容器云平台网络资源配置管理系统的设计

冯文超

(,兰州 730070)

摘要: IaaS , , 、 。 , 。 , , Linux , , 。

关键词: ; ;

中图分类号:TP393.07 **文献标识码:**A **文章编号:**1000 - 0682(2018)01 - 0070 - 04

Design of network resource management and configuration system based on container cloud

FENG Wenchao

(*School of Electronics and Information Engineering*, *Lanzhou Vocational Technical College*, *Lanzhou* 730070, *China*)

Abstract: Container has more advantage in application deployment in cloud platform, however, there are great challenges in network isolation and resource management on container cloud. To handle the problems, network resource management and configuration system based on container cloud is proposed. In container cloud, the physical network is divided into multiple virtual network for tenants to use, each tenant get an isolated virtual network, at the same time, the network control interface provided by linux are used to manage network resource usage of each container. By using distributed configuration management and autonomous configuration, the network resource management and configuration on container cloud platform is implemented.

Key words: container cloud; network management; network configuration

0 引言

云计算已经成为数据中心提高资源利用率,降低企业运维成本的有效手段,当前大规模应用的云计算,如亚马逊的 AWS,阿里巴巴的阿里云等,都是 IaaS[1]的公有云平台。虽然云平台极大地降低了系统运维的成本,但在应用部署发布方面,应用程序厂商依然面临着巨大的版本维护成本。以 Docker[2]为代表的容器技术的发展,为应用程序的开发带来了极大的便利,Docker 具有“一次发布,到处运行”的优势,因此以 Docker 容器为典型代表的 PaaS 平台[3]逐渐成为云平台发展的新趋势。Kubernetes[4]作为当前最为成熟的容器管理工具,在网络配置管理方面,其扁平化网络管理方式使得该系统在网络隔离、资源控制方面有所不足。为了解决上述问题,该文以 Kubernetes 为基础,提出了容器云网络资源配置管理方案。将云平台底层物理网络划分成多租户使用的虚拟网络,隔离不同租户之间的虚拟网络,同时利用 Linux 系统提供的网络控制功能对容器的网络资源进行控制,使用分布式配置管理等技术,实现容器云平台大规模网络资源的配置和管理。

收稿日期:2017 - 06 - 07

作者简介:冯文超(1979),男,甘肃兰州人,硕士研究生,讲师,研究方向为计算机网络技术,图形图像。

1 系统设计

1.1 Kubernetes 容器云平台网络模型

在 Kubernetes 容器云平台中,Pod 是容器管理的基本单元,每个 Pod 中至少包含一个容器。与 Docker 默认的网络管理模式不同,Kubernetes 并不直接对容

器分配动态的 IP 地址,而是为容器管理的基本单元 Pod 分配 IP 地址,当 Pod 获取到 IP 地址后,该 Pod 内部所有的容器共享该 IP,且 Pod 内所有容器都可以获取到 Kubernetes 为 Pod 分配的端口资源。

跨节点通信最典型的技术是覆盖网络,最具代表性的实现是 VXLAN[5],VXLAN 是在传统传输层网络基础上构建出的虚拟逻辑覆盖网络技术。VXLAN 主要是通过使用“MAC – in – UDP”封装。如果分布在 2 台不同物理设备上的 Pod 之间的 Docker 通信,需要将原来要发送的数据包作为 UDP 包的数据,然后将 UDP 数据包在不同的物理设备之间通过物理网络进行传输,接收端通过解析 UDP 数据包,将内容转发到目的 Pod 的容器中。

1.2　系统总体设计

根据系统需求,系统主要需要满足容器云平台的虚拟网络构建、自动化配置以及网络隔离功能,因此在实现上,系统通过接收 Kubernetes 发出的创建 Pod 及其对应的网络的相关信息,如 Pod 中容器的数量、IP 地址、虚拟子网号等信息后,将其自动化地配置在各个工作节点上,由于不同的 Pod 都有单独的虚拟子网号,不同虚拟子网由于 VXLAN 而相互隔离,因此不同 Pod 之间的租户网络之间是相互隔离的。各 Pod 在分配到不同的虚拟子网中之后,就可以对各个租户网络的网络资源使用量进行分配和管理。系统总体架构如图 1 所示。

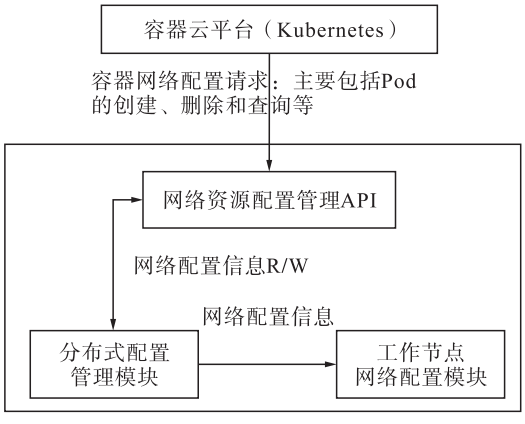

图 1　系统整体架构图

从功能上划分,系统主要包括 3 个部分。

1)网络资源配置管理 API:主要负责系统的对外服务模块,具体来说,该模块监听部署在服务器上的一个端口,容器云平台 Kubernetes 需要创建新的网络配置请求,首先连接该端口,并以一定固定的协议格式将网络配置请求的具体内容发送到该端口,网络资源配置管理 API 接收到这些信息后,将其解析,获取具体的网络配置操作命令,主要包括 Pod 节点的创建、删除和查询时,系统对容器云的网络相关的配置。

2)分布式配置管理模块:该模块主要负责 3 个功能,接收网络资源配置管理 API 发送过来的具体网络配置信息,然后将这些信息以分布式的方式存储在各运行节点上以避免单点故障,最后接受工作节点网络配置模块的对某个网络资源配置的访问。这些网络配置信息持久化地存储在数据管理数据库中,这些数据以 key – value 的方式组织。为了保证系统网络配置的可靠性,配置的存储系统采用分布式存储,每个存储节点都可以利用该模块共享网络拓扑和配置信息,管理后台以保证所有节点都能顺利完整地获取配置信息的一致性。

3)工作节点网络配置模块:该模块是系统的核心功能模块,主要负责实现由 Kubernetes 容器云平台发送过来的各种网络配置命令的实现。该模块定期轮询存储在各工作节点上的网络配置存储系统中的网络配置信息,然后针对不同的租户网络,检查该租户网络是否发生改变,如果有租户网络发生更新,则自动化的完成覆盖网络的建立和更新,并完成相关租户网络配置等工作。

网络资源配置管理 API 接收到创建虚拟网络的请求,然后对命令内容进行解析,获取到该虚拟网络的主要信息,如虚拟子网的名字,网络的 IP 地址范围以及该网络的出口带宽等各种属性信息,这些信息会存储在分布式配置管理模块中,工作节点网络配置模块会查询并根据相关内容创建相应的网络接口。

1.3　网络资源配置管理 API 设计

网络资源配置管理 API 的工作流程如图 2 所示。

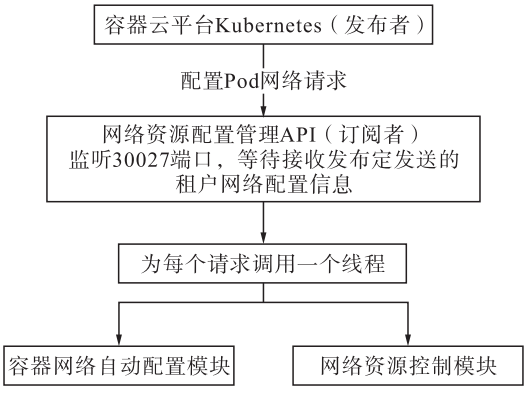

图 2　网络资源配置管理 API 工作流程

为了提高消息在不同的 Kubernetes 平台之间高效地传输,并且实现消息的广播,网络资源配置管理 API 采用消息队列 ZeroMQ[6],网络资源配置管理

API 采用发布者 - 订阅者模式,当该系统启动之后,网络资源配置管理 API 监听 30027 端口上等待,接收到相关信息之后,网络资源配置管理 API 将这些信息解析并存储到分布式配置存储模块中,等待工作节点网络配置模块读取配置信息,并最终在各工作节点上构建租户网络。

网络资源配置管理 API 中,发布者和订阅者之间通信的基本协议格式如下:

```
struct SzmqMsgHeader
{
    int m_iMsgLen;  //仅指 Data 的长度
    int m_iMsgType;
};
    //后面紧接着 data 部分
```

当 Kubernetes 创建 Pod 成功之后,则其通过调用网络插件对 Pod 的容器进行网络配置。网络资源配置管理 API 接收到这个请求后,调用相应的容器网络配置执行参数进行处理,其参数的格式如下:

```
struct SZmqConnection
{
    int m_iIDLen;  // Container ID 的长度
    string m_ID;
    int m_iNameLen;  // Container Name 的长度
    string m_Name;
    int m_iPIDLen;  // Container PID 的长度
    string m_PID;
    int m_iRequestIP;  // Container 指定的 IP 地址
    int m_iNetwork;  // Container 的网络地址
    int m_iOvsPortID;  // Container 连接的 Ovs 端口号 ID
    int m_iBandwidth;  // Container 的网络带宽
    int m_iDelay;  // Container 的网络延迟
    int m_iRxTotal;  // Container 的接收的字节数
    int m_iTxTotal;  // Container 的发送的字节数
    int m_iRxRate;  // Container 的接收速率
    int m_iTxRate;  // Container 的发送速率
    int m_iConnDetailLen;  // Container 的连接具体信息长度
    string m_ConnDetail;
    };
```

1.4 分布式配置管理模块的设计

1.4.1 分布式配置信息同步机制

分布式配置管理模块包含分布式存储模块和后台监控模块,存储模块主要负责存储网络配置信息,而运行在各个工作节点上的后台监控模块主要负责各租户网络的管理,包括创建、查询和修改等功能。

在容器云平台的每个工作节点上都运行着网络资源配置管理模块,主要负责各个工作节点上的网络配置工作,由于配置管理模块分布在各个工作节点上,可以有效地避免单点故障。在系统中,所有的配置信息是以 key - value 的方式存储,由于配置信息分布存储在各个节点上,该模块需要为其他模块提供查询和存储配置信息的接口,且各分布式节点在后台保证配置信息的一致性。

为了保证各个节点上信息的一致性,运行在各个工作节点中的后台模块必须采取同步措施,保证网络拓扑信息的完整性和一致性。该系统采用的是 Raft 同步算法[7],在该算法中,每个工作节点都可以有两种角色:Client 和 Server,但在任一时刻只能是 Server 或者 Client,同时一个集群内任意时刻可能有多个 Server。当工作节点为 Client 时,其主要工作只有调用 RPC[8]将本机更改的网络拓扑结构发送给 Server。对于 Server 来说,其工作较为复杂,除了响应 Client 的 RPC 操作,维护集群节点状态等,Server 还需要参与 Raft 算法的 leader 竞选操作。当集群中 leader Server 选举出来之后,由 leader Server 负责处理其他组件发送过来的查询和事务操作。分布式配置信息同步工作的架构如图 3 所示。

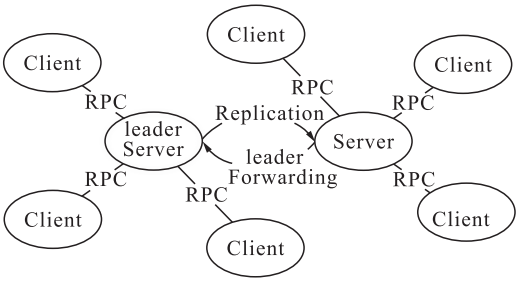

图 3 分布式配置信息同步机制

1.4.2 网络配置信息设计

为了实现租户之间的隔离,需要为不同租户分配不同的虚拟子网,Kubernetes 作为容器云平台的管理系统,必须可以支持不同租户之间虚拟子网的建立和分配,因此在 Kubernetes 中,网络配置需要主要包括 3 种:租户虚拟子网信息,虚拟子网 VNI[9]位向量信息以及虚拟子网 IP 位向量信息。

租户虚拟子网是 Kubernetes 为租户与其他租户之间实现网络隔离而创建的虚拟专用网络,该网络类似于一个单独的二层的局域网,在分布式信息存储中,key - value 存储的模式为:< 虚拟子网名字,具体信息 >,其结构如下:

```
struct TenantNetwork
{
    string m_szName;   //租户虚拟子网的名字
    string m_szSubnet; //租户虚拟子网的地址范围
    string m_szGateway;  //租户虚拟子网的网关
    uint_8 m_iVNI;
};
```

虚拟子网 VNI 位向量信息在分布式信息存储中的格式如下：<vlanStore,位向量>,在位向量中,每个 bit 表示一个 VNI,当某个 bit 为 1 时,表明该 VNI 已经被某个租户虚拟子网所使用。当创建某个 VNI 时,需要将对应的 bit 位置 1,如果删除该 VNI,将其置 0。

虚拟子网 IP 位向量信息的 key 为虚拟子网的名字,value 是位向量,每个 bit 代表一个 IP 地址,如果 bit 为 1 表示该 IP 被使用,否则,该 IP 空闲。

1.5　网络自动配置模块的设计

网络自动配置模块通过后台配置管理模块获取到租户网络的配置信息,然后在各工作节点为租户自动构建网络连接,其主要工作包括 3 部分:1)构建并维护覆盖网络;2)构建并维护租户的虚拟子网;3)自动配置容器网络。图 4 是一个典型的容器集群网络结构,4 个节点组成一个局域网络,该物理网络中包含 2 个虚拟子网 vnet0 和 vnet2,各个网络通过 OVS 连接,然后通过物理网络交换机相互连接。

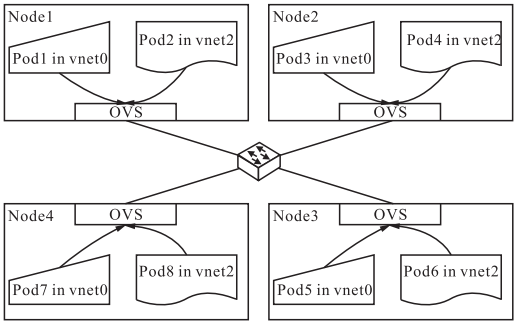

图 4　容器集群网络结构

1.5.1　覆盖网络的构建和维护

覆盖网络的目的就是在地层物理网络上为租户拓展一个二层的虚拟网络,以便于云平台的大规模网络扩展。如前文所述,VXLAN 是当前比较成熟且支持厂商最多的技术。

由于容器云平台中拥有众多工作节点,且工作节点会因为各种原因主动加入或退出云平台,因此需要动态调整云平台的覆盖网络,该系统利用故障检测工具 Serf[10] 来实现该功能。当一个工作节点申请加入该覆盖网络时,需要向集群中的 leader Server 发送注册信息,注册完成后,集群中的其他节点在 leader Server 节点的辅助下,使用 gossip 协议获取集群中的成员列表,同时自动与新加入的节点建立 VXLAN 连接。具体来说,网络自动配置模块利用 Serf 提供的集群管理功能,为每个节点定义一个回调函数,该回调函数的主要目的就是更新集群内的覆盖网络,当每次有节点注册或注销时,都会调用该函数重新组织新的集群覆盖网络。

1.5.2　虚拟子网自动管理功能设计

虚拟子网自动管理模块在工作节点的后台运行,通过调用分布式配置管理模块提供的网络配置管理接口,检查虚拟网络列表中状态已经发生改变的虚拟子网,如网络的新增、更新和删除等,虚拟子网自动管理的处理流程如图 5 所示。

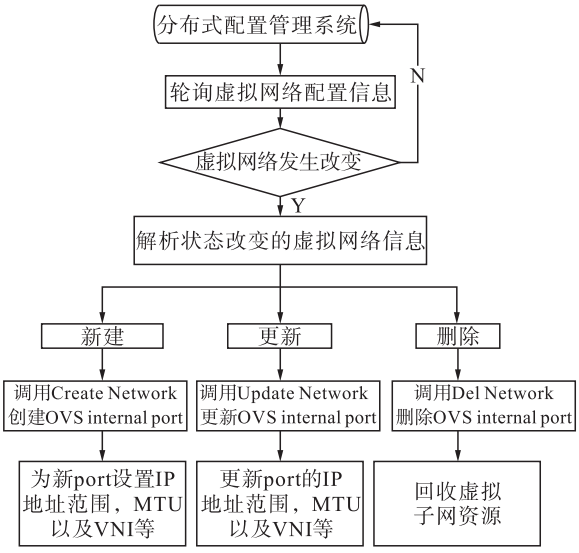

图 5　虚拟网络自动管理处理流程图

1.5.3　容器网络自动配置功能设计

在 Kubernetes 中,对容器网络的自动配置就是对容器所在 Pod 的 infra 容器的网络进行配置。infra 容器的网络配置信息包括 Pod 的虚拟子网名、容器的 PID、VNI 等信息,具体的配置流程如图 6 所示。

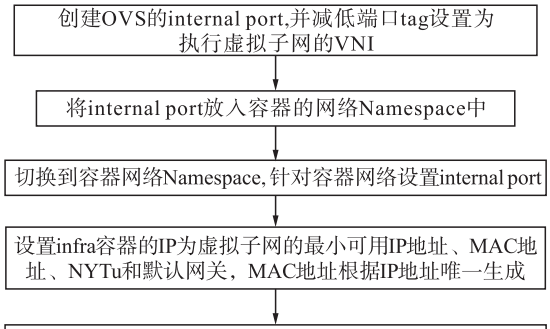

图 5　容器网络自动配置流程图

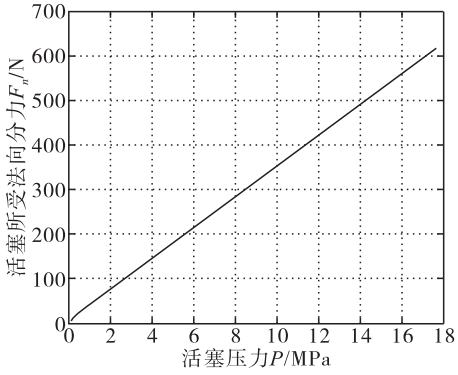

图 8 活塞所受法向力与活塞气体压力关系图

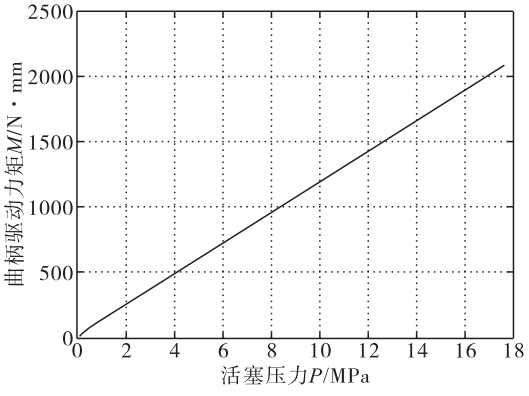

图 9 曲柄驱动力矩与活塞压力关系图

曲柄驱动力矩与活塞压力关系曲线如图 9 所示。从图 9 可以看出,在忽略惯性力矩和摩擦力矩的情况下,曲柄所需驱动力矩与活塞压力近似成正比例关系,当活塞压力为 15 MPa 时,所需驱动力矩近似为 1700 N · mm。

5 结论

该文从理论上对手动气体压力源曲柄滑块机构运动学、动力学关系进行了分析,建立了运动学、动力学数学模型,并采用数学分析软件 MATLAB 对所建立的数学模型进行了仿真分析。通过运动学仿真曲线分析,可以清楚地看出活塞位移、速度、加速度与曲柄转角的关系,对该曲柄滑块机构的进一步优化具有一定的理论依据。通过动力学仿真曲线分析,可以清楚地看出活塞组件、连杆组件以及曲柄驱动力矩的变化趋势,对具体零部件强度设计提供了载荷参考。

参考文献:

[1] 易俊. 内嵌压力源全自动便携式压力校验仪的研究[D]. 南京:南京理工大学,2007.

[2] 成大先,王德夫. 机械设计手册. 单行本. 常用设计资料[M]. 北京:化学工业出版社,2004:1 – 8.

[3] 陈德为. 曲柄滑块机构的 MATLAB 仿真[J]. 太原科技大学学报,2005,26(3):172 – 175.

[4] 郁永章,蒋培正,孙嗣莹,等. 压缩机工程手册[M]. 北京:中国石化出版社,2011:173 – 175.

(73)

2 结束语

容器云作为一种效率更高的虚拟计算平台,其在网络隔离性和扩展性方面的不足限制了其大规模的工业化应用。该文以容器云的网络资源配置为研究对象,提出了以 VXLAN 技术保证容器网络隔离,并使用基于 VXLAN 构建覆盖网络解决网络扩展性的问题,同时通过自动化配置 Kubernetes 的 infra 容器实现网络配置的自动化,最大限度地保证容器云网络的自动化配置。

参考文献:

[1] 斯进. 面向 IaaS 云服务的云系统中虚拟机监控及证据采集[J]. 现代电子技术,2016,39(4):86 – 88.

[2] 李杰,刘广钟. Hadoop 分布式集群的自动化容器部署研究[J]. 计算机应用研究,2016,33(11):3404 – 3407,3445.

[3] 魏豪,周抒睿,张锐,等. 基于应用特征的 PaaS 弹性资源管理机制[J]. 计算机学报,2016,39(2):223 – 236.

[4] 杜军. 基于 Kubernetes 的云端资源调度器改进[D]. 杭州:浙江大学,2016.

[5] 王永建,张健,张富根,等. 基于 VXLAN 的云数据中心网络研究[J]. 通信技术,2017,50(1):78 – 83.

[6] 张蓓蓓,张松,李京,等. 基于 ZeroMQ 的电子侦察系统数据通信平台的设计与实现[J]. 小型微型计算机系统, 2016,37(3):520 – 525.

[7] 张晨东,郭进伟,刘柏众,等. 基于 Raft 一致性协议的高可用性实现[J]. 华东师范大学学报(自然科学版), 2015(5): 172 – 184.

[8] 许卓明,栗明,董逸生,等. 基于 RPC 和基于 REST 的 Web 服务交互模型比较分析[J]. 计算机工程,2003,29(20):6 – 8.

[9] 杜涛. 基于 BP 神经网络技术的网络流量预测模型[J]. 网络安全技术与应用,2016(7):55,57.

[10] Serf Introduction[EB/OL]. https://www. serfdom. io/intro /index. html