·实用技术·

基于 Web 浏览器的远程容器登录系统设计

陈 霄 郭志川 孙 鹏 朱小勇

(中国科学院声学研究所 国家网络新媒体工程技术研究中心 北京 100190
中国科学院大学 北京 100049)

摘要:针对现有远程登录容器的方案中便利性差、权限管理较难实现、安全性较低等问题,本文设计了一种新的基于 Web 浏览器的远程容器登录系统。该系统利用 Websocket 和 Docker HTTP API 技术实现了通过浏览器端远程登录容器的功能。首先浏览器向控制中心发送 Websocket 连接请求,然后控制中心通过 Docker HTTP API 与 Docker Daemon 建立通信链路。链路建立后用户通过在网页上输入命令来在容器内进行操作。本系统解决了在基于容器技术的 PaaS 平台的应用场景下,用户登录容器操作的需求,并且在易用性和安全性上较其他方案有显著的改善。

关键词:容器,Docker,浏览器,云平台,Websocket,Web 控制台

Design of Remote Container Login System Based on Web Browser

CHEN Xiao, GUO Zhichuan, SUN Peng, ZHU Xiaoyong

(National Network New Media Engineering Research Center, Chinese Academy of Sciences, Beijing, 100190, China,
University of Chinese Academy of Science, Beijing, 100049, China)

Abstract: Aiming at the problems of exists methods such as the poor convenience, the difficult rights management and the low security; the paper designed a Remote Container Login System Based on Web Browser. By using Websocket and Docker HTTP API technology, the system implement the function of logging in to the container remotely in the browser. First, the browser sends a Websocket connection request to the control center, and then the control center establishes a communication link with the Docker Daemon via the Docker HTTP API. After the link is established, the user operates in the container by entering a command on the web page. The method solves the need to login a container in a container – based PaaS platform application scenario with significant improvements in ease – of – use and security over other methods.

Keywords: Container, Docker, PaaS, Web Browser, Websocket, Web Console

1 引言

云计算是目前广泛应用的一种计算模式。它是以虚拟化技术为基础,以网络为载体提供基础架构(IaaS)、平台(PaaS)、软件(SaaS)等服务为形式,同时可以整合大规模可扩展的计算、存储、数据、应用等分布式计算资源进行协同工作的超级计算模式[1]。云计算技术近些年不断发展,PaaS(平台即服务)云平台[2]的规模也在不断扩大。在这种趋势下,以 Docker 为代表的容器技术应运而生,在云平台场景下的虚拟化领域被广泛应用[3]。容器是基于操作系统内核的一种轻量级的虚拟化技术。其可以类比于虚拟机,但其本身并不是虚拟机,而是一种更轻量的虚拟化解决方案[4]。

本文于 2017 – 10 – 17 收到。

Docker 是一个容器级虚拟化解决方案。也是在 Linux 平台上的一款轻量级虚拟化容器的管理引擎[5]。整个项目是一个基于 go 语言开发的开源项目。对开发人员来说,Docker 提供了简单快速的隔离环境,无需再为配置环境操心。而对运营维护人员来说,Docker 可以快速构建出一整套无状态分布式基础设施,使得运维工作可以通过程序化的脚本来完成。

在传统的虚拟机场景中,每个用户都可以通过平台管理系统,根据自己被分配的权限,登录某些机器的某些账号。而当应用部署逐渐转移到基于容器构建的 PaaS 平台[6]之后,让用户登录进容器进行查看、调试应用就成为了 PaaS 平台的一个重要且必备的功能。

远程登录容器功能的传统实现方式是基于虚拟机的思想,在每个容器中启动一个 SSH 进程。在这种多容器的使用场景中,用户通过 SSH – Client 指定容器的 IP 远程连接到容器,让用户感觉到自己好像就在使用虚拟机。但是,这种方案会带来一些问题。首先是便利性较差。用户需要单独的 SSH 客户端以及登录信息。增加了使用平台的额外成本和复杂度,在使用的便利性上存在一定的问题。其次是权限管理。比如如何控制哪些用户能够登录哪些容器;如何和平台已有的权限管理系统集成。这些功能往往都需要通过堡垒机系统控制[7]。而在云平台中引入单独的堡垒机系统又会增加平台的复杂度以及维护成本。最后是登录方式的选择。无论使用密码还是私钥验证登录,容器内的密码或者密钥的管理都需要通过加入额外的程序解决,无疑会增加容器的复杂度。同时还要面对同权限容器的密码或密钥的一致性问题。

针对以上问题,本文设计了一种基于 Web 浏览器的远程容器登录系统。其基本思想是基于 Websocket 协议与 Docker HTTP API 实现了通过 Web 浏览器远程进入容器的功能。将其集成到基于容器的云计算平台中,可以很大地提高 PaaS 平台的可用性和安全性。为开发和运维人员及其他使用者提供了一种稳定又便捷的进入容器的方案。

2 通过 web 浏览器进入容器的解决方案

本系统设计目的在于解决在容器化的云平台上,运营和开发人员通过浏览器直接进入容器进行操作的需求。其对于发送命令的实时性、命令处理结果反馈的实时性都有着较高的要求,同时整个连接链路的可靠性也需要有一定的保障。本文基于此设计了包含控制客户端、控制中心、Docker Daemon、Docker 容器在内的远程登录方案。总体架构如图 1 所示,其中核心部分是容器控制中心,它的上游利用 Websocket 协议与浏览器通信,下游利用 Docker Daemon 的 Docker HTTP API 接口与容器通信,最终可以让使用者在 Web 浏览器上以类似 SSH 的方式登录并操作 Docker 容器。

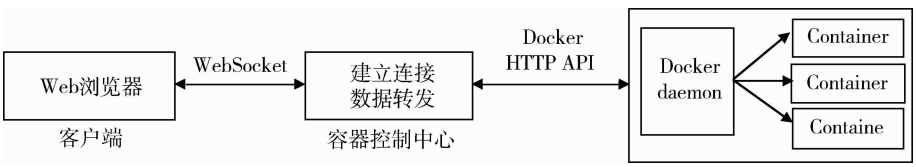

图 1 方案架构图

2.1 容器控制客户端

为了方便基于网页的客户使用,我们设计并开发了基于 HTML5 的网页客户端。客户端的主要功能有 Websocket 连接请求的发送、监听键盘输入以及 Websocket 返回的字节流。除此外还要将远端的标准输出和标准错误传送到本地客户端的标准输出和标准错误。

整个客户端有两部分组成:用户登录部分和 Web 控制台。其中用户登录部分即让使用者输入身份信息、宿主机 IP、容器 ID 等作为验证,此部分可接入权限管理系统,使得不同权限的使用者在容器中以不同的账户进行操作。Web Console 部分是我们利用 Java Script 开发的一个 Web 控制台界面,其中连接部分利用了 Websocket for JS 开源软件库,和下游建立 Websocket 长连接。Web 控制台则是基于 Xterm 开源项目进行开发,实现网页上的命令行界面。使用者经过登录认证之后,可进入到相应的容器中进行操作。操作方式是

在网页上输入命令并可以实时得到命令执行反馈。

2.2　基于 Websocket 的通信连接

本系统选择了 Websocket 协议作为客户端和控制中心间的连接协议。该协议可以支持客户端与远程主机之间进行全双工通信。用于此的安全模型是 Web 浏览器常用的基于原始的安全模式。协议包括一个开放的握手以及随后的 TCP 层上的消息帧。该技术的目标是为基于浏览器的、需要和服务器进行双向通信的(服务器不能依赖于打开多个 HTTP 连接)应用程序提供一种通信机制[8]。在实现 Websocket 连接过程中,需要通过浏览器发出 Websocket 连接请求,然后服务器发出回应,这个过程称为“握手”。在 Websocket API 的支持下,浏览器和服务器只需要做一个握手的动作,然后,浏览器和服务器之间就形成了一条快速通道。两者之间就可以直接互相传送。

2.3　容器控制中心

容器控制中心是整个方案的核心模块。对于上游客户端它是 Web 服务器,对于下游的 Docker Daemon 它又是请求的发起者。除了这种类似转发的功能外,它还承担了用户权限控制部分的功能。其主要功能包括:调用 Docker HTTP API、通过 Websocket 传递字节流、对用户登录信息的验证和鉴权等。将用户登录和鉴权功能交由控制中心执行可以有效的减轻容器的负载和复杂度,让容器可以以无状态的形式向外提供服务,便于迁移和启停。

整个控制中心模块和 Docker 项目一样使用 Go 语言进行开发,与 Docker 项目的兼容度非常高。同时,Go 语言在高并发场景下的优化使得控制中心可以接受大量的长连接请求。除此之外,我们在模块的开发中使用了 Gorilla 的 Websocket 开源库和 Docker Client 客户端,可以实时将使用者发送的命令通过持久化连接转发给下游,并将反馈消息实时发送回客户端。

2.4　Docker 远程调用接口

控制中心与容器的连接是通过 Docker 远程调用接口来实现的。Docker 使用客户端 - 服务器 (C/S) 架构模式,其中 Docker daemon 作为服务端接受来自客户的请求,并处理这些请求(创建、运行、分发容器)。客户端和服务端既可以运行在一个机器上,也可通过 socket 或者 RESTful API 来进行通信。Docker daemon 一般在宿主主机后台运行,等待接收来自客户端的消息。Docker 客户端则为用户提供一系列可执行命令,用户用这些命令实现跟 Docker Daemon 交互。

客户端使用 Docker 远程调用接口来管理和创建 Docker 容器,本方案中的控制中心模块就是请求了 Docker 远程调用接口以获取和容器之间的 TCP 长连接,并用 Hijack 的方式[9]对连接中的标准输入输出流进行转发和操作。值得注意的是,出于安全考虑,本系统中的 Docker HTTP 远程端口只暴露控制中心模块所部署的服务器。具体实现使用 Iptables 对端口加以限制。

2.5　系统运行过程

该部分结合附图及具体实施例对本文介绍的系统运行细节做进一步的阐述。需要注意的是所描述的过程是在用户信息认证和鉴权之后的过程,由于用户权限控制在不同应用场景下会有不同的需求和实现,在此不加以叙述。

整个过程如图 2 所示,基于 Web 浏览器的远程容器连接系统,其运行过程包括:

(1)Web 浏览器向控制中心服务端发送 HTTP 请求,并通过 Websocket 协议和控制中心建立通信链路;具体包括:Web 浏览器运行 JS 脚本,向控制中心服务端发送包含参数的 HTTP POST 请求,其中 POST 请求包含参数应包括目标容器宿主机 IP 地址、请求发送端 IP、目标容器 ID 和登录目标容器的用户

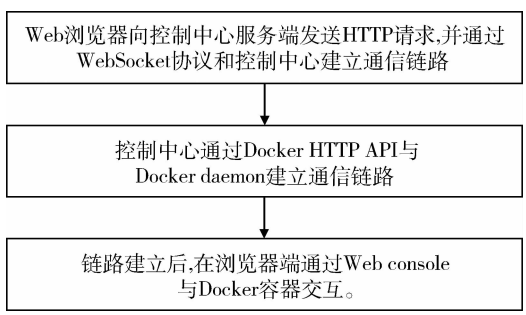

图 2　基于 Web 浏览器的容器登录系统运行过程图

名。具体 JSON 内容如下：

```
{"username": "#user_name",
"containerid": "#container_id",
"src_vmaddr": "#src_vmaddr",
"dst_vmaddr": "#dst_vmaddr"
}
```

浏览器得到控制中心的请求返回，展现控制中心所返回的界面。所展现的页面包含通过 JS 脚本实现的命令行控制台界面，具体实现利用了 JS 的开源框架 Xterm. js。

(2)控制中心通过 Docker HTTP API 与 Docker Daemon 建立通信链路；具体包括：目标容器暴露一个端口给外部，使得容器可以被远程访问；控制中心向容器暴露的端口发送请求调用 Docker HTTP API 中的接口，与 Docker Daemon 通信，并通过 Docker Daemon 进入到容器中；调用的 Docker HTTP API 接口具体为：依次调用 ExecCreate、ExecAttach 接口。

(3)链路建立后，用户通过在 Web 浏览器输入字符与 Docker 容器交互；具体包括：控制中心将调用 ExecAttach 接口后返回的数据流绑定到建立好的 Websocket 连接上，其中输入流绑定到 Websocket 读操作，输出流绑定到 Websocket 写操作。Golang 代码中具体实现方式为：启动两个协程，分别绑定读写数据流。在同一个 Websocket 连接上进行读和写的操作。

最后使用者在网页端通过键盘操作与 Docker 容器直接交互。

3 实验对比分析

目前主流的登录容器的方案有以下三种：SSH 登录、第三方工具(如 Nseneter, Nsinit)登录以及 Docker 自己提供的工具。其中 SSH 登录方式适用于远程登录容器，后两种方法只适用于通过 Docker 宿主机登录容器。表 1 描述了三种方案各自的特点。

表 1 登录容器方案对比

连接容器的方式	SSH 登录	第三方工具(Nsenter、Nsinit)	Docker 提供的工具(attach、exec)
操作方式	传统 SSH 登录操作	利用第三方工具指令操作	使用 attach(exec)登录容器，exit 退出容器
优点	符合平时登录服务器习惯，不用额外学习	使用相对方便快捷	使用相对方便快捷
缺点	密钥管理不易实现； SSH 升级复杂； 操作不易监控	需要学习第三方工具使用规则； 需要 Docker 宿主机 root 权限	需要 Docker 宿主机 root 权限； 需要同屏操作
适用范围	适用远程登录容器	适用 Docker 宿主机登录至容器内部	适用 Docker 宿主机登录至容器内部

3.1 SSH 远程登录方案比较

这种远程登录容器的方法是类比于虚拟机的一种实现。首先需要一个 SSH 客户端，然后为每个容器配置自己的 IP，并在容器中启动 SSH 进程。用户在客户端指定容器 IP 并输入密钥即可远程登录进入容器。这种方法在虚拟机上十分通用，但是到了容器应用场景会出现很多问题。首先，容器本身不具有单独的网卡和 IP，需要额外搭建容器网络。同时，在云平台场景下，容器会经常销毁和迁移。这个特点使得容器 IP 很容易发生漂移，为基于 IP 的 SSH 远程登录造成很大的困难。其次，这种登录方式将 SSH 的权限和密钥管理程序[10]都放到容器中执行，加重了容器本身的负载，不符合轻量级虚拟化的需求。并且作为无状态的容器如何存储密钥等信息也是很大的问题。

本文所设计系统与传统 SSH 登录，两种方案都实现了远程登录容器的功能，但是传统的 SSH 登录方案在安全性，和便捷性上都存在很大的问题。相比而言，本设计利用浏览器即可登录容器，不需要单独的客户端，提升了便利性；系统利用中间控制模块进行用户权限管理，在提高了系统安全性的同时保证了容器本身的轻量化和无状态化。相比于传统 SSH 方案有很大的改进。表 2 描述了两种方案的各自特点。

表 2　与 SSH 登录方案对比

方案	客户端	网络	安全	可用性
SSH 登录	单独客户端	构建容器网络	容器内安全管理	较差
本设计方案	浏览器	与宿主机共享	容器外控制	较好

3.2　其他方案比较

对于上文提到的其他两种方案，均只支持从宿主机登录容器。在多机多容器的云平台场景下，同一台宿主机上可能有多个完全不相关的容器。让没有其他容器服务权限的用户从宿主机进行登录操作，无疑是非常危险的行为。由此可见，与本设计相比，从宿主机登录的方案在安全性上并无法满足基于容器的云平台场景下的基本需求。

通过比较分析可以看出，目前登录容器的主流方案在多机多容器的云平台服务场景下有着很多的局限性。相比而言，本设计较好的满足了云平台的需求，为基于容器的云平台服务提供了很好的工具支持。

4　结束语

本文设计了一种基于 Web 浏览器的容器登录系统，解决了在容器化的云平台上，运营和开发人员通过浏览器直接进入容器进行操作的需求。同时利用 Websocket 和 Docker HTTP API 等技术使得整个系统的可用性和稳定性得到了保障。可以一定程度上支持多用户多连接的并发操作。和现有技术方案相比，本设计在便利性和安全性上有较大提高，可以应用于 PaaS 层的云平台中。后续将继续定制本系统中描述的权限控制模块，并在多连接场景下的并发性能和稳定性能上进行改进和研究。

参　考　文　献

[1] 吴吉义，平玲娣，潘雪增，等. 云计算：从概念到平台[J]. 电信科学，2009，25(12)：1－11

[2] Beimborn D，Miletzki T，Wenzel D W I S. Platform as a Service (PaaS)[J]. Wirtschaftsinformatik，2011，3(6)：381－384

[3] Liu D，Zhao L. The research and implementation of cloud computing platform based on docker[C]//International Computer Conference on Wavelet Active Media Technology and Information Processing. IEEE，2015. 475－478

[4] Dua R，Raja A R，Kakadia D. Virtualization vs Containerization to Support PaaS[C]//IEEE International Conference on Cloud Engineering. IEEE，2014. 610－614

[5] Merkel D. Docker：lightweight Linux containers for consistent development and deployment[J]. 2014，2014(239)：76－91

[6] Pahl C. Containerization and the PaaS Cloud[J]. IEEE Cloud Computing，2015，2(3)：24－31

[7] 宗波. 浅析堡垒机概念及工作原理[J]. 计算机光盘软件与应用，2012(18)：124－124

[8] Fette I，Melnikov A. The WebSocket Protocol[EB/OL]. 2011. http://tools.ietf.org/html/rfc6455.

[9] 许式伟，吕桂华. Go 语言编程：The Go programming language[M]. 北京：人民邮电出版社，2012.

[10] Williams S C. Analysis of theSSH Key Exchange Protocol[C]//Cryptography and Coding－，Ima International Conference，Imacc 2011，Oxford，Uk，December 12－15，2011. Proceedings. DBLP，2011. 356－374

作者简介

陈霄，男，(1992－)，硕士研究生，研究方向：嵌入式系统、容器虚拟化技术。

郭志川，男，(1975－)，博士，副研究员，研究方向：网络新媒体、智能终端技术等。

孙鹏，男，(1976－)，博士，研究员，研究方向：信号与信息处理。

朱小勇，男，(1982－)，博士，副研究员，研究方向：嵌入式系统，多媒体技术。