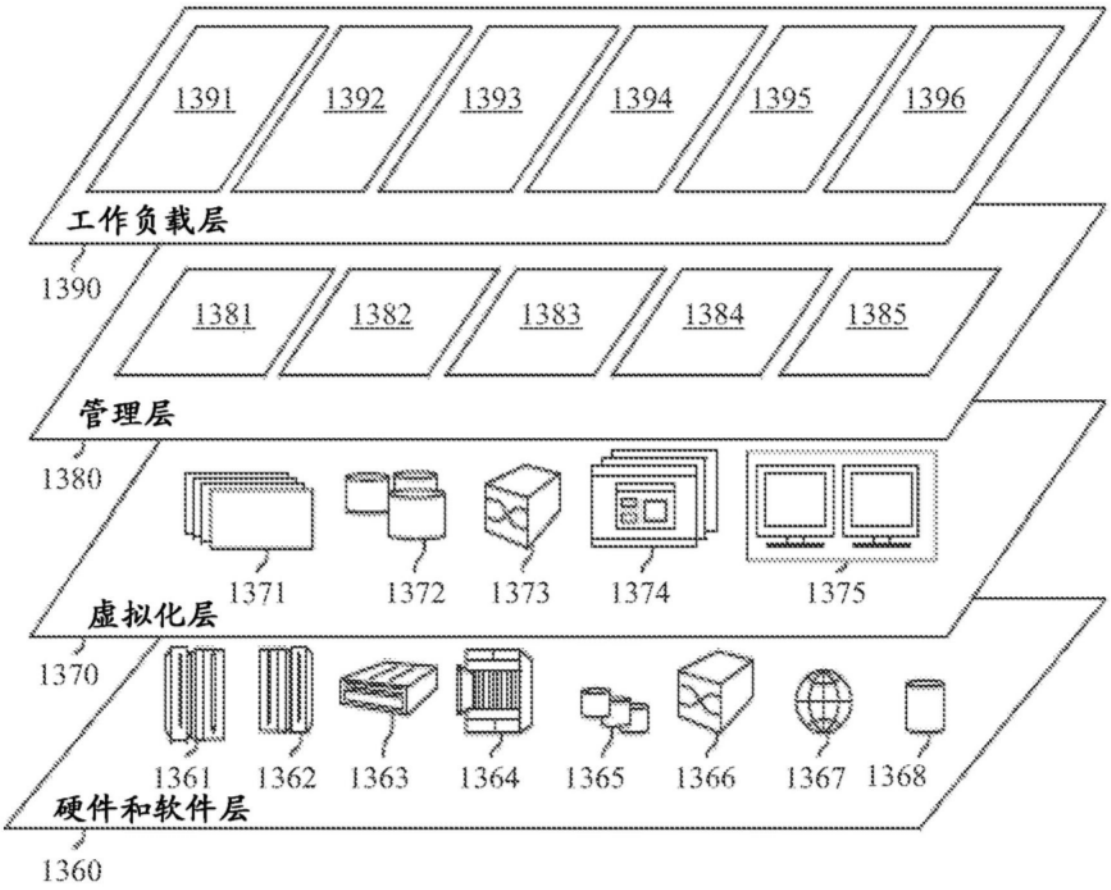

1391
1392
1393
1394
1395
1396
工作负载层
1390
1381
1382
1383
1384
1385
管理层
1380
1371
1372
1373
1374
1375
虚拟化层
1370
1361
1362
1363
1364
1365
1366
1367
1368
硬件和软件层
1360

图13