基于异常检测 Docker 容器的监控系统研究

谢兆贤,倪冰雪*,王若冰
(曲阜师范大学 网络空间安全学院,山东 曲阜 273165)

摘 要:Docker 容器监控系统在系统运维层面保障容器的安全。为解决当前容器监控系统存在部署过程复杂、异常检测精确度低、资源消耗量大和监控潜在黑洞等问题,采用 Prometheus+、Sysdig 和 Weave scope 等组件,构建支持可视化交互的综合型监控系统框架。该系统不仅可以快速准确定位异常来源、易于部署和资源消耗量低,还可以采用多种执行方式。依照监控组件资源使用量和组件执行模式,设计实验并对该系统进行研究。实验结果显示,Prometheus+磁盘故障检出率高并且可以长期存储数据,Sysdig 在异常检测的误检率低并且耗费资源少,Weave Scope 的异常检出率低,但是它可以同时监控多个容器。实验结果验证了该系统的有效性,其不仅可以全面地实时监控 Docker 容器内的各个节点,还可以解决多容器联合监控的问题。同时,系统从根本上降低了 Docker 容器整体的安全风险。

关键词:Docker 容器;监控系统;监控组件;安全性;异常检测
中图分类号:TP399 **文献标识码**:A **文章编号**:1673-629X(2022)06-0131-07
doi:10.3969/j.issn.1673-629X.2022.06.022

Research on Monitoring System of Docker Container Based on Anomaly Detection

XIE Zhao-xian, NI Bing-xue*, WANG Ruo-bing
(School of Cyber Science and Engineering, Qufu Normal University, Qufu 273165, China)

Abstract: Docker container monitoring system ensures the security of containers at the level of system operation and maintenance. In order to solve the problems of the current container monitoring system, such as complex deployment process, low accuracy of anomaly detection, large resource consumption, and monitoring potential black holes, a comprehensive monitoring system framework supporting visual interaction is constructed by using Prometheus+, Sysdig and Weave scope. This system can not only locate the source of anomaly quickly and accurately, but also can be deployed easily with low resource consumption. According to the monitoring component resource usage and component execution mode, the experiment is designed and the system is studied. The experimental results show that Prometheus+ disk exists a high fault detection rate and stores data for a long time. Sysdig has the features both low false detection rate and low resource consumption in anomaly detection. The abnormal detection rate of Weave Scope is even low, but it can monitor multiple containers concurrently. Experimental results verify the effectiveness of the system. Then, it is not only comprehensively monitor each node in Docker container in real time, but also solve the problem of joint monitoring of multiple containers. At the same time, this system fundamentally reduces the overall security risk of Docker container.

Key words: Docker container; monitoring system; monitoring component; security; anomaly detection

0 引 言

Docker 是一个重要的轻量级虚拟化技术,改变了传统开发、测试和部署应用的模式。由于容器技术的主机内核呈现“曝光式”,导致攻击者可以直接对主机内核发动恶意的攻击[1]。所以,Docker 所在宿主机以及容器本身安全性的问题,更显得重要。在运维体系中,监控是不可或缺的组成部分[2]。它不只是系统可靠性的基础,也是系统稳定运行和故障排除的关键[3]。在监控的过程中,常见的四种监控指标为 CPU 使用率、内存使用率、磁盘读写速率和网络速度。在现有的监控系统中,主要通过检测异常数据来保护容器的安全,异常检测是指在数据中发现与预期行为不符的问题模式,这种不合格的模式包含所有不同应用领域中的异常情况。

收稿日期:2021-07-12 **修回日期**:2021-11-15
基金项目:山东省自然科学基金资助面上项目(ZR2020MF048)
作者简介:谢兆贤(1971-),男,副教授,硕导,博士,研究方向为云计算、数据库、智能制造;通信作者:倪冰雪(1999-),女,研究方向为软件工程、智能数据。

目前,已知的三大类异常检测的方法有无监督异常监测方法[4]、监督式异常检测方法和半监督式异常检测方法。监控系统的异常检测方法,在实际检测的研究领域仍然存在不足。这是因为监控系统的误检率和漏检率,降低了异常检测的精确度。所以,只要能够降低误检率和漏检率,便能够保障容器的安全。

尽管现有的 Docker 容器可以使用的监控工具种类繁多,仍然存在部署过程复杂、稳定性差、资源消耗量大、管理过程繁琐和只能监控单一容器等问题。所以,监控系统若是不能进行综合监控,则不能对 Docker 容器的所有节点实施监控,更不能对系统中出现的故障进行提前预警。所以,针对以上存在的各项问题,该文开发了一个框架,对 Prometheus+、Sysdig 和 Weave scope 等方式进行组合,实现对 Docker 容器的全面监控。

该文有以下三点贡献:(1)进行 Docker 容器面临的安全风险研究,发展监控系统在运维层面保障容器安全的系统。(2)设置四个监控指标,即可掌握系统的精确度。使用 Prometheus+、Sysdig 和 Weave Scope 三种监控组件,根据 CPU 异常、内存异常、磁盘异常和网络异常等指标,监控异常情况。(3)进行实验,提供许多有价值的实验结果与分析。

1 背景知识

1.1 Docker 容器基础

开源的 Docker 容器是一种基于 LXC 的高级容器引擎[5],它利用 REST 技术管理 Docker 镜像启动的容器,支持新一代的云计算平台[6]。同时,Docker 主要由 Docker Client、Docker Daemon 和 Docker Regeistry 三个部分组成[7]。Docker Client 是远程控制器。Docker Daemon 是一个后台处理过程[8]。由计算机系统在 Docker Service 启动时自动创建[7],它会对客户端的请求进行管理。Docker Regeistry 是保存镜像的仓库。主要的任务是管理镜像、处理客户端请求、进行用户认证和反馈请求的服务。

1.2 Docker 容器安全

Docker 的安全主要依赖 namespace、cgroups 和 capability 三种 Linux 内核的安全机制[9]。这三种机制为 Docker 容器提供一个安全的工作环境,但是仍不能够保障 Docker 容器完全安全。根据 Docker 容器三个常见的攻击面,可以从网络、容器和镜像等三个方面讨论面临的安全问题[7]。

● Docker 网络安全风险。在容器轻量级的虚拟化网络环境中,网络面临的安全问题更加复杂。Docker 容器的网络通信没有细粒度的控制机制[10],容器可以不加限制地访问互联网,容器在联网时容易受到网络攻击[11]。由于没有办法对同一个宿主机内各容器之间的网络访问权限进行限制,这就使攻击者可以通过某个容器对宿主机内的其他容器进行 ARP 欺骗、MAC 泛洪攻击等攻击,进而带来许多的安全问题。除此之外,Docker 容器网络也容易受到拒绝服务(DoS)的攻击[12],这个攻击能够降低其他容器的网络数据处理能力,甚至占满宿主机的网络带宽资源。

● Docker 容器安全风险。由于 Docker 容器与宿主机共享操作系统的内核[10],仅靠 namespace 技术不能实现对系统资源的绝对隔离,因此存在容器与宿主机、容器与容器之间的进程隔离、文件系统隔离和进程间通信隔离等方面的安全风险。

● Docker 容器镜像风险。Docker Hub 中的镜像可以来自官方镜像仓库,也允许第三方组织和个人进行上传。Docker 镜像的安全风险表现在创建的过程、获取的来源或获取的途径等方面,开发者在构建镜像时可能会将一些敏感信息添加到镜像中[8],有可能导致 Docker 容器数据的信息外漏。

1.3 监控组件框架

1.3.1 Prometheus+

Prometheus+的主要模块有 Prometheus Server、Service Discovery、Node _ exporter 和 Grafana 等。Prometheus Server 主要用于抓取数据和存储时序数据,另外还提供查询和 Alert Rule 配置管理。Service Discovery 自动把节点发现到 Prometheus,不需要手动配置 Prometheus. yml 配置文件。Node_exporter 用于客户端数据收集,输出被监控组件信息的 HTTP 接口,主要负责收集数据并将信息汇报给 Prometheus Server 的组件。Grafana 是一个跨平台的度量分析和数据可视化工具,用于展示 Node_exporter 中收集的时序数据。在 Grafana 可视化界面上,明确显示出被监测的各项性能在特定时期的变化[13]。

1.3.2 Sysdig

Sysdig 能够分析 Linux 系统的“现场”状态,将该状态保存为转储文件以供离线分析检查。通过 Sysdig 工具,用户能够很方便地查看到主机上所有应用程序的 CPU、文件 I/O、网络访问状况。

1.3.3 Weave Scope

Weave Scope 提供自上而下的应用程序视图以及整个基础架构视图,可以自动生成应用程序的映像,具有丰富的功能和友好的操作界面,可以实施多主机的监控。它通过查看容器的上下文指标、标签和元数据,在容器内部的进程之间导航,以托管运行在可扩展、可排序表中的容器。

1.4 符号定义

文中涉及到的符号与意义,如表 1 所示。

表 1　符号意义

符号	定义
$T_{prometheus}$	Prometheus+监控异常所需时间
T_{sysdig}	Sysdig 监控异常所需时间
$T_{weave\ scope}$	Weave Scope 监控异常所需时间
T_{cpu}	CPU 异常所需时间
$T_{network}$	网络异常所需时间
T_{memory}	内存异常所需时间
T_{disk}	磁盘异常所需时间
T_{pull}	拉取镜像所需时间
T_{run}	镜像启动所需时间
T_{web}	网页加载所需时间
T_{mfile}	文件挂载时间
T_{cfile}	文件创建时间
$T_{csysdig}$	数据加载时间
T_{file}	文件加载总时间
T_{finish}	实验结束时间
T_{begin}	实验开始时间

2　监控系统

2.1　监控框架

Docker 的运维是一个体系,而监控在整个系统运维层面中的重要性是不能忽略的,监控系统可以对整个系统的运行状态进行实时展示,对系统中出现的故障进行提前预警与应急响应[14],因此需要对 Docker 容器运行时的各项性能指标进行实时监控。在监控的实现过程中,一般把容器看作宿主机上的一系列进程树。容器监控系统的数据采集并不像主流的监控系统的数据采集[6],主流的监控系统需要在目标机器上部署 agent 模块,通过 agent 模块来做数据采集,容器的监控系统针对容器虚拟化的特点,一般在容器的宿主机上对容器进行数据采集。监控的目的是监测和记录 Docker 容器各项性能的变化趋势,对 Docker 进行有效的环境管理。监控系统架构如图 1 所示。

agent 用来对节点监控的数据进行采集,由管控中心将监控项配置下发给 agent。通过 web 管理各种监控模型和视图的展示。然而,消息队列是一种高吞吐量的分布式发布订阅消息系统,主要处理用户在网站中的所有动作流数据,这些数据通常是由于吞吐量的要求而通过处理日志和日志聚合来解决。使用消息队列的目的是通过 Hadoop 的并行加载机制统一线上和离线的消息处理,为了通过集群提供实时的消息。可以用 analyser 来订阅消息队列,对数据进行分析处理、存储和报警。

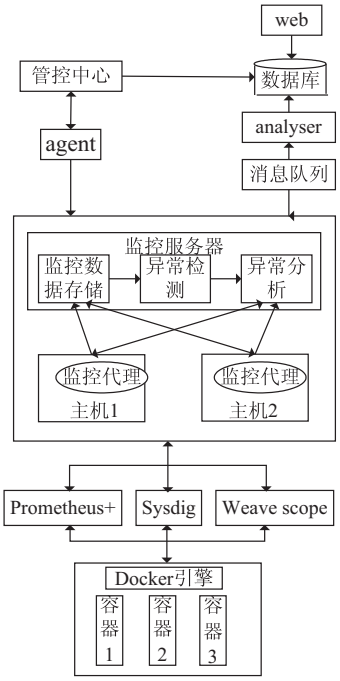


图 1　监控系统架构

监控服务器主要用来接收 agent 采集的监控数据,并统一存放到消息队列。主要由监控代理、监控数据存储、异常检测和异常分析四个部分组成。监控代理模块采用非侵入式的方式获取容器的资源利用率,监控数据存储模块负责接收来自各主机的监控数据,它只能存储最近一段时间的监测数据,并将数据组织成指定的格式,发送给异常检测模块。异常检测模块通过基于 iForest-based 的异常评估方法检测从监控数据存储模块接收到的数据,并将容器异常信息发送给异常分析模块。异常分析模块首先从每个主机获取异常容器的日志,然后对日志进行分析并定位异常原因。该监控系统主要研究 Prometheus+、Sysdig 和 Weave Scope 三种监控组件,采用该监控系统可对多主机下的 Docker 容器进行监控。

2.2　监控组件流程图

2.2.1　Prometheus+流程

图 2 显示 Prometheus+运行序列关系。首先,应用人员通过流程①启动虚拟平台,然后通过流程②向 Docker Hub 请求分别拉取 Prometheus、Node_exporter、Grafana 镜像,通过流程⑩返回拉取结果,拉取成功后通过流程③启动 Node-exporter,使用流程⑦返回启动的容器 ID;反之,拉取失败返回错误信息。接下来使用流程④在网页浏览器上访问端口,并且查看该镜像端口的连接状态;若已经建立连接,则通过流程⑧返回该镜像收集到的数据;若连接失败,则返回错误信息。

其次,执行流程②创建 prometheus 的目录和文件,创建成功后编辑配置文件,执行流程⑤docker run 命令启动该镜像,启动成功后查看端口的连接状态;若可以

返回该镜像的详细信息,则证明已经成功建立连接。若连接失败则返回错误信息。

第三,通过流程②新建一个空文件夹 grafana-storage,用来存储数据和编辑该镜像的配置文件。执行流程⑥启动该镜像并设置它的端口,通过流程④使用浏览器添加数据源到访问节点的端口,进行保存。保存成功后,通过流程⑨收集监控数据,从流程⑪返回收集到的信息。数据源配置成功后,物理机通过流程⑫可查看 Docker 中容器资源的使用情况和性能特征。综合以上,Prometheus+运行序列如图 2 所示。

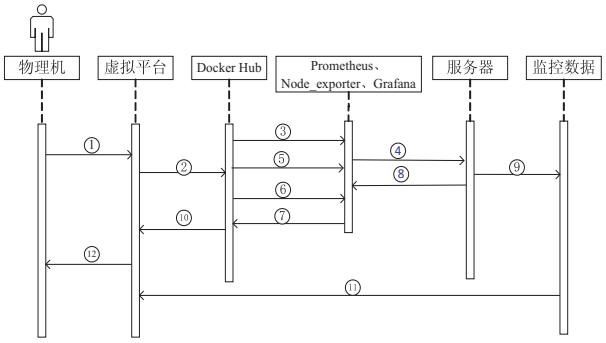


图 2 Prometheus+运行序列

根据 Prometheus+运行序列,发展 ProOPTime 算法。经由此算法,有效得出拉取全部镜像的时间、监控异常的全部时间、Prometheus+的运行时间以及监控组件操作的总时间。

算法 1:ProOPTime。

输入:n,m

输出:T

1. T_{begin} = clock( );

2. for i=1; i<n; i++ do

3. T_{pull} += P_i; //拉取全部镜像时间

4. end for

5. $T_{prometheus}$ = average(T_{cpu}) + average($T_{network}$) + average(T_{memory}) +average(T_{disk});

6. T_{run} = $T_{prometheus}$ + T_{web};

7. send web to user;

8. T_{finish} = clock( );

9. T = T_{pull} + T_{run}; //全部时间

10. return T

11. end

2.2.2 Sysdig 流程

图 3 显示 Sysdig 运行序列关系。首先,应用人员通过流程①启动虚拟平台,然后通过流程②向 Docker Hub 请求分别拉取 sysdig 镜像,拉取成功后通过流程③中的 docker run 命令运行该镜像,并设置 privileged 参数为 true,该参数使容器内部的 root 可以完全掌握操作系统所有的权限。参数设置完成后,容器执行流程④挂载/var/run/docker.sock 文件,该文件是 Docker 守护进程(Docker daemon)默认监听的 Unix 域套接字(Unix domain socket),容器中的进程可以通过流程⑧与 Docker 守护进程进行通信。

然后,使用流程⑤分别在/dev、/proc、/boot、/usr 和/lib/modules 目录下创建 dev 文件、proc:ro 可读文件、boot:ro 可读文件、usr:ro 可读文件和 modules:ro 可读文件。文件创建成功后通过流程⑨返回创建信息,并通过流程⑥收集监控数据,收集到的数据通过流程⑫传送给虚拟平台。最后,在流程⑦中运行 csysdig 命令,启动该监控工具的可视化界面,查看各项监控数据。Sysdig 运行序列如图 3 所示。

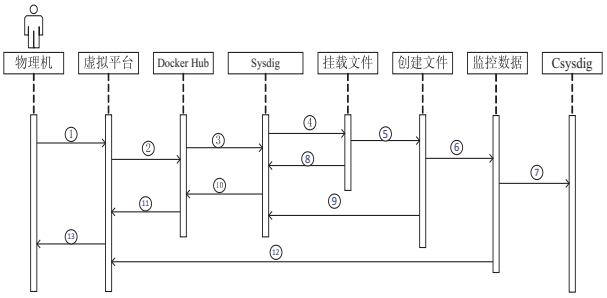


图 3 Sysdig 运行序列

从 Sysdig 的序列图,发展 SysOPTime 算法。由该算法,得到挂载和创建文件的总时间、监控异常情况的全部时间、sysdig 监控异常的时间与生成可视化数据时间等。

算法 2:SysOPTime。

输入:n

输出:T

1. T_{begin} = clock( );

2. T_{pull} = pull image in Docker Hub;

3. T_{file} = T_{mfile} + T_{cfile}; //挂载文件和创建文件的总时间

4. T_{sysdig} = average(T_{cpu}) + average($T_{network}$) + average(T_{memory}) + average(T_{disk}); //监控异常情况需要的总时间

5. T_{run} = T_{sysdig} + $T_{csysdig}$; //监控异常的时间与生成可视化数据时间之和

6. send csysdig to user;

7. T_{finish} = clock( );

8. T = T_{pull} + T_{file} + T_{run}; //全部时间

9. return T

10. end

2.2.3 Weave Scope 流程

图 4 显示 Weave Scope 运行序列关系。显示应用人员通过流程①启动虚拟平台,然后通过流程②向 Docker Hub 请求拉取 scope 镜像,结果由流程⑩反馈给虚拟平台。拉取失败则返回错误信息;拉取成功则通过流程③启动该容器,同时流程⑨返回该容器 ID。容器启动成功后,由流程④通过 curl -L https://github.com/weaveworks/scope/releases/download/latest_release/scope -o /usr/local/bin/scope 命令,通过流程⑤跟随 web 服务器重定向,完成后使用流程⑦在命

令行输出服务器返回的内容。所以,curl 请求 web 服务器成功后,使用 chmod a+x 命令对所有用户开启执行/usr/local/bin/scope 文件的权限。最后,重新启动 scope 容器,将流程⑥收集到的监控数据,通过流程⑪返回给虚拟平台。然后,通过流程⑫查看监控的结果。

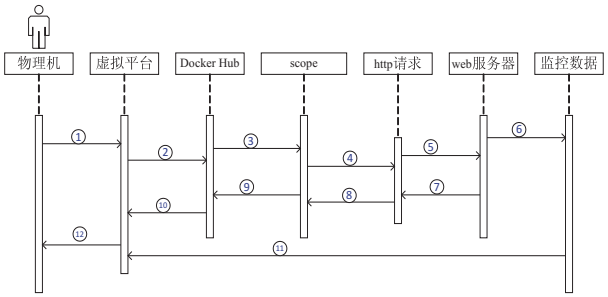


图 4　Weave Scope 运行序列

根据 Weave Scope 的运行序列,发展 WScopeOPTime 算法。此算法可以得到监控异常情况的全部时间、Weave scope 运行时间与构建网页时间之和以及该监控组件操作总时间。

算法 3:WScopeOPTime。
输入:n
输出:T
1. T_{begin} = clock( );
2. T_{pull} = pull image in Docker Hub;
3. $T_{weave\ scope}$ = average(T_{cpu}) + average($T_{network}$) + average(T_{memory}) + average(T_{disk}) ;　//监控异常情况需要的总时间
4. $T_{run} = T_{weave\ scope} + T_{web}$;　//运行时间
5. send web to user;
6. T_{finish} = clock( );
7. $T = T_{pull} + T_{run}$;　//总时间
8. return T
9. end

3　实验结果与分析

本次实验在 Linux 环境下搭建部署 Docker,并且安装和部署 Prometheus+、Sysdig、Weave Scope 三种监控组件。在三种监控组件监控 Docker 容器的过程中使用cAdvisor进行测试,测试三种监控组件在监控 Docker 容器各项性能时,它们的 CPU 占有情况、内存使用情况以及网络吞吐量的差别,分析其在监控 Docker 容器各项性能指标时监控组件对资源的使用情况,得出其在工作过程中各自的优势和差别。并对三种监控组件的监控功能的完善性度量进行性能分析。其次,通过给容器设置异常,来进行对比实验。由于没有容器异常注入的标准,将异常分为 CPU、内存、磁盘和网络四个涉及不同资源度量的常见类别。异常主要表现为 CPU 无限循环使用、内存泄漏和内存溢出、磁盘调度不当和网络攻击等现象。

3.1　实验环境

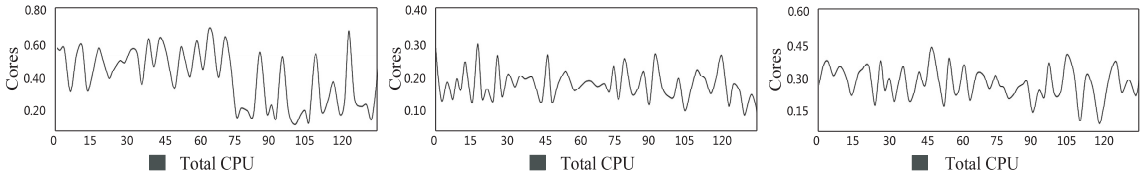
实验环境分成物理机和虚拟机两类,各自包含硬件与软件的配置。物理机硬件的配置为处理器 Intel(R) Core(TM) i7-9750H CPU @ 2.60 GHz 2.59 GHz,内存为 8.00 GB;软件的配置为 Windows 10,64-bit X86 系统,VMware Workstation Pro 的 12.5 版本。虚拟机内的硬件配置为 2 个处理器,硬盘为 20 GB,网络适配器为桥接模式,存在 USB 控制器;软件配置为 Ubuntu16.04 内存为 1 GB,Docker 为 20.10 版本。

3.2　实验结果

3.2.1　CPU 测试

图 5(a)、(b)和(c)分别显示 Prometheus+、Sysdig 和 Weave Scope 运行过程的 CPU 使用量。这三种监控组件在监控 Docker 容器时,CPU 的使用量波动幅度都比较大。其中,Prometheus+的波动最大,介于 0.1 核心与 0.7 核心之间,Sysdig 的 CPU 使用量波动比较小,介于 0.1 核心与 0.3 核心之间。依据式(1)得出,Prometheus+、Sysdig、Weave Scope 三种监控组件的平均 CPU 使用率分别为 20.9%、8.25% 和 15.1%。Prometheus+不仅波动幅度较大,而且 CPU 使用率最高,资源消耗最多。内核数(cores)显示为使用的 CPU 总核数,在本次实验中虚拟机使用的 CPU 总核数为 2 核心。监控组件在监控 Docker 容器的过程,CPU 性能测试的结果,如图 5 所示。

$$\overline{X_1} = \frac{\sum_{j=0}^{3} \frac{\sum_{i=30j+1}^{30j+30} \frac{x_i}{\text{cores}}}{30}}{4} \tag{1}$$



(a)Prometheus+　(b)Sysdig　(c)Weave Scope

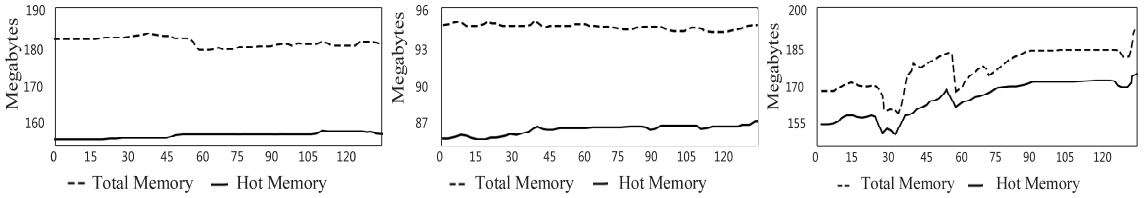
图 5　监控组件的 CPU 使用情况

3.2.2　内存测试

图 6 显示内存测试的结果,观察各种监控组件的内存使用量。图 6(a)、(b)和(c)分别显示 Prometheus+、Sysdig 和 Weave Scope 运行过程的内存使用量。在监

控 Docker 容器时 Prometheus+和 Sysdig 波动较小,几乎不变。Weave Scope 与它们相比较波动较大,波动范围在 140 Mb 与 185 Mb 之间。依据式(2)得出,Prometheus+、Sysdig、Weave Scope 三种监控组件的平均内存使用量分别为 168.4 Mb、90 Mb 和 164.7 Mb。由此可知,Prometheus+与 Weave Scope 两者内存的使用量大致相等,Sysdig 的内存使用量最少,比 Prometheus+少使用 78.1 Mb 的内存,比 Weave Scope 少使用 74.4 Mb 的内存。

$$\overline{X_2} = \frac{\sum_{j=0}^{3} \frac{\sum_{i=30j+1}^{30j+30} \frac{x_i + y_i}{2}}{30}}{4} \tag{2}$$



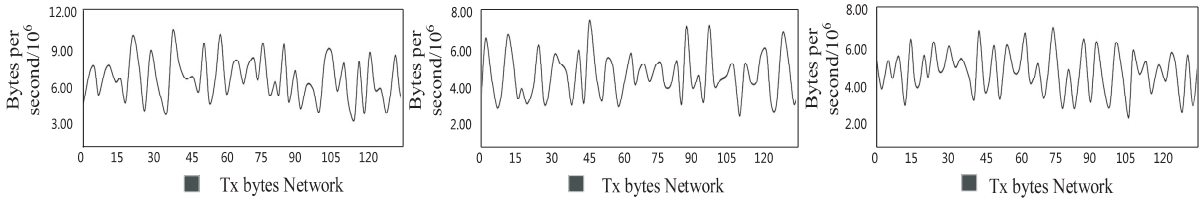
(a)Prometheus+ (b)Sysdig (c)Weave Scope

图 6 监控组件的内存使用情况

3.2.3 网络吞吐量测试

图 7 为吞吐量的测试分析图,显示各种监控组件的网络吞吐量。图 7(a)、(b)和(c)分别显示 Prometheus+、Sysdig 和 Weave Scope 运行过程的网络吞吐量。由此可知,三种监控组件在监控 Docker 容器时,网络吞吐量波动幅度都比较大,其中 Prometheus+波动较大,波动幅度介于 3.2 Mbps 与 9.2 Mbps 之间,Sysdig 与 Weave Scope 波动幅度大致相等。依据式(3)可得出,Prometheus+、Sysdig、Weave Scope 三种监控组件的平均网络吞吐量分别为 7.05 Mbps、5.1 Mbps 和 5.3 Mbps。Prometheus+网络吞吐量最大,数据传输效率最快,而 Sysdig 网络吞吐量最少。

$$\overline{X_3} = \frac{\sum_{j=0}^{3} \frac{\sum_{i=30j+1}^{30j+30} x_i}{30}}{4} \tag{3}$$



(a)Prometheus+ (b)Sysdig (c)Weave Scope

图 7 监控组件的网络吞吐量

3.3 实验分析

3.3.1 检测异常分析

四种模拟 Docker 容器面临的异常情况,说明如下:

● CPU 无限循环使用。通过插入额外的代码来调用压力工具[11],在应用程序中注入这个错误代码,程序由于异常进入无限循环模式,这可以模拟 CPU 中的无限循环,并占用 100% 的 CPU 利用率。

●内存泄漏。注入的代码在不释放对象的情况下分配堆内存,这会逐渐占用 100% 的内存利用率。

●磁盘输入/输出故障。使用可以产生很多线程或进程并执行用户指定的特定类型 I/O 操作的 FIO,注入额外的磁盘读写操作,模拟磁盘 I/O 故障[15]。

●网络拥塞。通过使用 wondershaper 来限制指定网络接口的带宽,从而模拟网络拥塞。

该文以检出率和误检率评价异常检测的结果。TP 为被正确分类的异常数量,FN 为未经识别的异常数量,FP 为将正常判断为异常的数量。

$$P_{\text{检出率}} = \frac{\text{TP}}{\text{TP} + \text{FN}} \times 100\% \tag{4}$$

$$P_{\text{误检率}} = \frac{\text{FP}}{\text{TP} + \text{FP}} \times 100\% \tag{5}$$

设计四种异常实验,明确监控组件对异常的检测精确度,并且使用各种监控组件对上述四种典型的异常进行测试。通过实验可知,Sysdig 在异常检测方面误检率低于其他监控组件,这是因为它的资源度量在正常负载下非常稳定。当出现异常时,监测数据的异常值变化很大,因此它可以更加准确地检测异常,具有较高的检测精确度。Prometheus+的资源度量在正常负载下波动不小,有时连续波动会导致异常值上升超过异常检测阈值,导致错误检出。在磁盘输入输出故障注入的情况下,Weave Scope 的检测率明显低于 Prometheus+,这是因为异常磁盘读写速率与波动较小的正常磁盘读写速率相差不大。因此,监测数据的局部密度变化很小,Weave Scope 的异常检出率很低。

3.3.2　监控系统分析

综上所述,在运行 Prometheus+时,CPU 的利用率、内存的使用量和网络的吞吐量,都要高于 Sysdig 和 Weave Scope。相较于 Prometheus+和 Weave Scope,Sysdig 安装步骤较为简便,安装一次即可永久部署监控系统。此外,在监控的过程中,Sysdig 的 CPU 使用量、内存使用量和网络吞吐量的使用量都是最少的。由于此工具在可视化界面上显示的是实时数据,缺点是无法查看容器各项性能指标的变化和趋势。同时,只能以命令行进行操作,相比于 Prometheus+增加用户操作的复杂性。

对比于 Prometheus+和 Sysdig,Weave Scope 能够对多个容器或主机进行监控,通过生成容器地图看到各个容器之间的关系和显示请求公网的行为。同时,能够分别根据 Docker 容器中各个进程和镜像的各项性能指标,查看当前时刻的 CPU 使用量、内存使用率、网络吞吐量以及文件使用情况等各项性能指标,缺点是无法知道 Docker 容器在不同时间各项性能指标的变化情况。

4　结束语

将 Prometheus+、Sysdig 和 Weave Scope 组件构成监控系统,并对其进行性能测试,得出各项资源随时间变化的关系曲线。其次,根据监控数据的特点,构建四种容器异常,包含 CPU 异常、内存异常、磁盘异常以及网络异常等,对监控系统的异常检测精确度进行验证。实验结果发现,Weave Scope 可以监控多个容器。Prometheus+对资源需求的依赖比较高,优点是它可以长期存储监测的数据,并且生成警告机制。Weave Scope 和 Sysdig 一样,对资源的需求比较少,缺点是只能显示实时数据。未来的监控系统应该朝向部署容易,同时对多个主机和容器进行监控。当受到外界攻击或自身发生错误时,能够自动生成警报机制。

参考文献:

[1] TOMAR A,JEENA D,MISHRA P,et al. Docker security:a threat model, attack taxonomy and real-time attack scenario of DoS[C]//2020 10th international conference on cloud computing,data science & engineering (confluence). Noida, India:IEEE,2020:150-155.

[2] CINQUE M,CORTE R D,PECCHIA A. Microservices monitoring with event logs and black box execution tracing[J]. IEEE Transactions on Services Computing,2019,7(99):1-1.

[3] CHEN J,LIU J,XIAN M,et al. Docker container log collection and analysis system based on ELK[C]//2020 international conference on computer information and big data applications (CIBDA). Guiyang, China:[s. n.],2020:317-320.

[4] WANG Y,XUE B,WANG L,et al. Iterative anomaly detection[C]//2017 IEEE international geoscience and remote sensing symposium (IGARSS). Fort Worth, TX, USA: IEEE,2017:586-589.

[5] MOHALLEL A A,BASS J M,DEHGHANTAHA A. Experimenting with docker:Linux container and base OS attack surfaces[C]//2016 international conference on information society (i-Society). Dublin:[s. n.],2016:17-21.

[6] JIANG Y,LIU W,SHI X,et al. Optimizing the copy-on-write mechanism of docker by dynamic prefetching[J]. Tsinghua Science and Technology,2021,26(3):266-274.

[7] HUANG D, CUI H, WEN S, et al. Security analysis and threats detection techniques on docker container[C]//2019 IEEE 5th international conference on computer and communications (ICCC). Chengdu, China: IEEE, 2019: 1214-1220.

[8] 鲁　涛,陈　杰,史　军. Docker 安全性研究[J]. 计算机技术与发展,2018,28(6):115-120.

[9] 王　鹃,胡　威,张雨菡,等. 基于 Docker 的可信容器[J]. 武汉大学学报:理学版,2017,63(2):102-108.

[10] SULTAN S,AHMAD I,DIMITRIOU T. Container security:issues,challenges, and the road ahead[J]. IEEE Access, 2019,7:52976-52996.

[11] ZOU Z,XIE Y,HUANG K,et al. A docker container anomaly monitoring system based on optimized isolation forest[J]. IEEE Transactions on Cloud Computing,2019,10(1):134-145.

[12] BHATIA G,CHOUDHARY A,DADHEECH K. Behavioral analysis of docker swarm under DoS/DDoS Attack[C]//2018 second international conference on inventive communication and computational technologies (ICICCT). Coimbatore:[s. n.],2018:985-991.

[13] TANSANGWORN N. Development of IoT edge hub for wireless sensor networks based on docker container[C]//2020 IEEE international conference on smart internet of things (SmartIoT). Beijing,China:IEEE,2020:356-357.

[14] 肖　遥, 朱志祥. 一种基于 Docker 的监控系统的设计与实现[J]. 计算机与数字工程,2019,47(11):2919-2925.

[15] BHIMANI J,YANG Z,MI N,et al. Docker container scheduler for I/O intensive applications running on NVMe SSDs [J]. IEEE Transactions on Multi-Scale Computing Systems, 2018,4(3):313-326.