****

本科毕业设计（论文）

（2019届）

题目：基于云平台的传统业务管理方案

**学 院** **信息工程与艺术设计学院**

**专 业**  **软件工程**

**学 号** **2015b11023**

**学生姓名**  **徐喜东**

**指导教师** **梁曦**

**提交日期** **2019 年 5 月 24 日**

声明及论文使用的授权

本人郑重声明所呈交的论文是我个人在导师的指导下独立完成的。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写的研究成果。

论文作者签名： 年 月 日

本人同意浙江水利水电学院有关保留使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以上网公布全部内容，可以采用影印、缩印或其他复制手段保存论文。

论文作者签名： 年 月 日

摘 要

该篇文章主要是探究在云平台上部署MHA架构的方案的选择和对比。

众所周知，软件需要部署在服务器上，而服务器的架构和生产息息相关，因此，我希望能构建一套较为成熟的架构，能够在今后的学习和生活中使用。

在数据库的选择上，我跟从了当前国内由阿里巴巴主导的去IOE的趋势，选择了MySQL数据库。而在实际生产环境中，要避免数据库服务器单点故障的可能性，因此对数据库选择搭建MHA的高可用集群架构。

在部署方面，因为从初始状态部署MHA高可用集群流程会比较长，涉及到的配置文件和字段相对来说较多，配置起来较为复杂繁琐，因此我给出了两套部署方案，目的是能够实现快速部署MHA环境，并且实现可拓展性、兼具跨平台的特点。

目前，虚拟技术的使用非常广泛，是节省硬件资源、管理服务和集群的有效手段。为了能够快速部署MHA架构，我决定在两种不同的虚拟化技术——docker容器技术与VM虚拟化技术之间选择一种较为优秀的方案。

**关键词**：Linux系统，MHA集群，docker容器

Abstract

This article mainly explores the choice and comparison of deployment schemes of MHA architecture on cloud platform.

As we all know, software needs to be deployed on servers, and the architecture and production of servers are closely related. Therefore, I hope to build a more mature architecture that can be used in future learning and life.

In terms of database selection, I followed the current trend of de-IOE led by Alibaba in China and chose MySQL database. In the actual production environment, in order to avoid the possibility of single point failure of database server, we choose to build MHA high availability cluster architecture for database.

In terms of deployment, because deploying MHA high-availability cluster from the initial state will take a long time, involving relatively more configuration files and fields, and the configuration is more complicated and cumbersome, I give two sets of deployment schemes. The purpose is to achieve rapid deployment of MHA environment, and to achieve scalability and cross-platform characteristics.

At present, virtual technology is widely used, which is an effective means to save hardware resources, manage services and cluster. In order to rapidly deploy the MHA architecture, I decided to choose a better solution between two different virtualization technologies——docker container technology and VM virtualization technology.

**Key words:** Linux system, MHA cluster, docker container

目录

[摘 要 I](#_Toc8985811)

[引言 1](#_Toc8985813)

[第一章 背景介绍 2](#_Toc8985814)

[1.1 概念论述 2](#_Toc8985815)

[1.1.1 集群的定义 2](#_Toc8985816)

[1.1.2 MHA简介 2](#_Toc8985817)

[1.1.3 MHA组成 2](#_Toc8985818)

[1.1.4 MHA工作过程 2](#_Toc8985819)

[1.1.5 Docker概述 2](#_Toc8985820)

[1.1.6 Docker发展 3](#_Toc8985821)

[1.1.7 Docker容器技术和虚拟机对比 3](#_Toc8985822)

[1.1.8 Docker核心技术 4](#_Toc8985823)

[1.1.9 Docker特性 4](#_Toc8985824)

[1.1.10 NFS工作原理和工作流程 5](#_Toc8985825)

[1.1.11 NFS相关配置 5](#_Toc8985826)

[1.2 本章小结 6](#_Toc8985827)

[第二章 方案设计 7](#_Toc8985828)

[2.1 设计思路 7](#_Toc8985829)

[2.1.1 总述 7](#_Toc8985830)

[2.1.2 MHA集群物理条件 7](#_Toc8985831)

[2.1.3 MHA拓扑设计 7](#_Toc8985832)

[2.1.4 准备集群环境 8](#_Toc8985833)

[2.1.5 部署容器环境 9](#_Toc8985834)

[2.1.6 对比实验和分析 9](#_Toc8985835)

[2.2 本章小结 9](#_Toc8985836)

[第三章 基础环境部署 10](#_Toc8985837)

[3.1 创建虚拟机 10](#_Toc8985838)

[3.1.1 总述 10](#_Toc8985839)

[3.1.2 下载镜像 10](#_Toc8985840)

[3.1.3 新建虚拟机 10](#_Toc8985841)

[3.1.4 给虚拟机安装系统 13](#_Toc8985842)

[3.1.5 模板化虚拟机 13](#_Toc8985843)

[3.2 安装mysql 19](#_Toc8985844)

[3.2.1 创建快照"模板v1.1"并创建连接克隆mysql51 19](#_Toc8985845)

[3.2.2 以mysql51为例，建立mysql软件仓库 20](#_Toc8985846)

[3.3 安装docker 22](#_Toc8985847)

[3.3.1 获取软件包 22](#_Toc8985848)

[3.3.2 配置yum,安装docker 22](#_Toc8985849)

[3.4 初始化mysql配置 23](#_Toc8985850)

[3.4.1 查看mysql初始密码 23](#_Toc8985851)

[3.4.2 修改mysql密码策略与登录密码 23](#_Toc8985852)

[3.4.3 添加远程登录权限 24](#_Toc8985853)

[3.4.4 生成mysql51快照，并以做链接克隆52-56 24](#_Toc8985854)

[3.5 本章小结 24](#_Toc8985855)

[第四章 镜像的封装使用和MHA部署 25](#_Toc8985856)

[4.1 mysql镜像 25](#_Toc8985857)

[4.1.1 封装mysql镜像 25](#_Toc8985858)

[4.2 搭建 MHA主库 25](#_Toc8985859)

[4.2.1 软件包安装 25](#_Toc8985860)

[4.2.2 制作MHA镜像 26](#_Toc8985861)

[4.2.3 用主库文件生成mysql52、mysql53目录 27](#_Toc8985862)

[4.3 搭建MHA集群从库 28](#_Toc8985863)

[4.3.1 创建mysql54从库目录 28](#_Toc8985864)

[4.4 搭建MHA基础环境 28](#_Toc8985865)

[4.4.1 免密配置 28](#_Toc8985866)

[4.5 配置MHA-Manager管理主机 29](#_Toc8985867)

[4.5.1 创建一个新的MHA容器，制作MHA-manager镜像 29](#_Toc8985868)

[4.6 部署MHA环境(其二) 31](#_Toc8985869)

[4.6.1 给所有主机安装nfs-utils工具 31](#_Toc8985870)

[4.6.2 配置mysql51上的NFS 31](#_Toc8985871)

[4.7 搭建并部署自定义镜像仓库 33](#_Toc8985872)

[4.7.1 在mysql51上搭建自定义镜像仓库 33](#_Toc8985873)

[4.7.2 在mysql52-mysql56上使用私有镜像 33](#_Toc8985874)

[4.8 部署MHA环境(其二) 34](#_Toc8985875)

[4.8.1 部署主库mysql51 34](#_Toc8985876)

[4.8.2 部署备份主库mysql52和mysql53 34](#_Toc8985877)

[4.8.3 部署从库mysql54/mysql55 35](#_Toc8985878)

[4.8.4 部署MHA管理机mysql56 36](#_Toc8985879)

[4.8.5 脚本布置 37](#_Toc8985880)

[4.9 使用通常虚拟机进行MHA部署 38](#_Toc8985881)

[4.9.1 停止容器并启动各个节点的mysqld服务 38](#_Toc8985882)

[4.9.2 MHA管理节点进行健康检查 38](#_Toc8985883)

[4.10 实验中出现意外的问题 39](#_Toc8985884)

[4.10.1 问题描述 39](#_Toc8985885)

[4.10.2 问题探究和讨论 39](#_Toc8985886)

[4.10.3 问题的解决 39](#_Toc8985887)

[4.11 关于Docker的泛用性讨论 40](#_Toc8985888)

[4.11.1 MHA数据节点不能部署在docker平台上 40](#_Toc8985889)

[4.11.2 docker技术的泛用性讨论 41](#_Toc8985890)

[4.11.3 docker镜像启动过程中出现的问题 41](#_Toc8985891)

[4.12 本章小结 41](#_Toc8985892)

[第五章 实验结果论述 42](#_Toc8985893)

[5.1 关于Docker和VM虚拟化的对比 42](#_Toc8985894)

[参考文献 43](#_Toc8985895)

[致谢 44](#_Toc8985896)

引言

当前时代，数据膨胀，万物上云。随之而来的，是架构的变迁和优化。

本文希望能够在正式参与工作之前，搭建一套能够快速部署、承载高访问量，且拥有可拓展性的一套数据库架构，在日后的生产和生活当中能够使用。

当前数据库分为传统数据库与NoSQL数据库，在校学习中NoSQL学习相对较少，因此选择了搭建传统数据库，且在国内去IOE的一种趋势下，选择了MySQL体系。

一套完整而全面的生产架构体系涉及颇多，包含了堡垒机、审计系统、Git服务器、认证服务、时间同步、LVS转发调度、Nginx转发调度、负载均衡、存储共享、存储集群、数据库集群、数据缓存集群、ELK日志集群与监控集群等多项服务(如图0-1)，并且包含网络相关架构。在这里，只对MySQL集群进行讨论。

本文最后想要达到的效果，是通过镜像技术快速部署能承载高访问的高可用的MHA集群，做到能够快速部署数据库平台解决传统的问题。其中涉及到的技术有：Docker镜像、MySQL主从、MySQL高可用。

因为架构需要用到多台机器联合处理，因此使用VM技术来进行环境的模拟，并在其上进行部署。

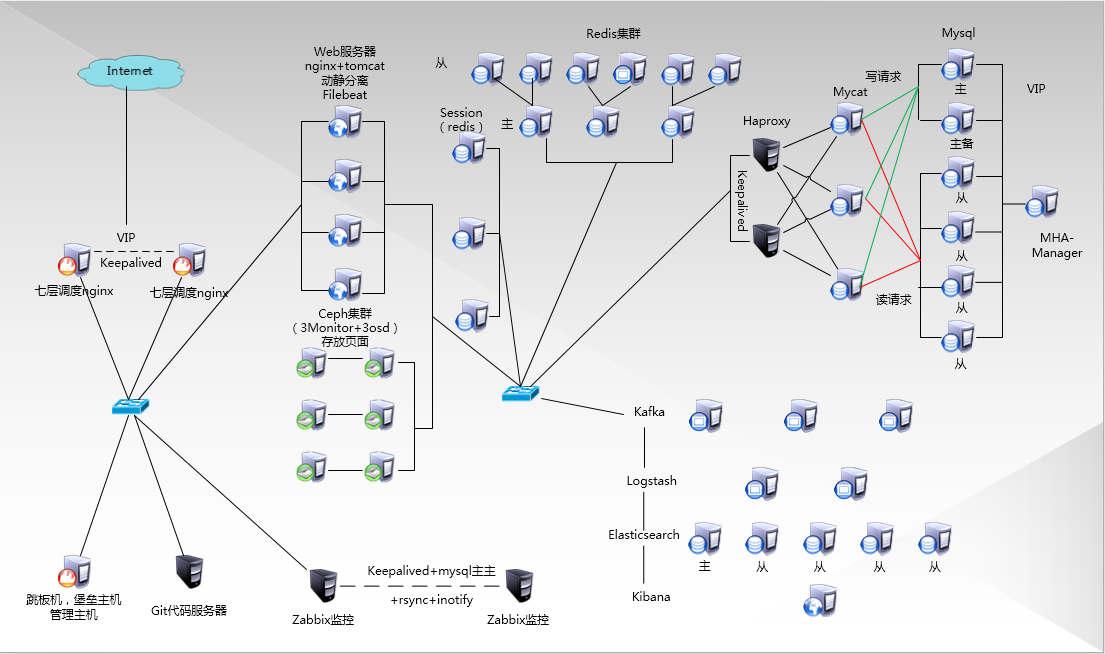


图0-1 完整架构示意图

1. 背景介绍
   1. 概念论述
      1. 集群的定义

集群是指多台服务器一起提供相同或不同的服务。

* + 1. MHA简介

MHA全称为Master High Availability，由日本DeNA公司youshimaton使用perl语言开发，是一套优秀的可以用来实现MySQL服务器的高可用的解决方案，MHA对于MySQL数据库的自动故障切换操作能做到在0~30s之内，并且能确保在故障切换过程中保证数据的一致性，以达到真正意义上的高可用。

* + 1. MHA组成

MHA由MHA Manager(管理节点)和MHA Node(数据节点)组成，其中Manager可以单独部署在一台独立的机器上也可以部署在一台slave节点上，而Node节点运行在每台MySQL服务器上。

* + 1. MHA工作过程

MHA在工作的时候，由Manager定时探测集群中的master节点，当master故障时，Manager会自动将拥有最新数据的slave提升为新的master。在这个过程中，有以下几个关键点：

1. 从宕机崩溃的master保存二进制日志事件
2. 识别含有最新更新的slave
3. 应用差异的中继日志( relay log )到其他的slave
4. 应用从master保存的二进制日志事件
5. 提升一个slave为新的master
6. 使其他的slave连接新的master进行复制

通过这个过程，我们可以使MySQL集群避免单点故障的问题，防止一台数据库服务器宕死导致的整个架构瘫痪。

* + 1. Docker概述

Docker 是一个开源的、可供多平台一起使用的，部署在Linux机器上的应用容器引擎，Docker让开发者可以打包他们自己经过不断努力写出来的应用、以及依赖包到一个可以被转移的、拥有高可移植性的、可靠的容器中，然后发布到任何当前社会上流行并在广泛使用中的 Linux 机器上，除此之外，Docker也可以通过自己的方式实现虚拟化。容器完全使用了sandbox沙盒机制，可以在执行后进行回滚，容器与容器之间不会有任何接口（类似 iPhone 的 app）。这样的特性下，容器几乎没有性能开销，可以很容易地在各个地点、各个不同的机器和数据中心中运行。最重要的是,容器既不依赖于任何语言，也不依赖于任何框架或者依赖任何的包装系统。

* + 1. Docker发展

Docker 是 dotCloud 公司经过各种思考和尝试之后进行研发的一个开源的一个基于 LXC 的高级容器引擎，源代码托管在 Github 上, 基于 go语言并遵从 Apache2.0 协议开源。它让开发者可以打包他们的应用以及依赖包到一个可移植的 container 中，然后发布到任何流行的 Linux 机器上。

* + 1. Docker容器技术和虚拟机对比

相同点：docker容器技术和虚拟机技术，都是虚拟化技术。

不同点：

1. 虚拟化结构：

Server(硬件)

Host OS(windows/centos)

Hyperviser(VMWare/KVM)

Guest OS(虚拟机操作系统)

Bins/Libs(库)

Apps

1. 容器结构：

Server（硬件）

Host OS(centos7)

Docker Engine

Bins/Libs(库)

Apps

相比之下，Docker容器技术和VM虚拟化技术之间相差一个GuestOS，占用资源更少，效率更高，Docker使用当前系统内核。我们知道一台物理系统启动时间要10秒左右。而Docker实例启动时间可以控制在1~2s。

* + 1. Docker核心技术

Docker的核心技术是NameSpace和Cgroup，NameSpace实现Container的进程、网络、消息、文件系统和主机名的隔离，而Cgroup实现对资源的配额和度量，其中Cgroup的配额可以指定实例使用的cpu个数，内存大小等，就像VMWare虚拟机中的硬件配置参数一样，方便好用。

* + 1. Docker特性

Docker的特性有：文件系统隔离、资源隔离、网络隔离、日志记录、变更管理和交互式shell。

其中，文件系统隔离使每个进程容器运行在一个完全独立的根文件系统里，这样的系统可以存在于任何目录下。而资源隔离通过cgroup可以让我们的系统资源，如CPU和内存等，可以分配到不同的容器中。网络隔离通过虚拟接口和虚拟IP地址使每个进程容器运行在自己的网络空间。

而日志记录指Docker将会收集和记录每个进程容器的标准流(stdout/stderr/stdin)，用于实时检索或批量检索。同时，容器文件系统的变更可以提交到新的镜像中，并可重复使用以创建更多的容器，无需使用模板或手动配置，我们称这为变更管理。最后，Docker可以分配一个虚拟终端并关联到任何容器的标准输入上，例如运行一个一次性交互shell。

通过以上论述我们不难看出。docker相对于虚拟化，拥有相似而不同的特性。它们在不同的情景下，拥有不同的优缺点：

1. 优点：

1.Docker也是一种虚拟技术，因此一些优势和VM一样，但不是所有都一样。它比VM小，比VM快，Docker容器的尺寸减小相比整个虚拟机大大简化了分步到云和从云分发时间和开销。启动一个容器实例的时间很短，一两秒就可以启动一个实例。这样也就决定了它可以用于程序封装，我们启动一个镜像，相当于是启用一个程序。

2.对于在笔记本电脑里，数据中心的虚拟机，以及任何的云上，运行相同的没有变化的应用程序，Docker可以做到让IT的发布速度更快。它是是一个开放的平台，可以用来让我们快速构建，发布和运行分布式应用程序，并使应用程序能够快速从组件组装和避免开发和生产环境之间的摩擦。

3.我们可以在部署公司局域网的时候或在使用虚拟机的时候使用它，这样可以大大加快我们的部署效率。

4.在实际工作中，我们的开发人员并不关心后台具体是哪个Linux系统，使用Docker部署的话，我们的开发人员可以根据所有依赖关系，针对他们所选择的操作系统，构建相应的软件。然后在部署时，一切是完全一样的，因为一切都在DockerImage的容器在其上运行。开发人员负责并能够确保所有的相关性得到满足。这使前后端在工作中的协调性得到大大提升。

5.Docker目前在世界范围内有大量的支持。Google，微软，亚马逊，IBM等都支持Docker的使用。

6.Docker支持Unix/Linux操作系统，也支持Windows或Mac，我们可以将任何需要的东西进行封装。

1. 缺点和局限性：

Docker用于应用程序时是非常有效的，但这不包含数据。日志，跟踪和数据库等通常应放在Docker容器外，一个容器的镜像通常都很小，不适合存储大量的数据，存储可以通过外部挂载的方式使用。比如使用NFS，ipsan，MFS等，我们可以通过-v选项映射磁盘分区。

* + 1. NFS工作原理和工作流程

1. NFS工作原理

NFS的工作依赖于三个必须的进程，rpc.nfsd、rpc.mountd和portmap，在工作的过程中，NFS服务器还可能需要启动其他进程，如rpc.lockd、rpc.statd等。

1. NFS的工作流程

在NFS服务的工作过程中、客户端首先向NFS服务器提交RPC请求，并明确访问NFS服务，并告知具体的读写操作。此时，RPC开始查找端口映射表，并根据客户端请求，返回相应的结果。最后，客户端获取指定的NFS服务器端口后，直接与NFS进程进行读写操作。

* + 1. NFS相关配置

1. nfs相关文档

"exports"文件是NFS服务的主配置文件，它位于Linux中的"/etc"目录下，和其他服务的主配置文件一样，绝大部分的配置都是通过编辑该文件完成。"xtab"文件位于"/var/lib/nfs"目录，这个文件主要用来记录客户端与NFS服务器的连接记录。如果想查看哪些客户端曾经连接过NFS服务器，查看该文件即可。

1. 常规服务器架设流程

NFS服务的常规架设流程为：

1.客户端访问NFS服务器的共享目录。

2.NFS服务器查看exports文件查看目录是否被共享，以及共享目录所设置的权限。

3.如果客户端经过审核有权限查看共享目录，NFS服务器则将共享目录内容反馈给客户端。

NFS服务端发布共享目录的格式为："共享目录 [客户端1参数][客户端2参数]"，其中包含两个关键点：共享目录和客户端。客户端的指定方式为使用IP地址指定单一主机，也可指定一个网段。在配置过程中，我们会使用到很多参数，其中"ro"为设置共享权限为只读，"rw"为设置共享权限为读写，而当使用NFS服务器共享目录的使用者是root时，nfs会默认进行降权，为了不被映射成为匿名账号，需要使用"no\_root\_squash"。

在数据保存方面，我们有两种方式可供选择，一种是保持数据同步，也就是将数据同步写入内在和硬盘，选择这种方式需要使用"sync"参数，这可能导致效率降低。而另一种是先将数据保存在内存中，而不是直接保存在硬盘中，对应的参数为"async "，但这种方式对意外的应对较差，当出现断电和死机的情况时，极有可能出现数据丢失。

* 1. 本章小结

本章主要是对实验中用到的各种工具和技术进行理论介绍。

1. 方案设计
   1. 设计思路
      1. 总述

在这一章中，我阐述自己的实验设计思路。

我们在部署这样的架构的过程中，需要相应的物理条件，因此我们暂定6台机器，因为没有真正的服务器在手里可供调用，因此我只能在自己的笔记本电脑上跑虚拟机。本次的实验均在虚拟机上实现。

在此之前，我曾在KVM上做过MHA的部署并取得了成功，当时每台Node节点为1核512MB内存。而此次因为要用到Docker容器技术，我将Node节点机器的内存提到了1024MB。

在有了物理环境基础后，开需要布署集群环境，因为我们最后需要6台服务器协同工作，而并非彼此独立。

当集群环境部署完毕后，开始出现分歧。首先，我们要做的是在虚拟化和docker上制作，对比两者效率，因此我们需要将环境部署两次，一次在机器上部署，另一次在docker中部署。

部署完环境后，我们就可以开始测试集群是否能够正常运行。然后对比两种方式对资源的损耗。

* + 1. MHA集群物理条件

MHA的部署需要六台虚拟机作服务器，其中一台虚拟机作Manager管理节点，每台虚拟机内存1024MB，CPU为1，且虚拟机需要部署CentOS 7.4系统和MySQL服务，因为最终将环境封装入镜像，且数据库服务器一般工作在内网中，因此可以使用root权限搭建环境。因为物理机上部署过mysql环境，因此客户端可以由物理机来充当。

* + 1. MHA拓扑设计

1. mysql51：Master主节点服务器，同时作为docker模板制作机器，私有docker镜像服务器，ftp服务器以及nfs共享服务器，用来存放mysql目录文件。其他机器使用的mysql数据库中的数据全存在此服务器上，用来模拟真实环境中的存储集群。
2. mysql52：备用1主节点服务器
3. mysql53：备用2主节点服务器
4. mysql54：第一台slave服务器
5. mysql55：第二台slave服务器
6. mysql56：MHA\_manager服务器
7. VIP地址：192.168.1.100
8. client：用来访问和测试的客户端

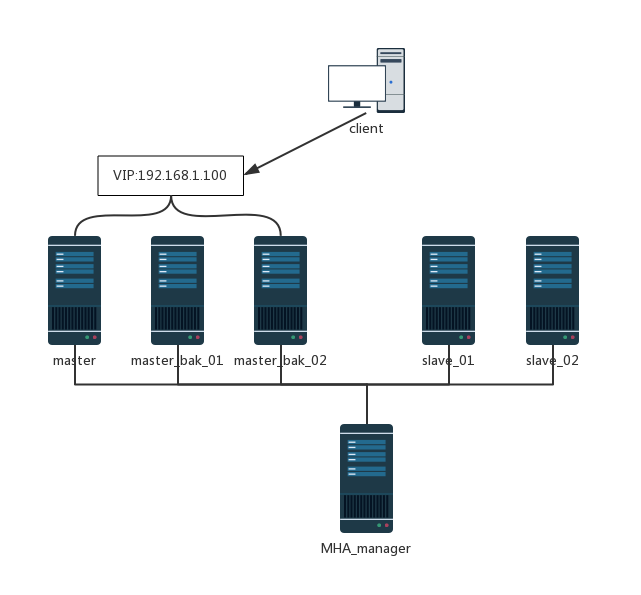


图2-1 MHA拓扑图

* + 1. 准备集群环境

因为需要用到多台机器，并且机器具有一定的相通性，于是决定利用COW技术制作一台模板机器，将此机作为后端盘模板，其他机器皆由此机产生，既省去了挨个部署的时间，又相对节省存储空间。

因为我们的MHA集群为一个整体，因此要让它们更好的协同工作，我们需要配置ssh密钥对认证登录，所有节点之间互相以root密钥配对认证登录

最后，我们需要安装MHA相关依赖，因为MHA由Perl语言编写，因此所有主机上需要安装Perl依赖包，而默认情况下mysql不允许网络访问，因此需要在所有数据节点上授权监控用户，这之后，我们需要在所有数据库服务器上安装MHA-node包，在管理主机上安装MHA\_node和MHA-manager包。

* + 1. 部署容器环境

首先，我们需要确认在每台机器上都安装了docker服务，这一点我们可以通过上面配置的模板机中得到保证，在这以后，我们需要从docker hub下拉centos系统的镜像，然后在此镜像的基础上，将不同机器的MHA配置封装为镜像。最后，运行容器实例，将宿主机作为客户端进行测试。

* + 1. 对比实验和分析

在环境都布置完毕后，我们为了实验方便，需要部署一些脚本，在这之后，先通过docker启动并部署我们的集群，看看是否成功，再分析docker的资源消耗。在这以后，停掉docker容器，使用原本的机器构建MHA。最后分析虚拟化和docker对资源的损耗。

* 1. 本章小结

本章主要介绍了整个环境的搭建思路和步骤。

1. 基础环境部署
   1. 创建虚拟机
      1. 总述

因为个人使用的笔记本为Windows系统，且本人对Windows原生的Hyper-V没有进行深入了解，因此选择老朋友VMWare Workstation 14。系统选择CentOS 7.4，包为完整镜像包(Everything)，版本为CentOS-7-x86\_64-Everything-1708.iso。语言选择美国英语，系统选择最小化安装，root密码设定为123456。

* + 1. 下载镜像

镜像地址：

<http://archive.kernel.org/centos-vault/7.4.1708/isos/x86_64/CentOS-7-x86_64-Everything-1708.iso>

一般情况下，我们不信任第三方软件和破解软件，在官方镜像开源的情况下，优先选择官方镜像进行下载和部署。

* + 1. 新建虚拟机

1. 创建一个基本的虚拟机，选择典型安装即可。

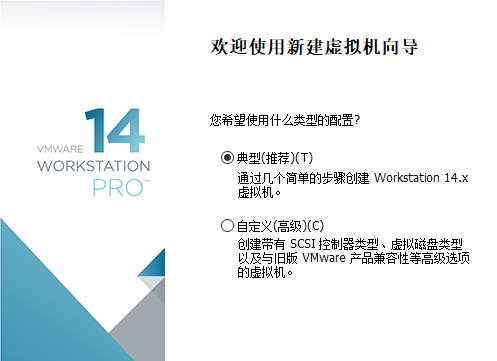


图3-1 安装类型选择

1. VMWare WorkStation会默认执行一系列的初始化操作，为了避免这一点我们选择稍后安装操作系统。



图3-2 安装类型选择

1. 选择Linux -> CentOS 7 64位

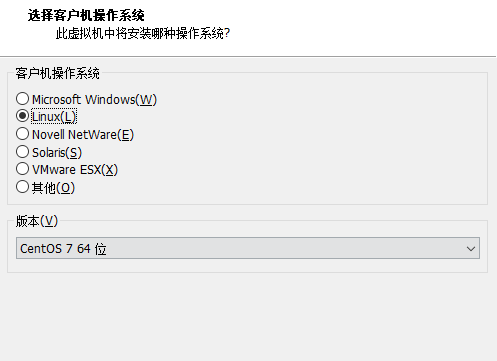


图3-3 系统类型选择

1. 这里设置的是最大值而非占用值，后端盘设置为50G，最后我们的mysql51大小达到了21G。

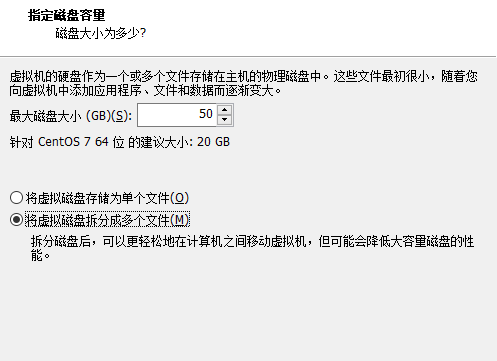


图3-4 指定后端盘文件最大容量

1. 内存设置为1G。

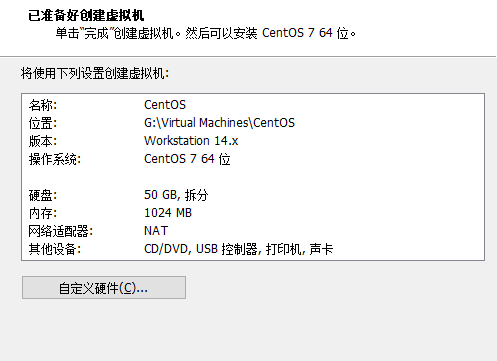


图3-5 设定内存大小

1. 在虚拟机CentOS光驱中插入镜像。

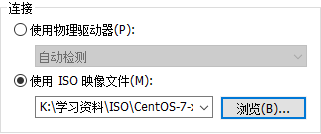


图3-6 插入光驱

* + 1. 给虚拟机安装系统

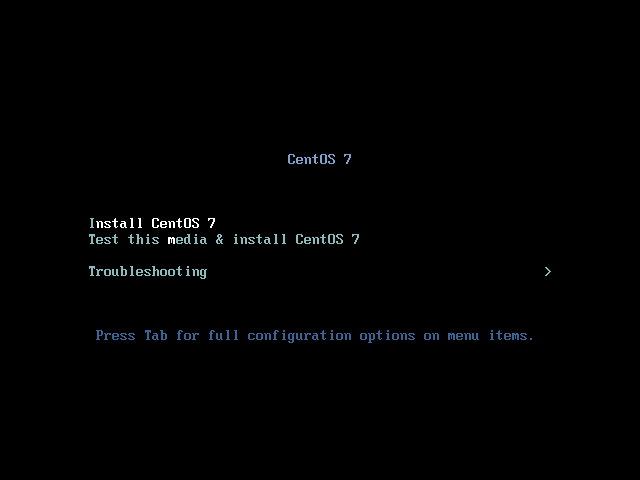


图3-7 CentOS 7.4 安装界面

1. 语言选择US
2. 选择最小化安装

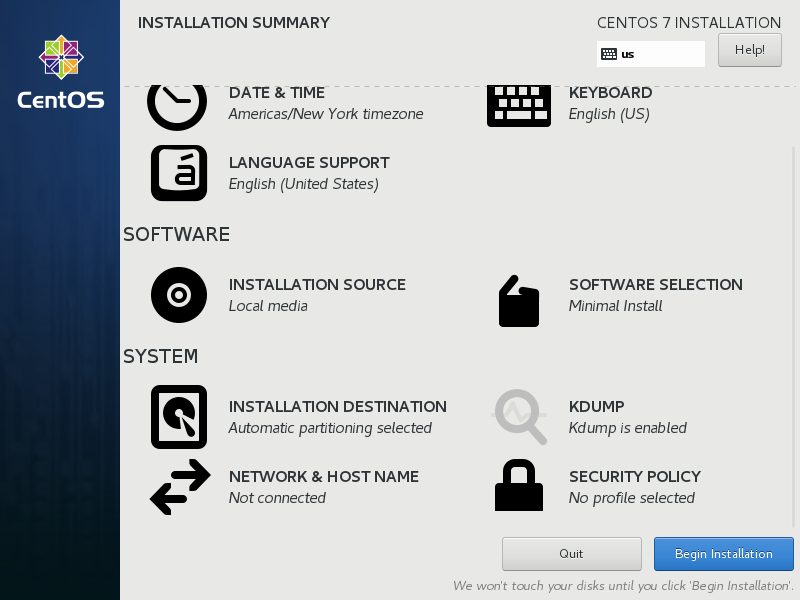


图3-8 详细安装配置

1. 密码设置为123456
   * 1. 模板化虚拟机
2. 网络配置

因为该虚拟机为最小化安装，故没有图形界面。第一步需要进行网络的配置和yum软件仓库的搭建。因为是在虚拟机上进行环境的模拟和部署，因此网络的搭建分为外部虚拟网络的搭建和内部网络配置的部署。

我们将虚拟机的虚拟网卡选用NAT转发模式，虚拟网段设置为192.168.1.0/24，这样虚拟机可以将我们的物理机网卡作为路由器，通过NAT转发的形式和外部网络接轨，这一步是为了我们的虚拟机能够从Docker hub上拉取我们所需要的镜像。而建立虚拟局域网是为了让我们的集群工作在同一个网络中，能够有效的进行通信以及数据交换。

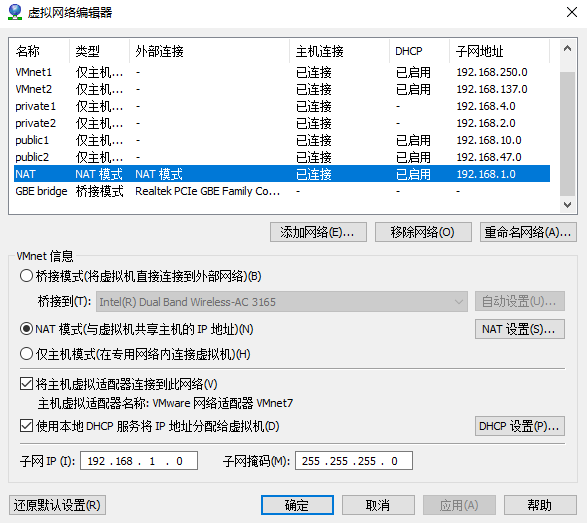


图3-9 虚拟网络配置

因为这一台虚拟机为模板机，所有配置需要去个性化保留共性，因此不能配置固定IP，而是在配置过程中使用虚拟IP的形式，确保重启后配置会初始化。因此我们设置网卡ens33开机启动，模式为static，不设IP，此时系统中未安装vim编辑器，且在网络没有配置完备的情况下配置yum源并非我们期待的去个性化设置，于是使用vi进行编辑，对网络设备ens33进行配置。

因Linux系统中一切皆文件的特性，故网络设备ens33在CentOS中的文件路径为/etc/sysconfig/network-scripts/ifcfg-ens33

修改网络设备配置文件并启动：

|  |
| --- |
| [root@localhost ~]# vi /etc/sysconfig/network-scripts/ifcfg-ens33  TYPE=Ethernet  PROXY\_METHOD=none  BROWSER\_ONLY=no  BOOTPROTO=none  DEFROUTE=yes  IPV4\_FAILURE\_FATAL=no  NAME=ens33  DEVICE=ens33  ONBOOT=yes  [root@localhost ~]# ifup ens33 |

此时，模板机网卡虽然启动，但是没有配置IP地址，因此，按照分析，我们需要给设备ens33配置一个虚拟网络地址(VIP)：192.168.1.10

|  |
| --- |
| [root@localhost ~]# ip addr add 192.168.1.10/24 dev ens33 |

1. 搭建本地Yum源

Yum(Yellowdog Update Manager)仓库是一个类似于手机应用商店一样的存在，RedHat系列的Linux系统通过Yum软件仓库获取需要的应用软件。Yum仓库可以是本地Yum源也可以是网络Yum源，在这里，我们要搭建的是一个本地+Ftp网络Yum源。

系统镜像中提供了相当多的匹配系统版本的软件，这部分软件被称为基础应用，在系统中表示为Base包，因此，我们只需要将系统镜像加载到光驱上，再将光驱上的镜像挂载到本地即可使用，这样可以避免浪费无用的磁盘空间，并且因为本地加载的缘故，读取速度和效率也会大大提升，且不占用网络带宽。

我们在虚拟模板机光驱中插入iso镜像。

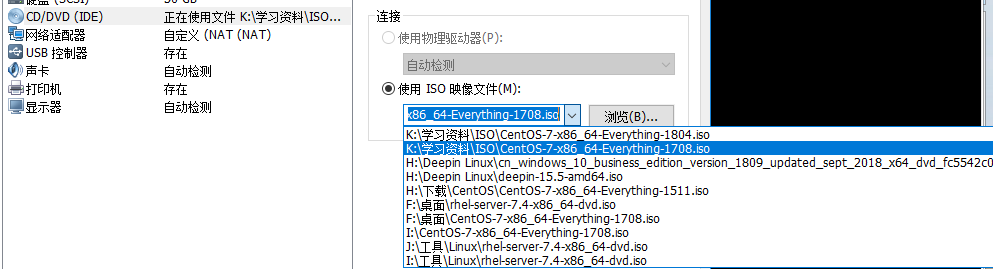


图3-10 镜像插入

然后使用mkdir命令创建对应目录，通过mount命令将镜像挂载到创建的目录，光驱设备的路径为"/dev/cdrom"(同"/dev/sr0")，此时我们可以在目录下看到镜像的内容，并且镜像内容对系统而言是只读的。

之后，在"/etc/yum.repos.d/"目录下创建我们的yum仓库配置文件，并修改创建的CentOS\_Base.repo，最后使用"yum repolist"命令刷新仓库列表，验证yum是否搭建成功，出现图中输出，说明本地Yum搭建成功。

|  |
| --- |
| [root@localhost ~]# mkdir -p /yum/CentOS7  [root@localhost ~]# mount /dev/cdrom /yum/CentOS7  [root@localhost ~]# rm -f /etc/yum.repos.d/\*  [root@localhost ~]# vi /etc/yum.repos.d/CentOS\_Base.repo  [CentOS]  name=CentOS7\_Base  baseurl=file:///yum/CentOS7  enabled=1  gpgcheck=0  [root@localhost ~]# yum repolist |

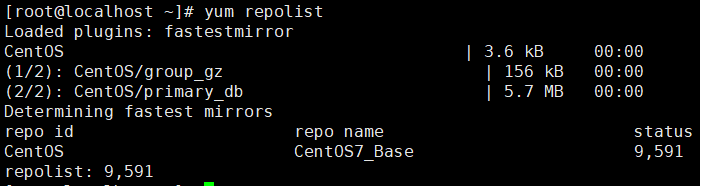


图3-11 搭建好的yum仓库

1. 常用软件安装

本虚拟机为模板虚拟机，为了让后面的机子使用起来更加方便，我们将安装一些常用的工具软件。

表3-1 常用软件清单表

|  |  |  |
| --- | --- | --- |
| 软件名 | 作用 | 包名和版本 |
| vim编辑器 | vi编辑器增强版，功能强大，用于Linux内的文本编辑 | vim-filesystem-7.4.160-2.el7  vim-enhanced-7.4.160-2.el7  vim-minimal-7.4.160-2.el7  vim-common-7.4.160-2.el7 |
| bash补全工具 | 用于Linux界面下Tab键自动补全功能 | bash-completion-2.1-6.el7.noarch |
| 网络工具 | 更快捷方便的网络管理 | net-tools-2.0-0.22.20131004git.el7 |
| 自定义yum仓库工具 | 能够支持将目录变为自定义yum仓库 | createrepo-0.9.9-28.el7.noarch |
| FTP服务 | 能够搭建本地FTP服务，方便局域网内的文件传输 | vsftpd-3.0.2-22.el7 |

|  |
| --- |
| [root@localhost ~]# yum install -y \  > {createrepo,bash-completion,vim,net-tools,vsftpd}  [root@localhost ~]# for i in \  > {createrepo,bash-completion,vim,net-tools,vsftpd}  > do rpm -qa | grep $i  > done  [root@localhost ~]# rpm -qa  ...  createrepo-0.9.9-28.el7.noarch  bash-completion-2.1-6.el7.noarch  vim-filesystem-7.4.160-2.el7.x86\_64  vim-enhanced-7.4.160-2.el7.x86\_64  vim-minimal-7.4.160-2.el7.x86\_64  vim-common-7.4.160-2.el7.x86\_64  net-tools-2.0-0.22.20131004git.el7.x86\_64  vsftpd-3.0.2-22.el7.x86\_64  ... |

1. 持久化模板配置

首先我们要将搭建好的yum源持久化，为此，我们需要启动ftp服务且设置开机自启。并将yum源移动位置。

|  |
| --- |
| [root@localhost ~]# systemctl start vsftpd  [root@localhost ~]# systemctl enable vsftpd  [root@localhost ~]# mkdir /var/ftp/CentOS7  [root@localhost ~]# umount /yum/CentOS7  [root@localhost ~]# mkdir /var/ftp/CentOS7  [root@localhost ~]# vim /etc/fstab  /dev/cdrom /var/ftp/CentOS7 iso9660 defaults 0 0  /var/ftp/CentOS7 /yum/CentOS7 xfs bind 0 0  [root@localhost ~]# mount -a |

然后为了避免每一次修改网络配置都需要修改配置文件，我们写一个快速配置ip的脚本eip.sh，方便后续操作

|  |
| --- |
| #!/bin/bash  ######################################  ############获取设备验证##############  getDev(){  ifconfig $1 &> /dev/null  if [ $? -ne 0 ];then  echo "error device" >&2  exit 2  else  echo "change $1"  fi  }  ############改变配置####################  changeIP(){  path='/etc/sysconfig/network-scripts/ifcfg-'  for i in {BOOTPROTO,IPADDR,PREFIX,GATEWAY,ONBOOT}  do  read -p "$i: " $i  done  cat > $path$1 << EOF  TYPE=Ethernet  PROXY\_METHOD=none  BROWSER\_ONLY=no  BOOTPROTO=$BOOTPROTO  DEFROUTE=yes  IPV4\_FAILURE\_FATAL=no  NAME=$1  DEVICE=$1  ONBOOT=$ONBOOT  IPADDR=$IPADDR  PREFIX=$PREFIX  GATEWAY=$GATEWAY  EOF  echo  for i in {BOOTPROTO,IPADDR,PREFIX,GATEWAY,ONBOOT}  do  temp=  eval temp=$(echo \$$i)  if [ ! $temp ];then  echo $i remove  sed -i "/$i/d" $path$1  else  echo $i=$temp  fi  done  }  ##################执行##################  if [ "$1" != "" ];then  getDev $1  changeIP $1  else  echo "device please" >&2  fi |

最后我们需要关闭selinux、防火墙以及设置快速关机，因为SeLinux的安全系数非常高，很多程序和功能会被它所阻止，因此我们一般会将它关闭;而内网中不需要防火墙，我们将防火墙状态调整为mask。

|  |
| --- |
| [root@mysql51 ~]# sed -i 's/^SELINUX=.\*/SELINUX=disabled/' /etc/selinux/config  [root@mysql51 ~]# systemctl stop firewalld  [root@mysql51 ~]# systemctl disable firewalld  [root@mysql51 ~]# systemctl mask firewalld  [root@localhost ~]# vim /root/.bashrc  alias gg='poweroff'  [root@localhost ~]# source /root/.bashrc  [root@localhost ~]# gg |

* 1. 安装mysql
     1. 创建快照"模板v1.1"并创建连接克隆mysql51

此时，我们的模板机已经配置完毕，根据此模板进行集群机器的创建。



图3-12 创建快照



图3-13 创建mysql51

* + 1. 以mysql51为例，建立mysql软件仓库

在mysql51创建好后，我们要对mysql51进行自定义修改，方便搭建集群环境，主要步骤有：首先将主机名修改为对应的标识，再将集群的地址和主机名对应关系按照拓扑设计中写入hosts本地解析文件，最后修改每台主机的IP地址和DNS。

* + 1. 修改hostname

|  |
| --- |
| [root@localhost ~]# hostnamectl set-hostname mysql51 |

* + 1. 修改hosts文件

本地解析文件路径：/etc/hosts，在hosts文件中添加表3-2中配置。

表3-2 本机解析配置

|  |  |
| --- | --- |
| 主机解析名 | 对应IP |
| mysql51 | 192.168.1.51 |
| mysql52 | 192.168.1.52 |
| mysql53 | 192.168.1.53 |
| mysql54 | 192.168.1.54 |
| mysql55 | 192.168.1.55 |
| mysql56 | 192.168.1.56 |

* + 1. 修改IP和DNS

之前，我们写了一个交互式配置IP的脚本来配置IP，此时我们调用脚本，输入要修改的网络设备，按照提示进行修改即可。

重启网络服务后，此时配置已经生效，但想要正常地访问外网(dockerhub)，我们还需要配置DNS解析，DNS服务器地址为虚拟网络中的物理机地址192.168.1.254，DNS解析配置文件路径为"/etc/resolv.conf"，字段为"nameserver"。

|  |
| --- |
| [root@mysql51 ~]# eip.sh ens33  change ens33  BOOTPROTO: none  IPADDR: 192.168.1.51  PREFIX: 24  GATEWAY: 192.168.1.254  ONBOOT: yes  BOOTPROTO=none  IPADDR=192.168.1.51  PREFIX=24  GATEWAY=192.168.1.254  ONBOOT=yes  [root@mysql51 ~]# systemctl restart network  [root@mysql51 ~]# echo 'nameserver 192.168.1.254' > /etc/resolv.conf |

这时候已经可以访问外网了，于是下载并安装mysql相关软件包（软件包清单如表3-3所示），将它们放到ftp下的software目录中。然后创建自定义软件仓库mysql,在"/var/ftp/software/mysql"目录下使用createrepo命令创建自定义软件仓库。

在这以后，配置额外yum源，先使用mount命令将yum配置目录挂载到/root/repos下，方便修改和传输，再写第二个配置文件else.repo。

写完配置，就可以通过yum调用之前配置好的mysql仓库了，安装并启动mysql。

表3-3 mysql软件包

|  |
| --- |
| mysql-community-client-5.7.17-1.el7.x86\_64.rpm |
| mysql-community-common-5.7.17-1.el7.x86\_64.rpm |
| mysql-community-devel-5.7.17-1.el7.x86\_64.rpm |
| mysql-community-embedded-5.7.17-1.el7.x86\_64.rpm |
| mysql-community-embedded-compat-5.7.17-1.el7.x86\_64.rpm |
| mysql-community-embedded-devel-5.7.17-1.el7.x86\_64.rpm |
| mysql-community-libs-5.7.17-1.el7.x86\_64.rpm |
| mysql-community-libs-compat-5.7.17-1.el7.x86\_64.rpm |
| mysql-community-minimal-debuginfo-5.7.17-1.el7.x86\_64.rpm |
| mysql-community-server-5.7.17-1.el7.x86\_64.rpm |
| mysql-community-test-5.7.17-1.el7.x86\_64.rpm |

|  |
| --- |
| [root@mysql51 ~]# cd /var/ftp/  [root@mysql51 ftp]# mkdir -p software/mysql  [root@mysql51 ftp]# cd software/mysql  [root@mysql51 mysql]# createrepo .  Spawning worker 0 with 11 pkgs  Workers Finished  Saving Primary metadata  Saving file lists metadata  Saving other metadata  Generating sqlite DBs  Sqlite DBs complete  [root@mysql51 ~]# vim /etc/profile.d/startMount.sh  mount -o bind /etc/yum.repos.d /root/repos &> /dev/null  [root@mysql51 ~]# source /etc/profile.d/startMount.sh  [root@mysql51 ~]# vim repos/else.repo  [mysql]  name=mysql  baseurl=ftp://192.168.1.51/software/mysql  enabled=1  gpgcheck=0  [root@mysql51 ~]# yum install mysql-community-\* -y  [root@mysql51 ~]# systemctl start mysqld |

* 1. 安装docker
     1. 获取软件包

软件包清单如3-4所示，软件包的获取可以去官方网站与合作网站，此处不做赘述。

表3-4 docker软件包

|  |
| --- |
| container-selinux-2.21-2.gitba103ac.el7.noarch.rpm |
| container-storage-setup-0.6.0-1.gite67c964.el7.noarch.rpm |
| docker-1.12.6-55.gitc4618fb.el7.centos.x86\_64.rpm |
| docker-client-1.12.6-55.gitc4618fb.el7.centos.x86\_64.rpm |
| docker-common-1.12.6-55.gitc4618fb.el7.centos.x86\_64.rpm |
| oci-register-machine-0-3.11.1.gitdd0daef.el7.x86\_64.rpm |
| oci-systemd-hook-0.1.12-1.git1e84754.el7.x86\_64.rpm |
| oci-umount-1.12.6-55.gitc4618fb.el7.centos.x86\_64.rpm |
| skopeo-containers-0.1.23-1.git1bbd87f.el7.x86\_64.rpm |

* + 1. 配置yum,安装docker

如果模板机只用来做实验，这一步其可以一并做进模板机中。但问题不大，我们后面的虚拟机使用mysql51的连接克隆即可。

我们首先和mysql一样，创建自定义yum仓库。然后将docker模块写入else.repo配置文件中去。配置完后利用yum工具安装docker后启动服务即可。

启动docker服务后，我们可以连接到外网的docker hub，使用docker pull命令下拉centos镜像。

|  |
| --- |
| [root@mysql51 docker]# mkdir -p /var/ftp/software/docker/  [root@mysql51 ~]# vim repos/else.repo  [docker]  name=docker  baseurl=ftp://192.168.1.51/software/docker  enabled=1  gpgcheck=0  [root@mysql51 docker]# yum install docker\* -y  [root@mysql51 ~]# systemctl start docker  [root@mysql51 ~]# systemctl enable docker  [root@mysql51 ~]# docker search centos  [root@mysql51 ~]# docker pull docker.io/centos |

* 1. 初始化mysql配置
     1. 查看mysql初始密码

mysql安装后默认会有一个随机密码，需要进入mysql日志查看初始密码才能登陆，日志文件路径为"/var/log/mysqld.log"。

|  |
| --- |
| [root@mysql51 ~]# less /var/log/mysqld.log  2019-04-23T15:04:03.349694Z 1 [Note] A temporary password is generated for root@localhost: C.3w;T)<vx1K |

* + 1. 修改mysql密码策略与登录密码

默认情况下，mysql的密码安全策略级别比较高，需要满足各种限制条件才能够允许修改密码，因此需要降低密码限制策略和降低密码限制长度。

mysql的配置文件为"/etc/my.cnf"，为了方便对docker容器的映射，将日志文件从/var/log放到mysql数据目录下，路径为"/var/lib/mysql"。涉及到的配置字段如表3-5所示。

将root用户的密码修改为123456(生产环境中可换其他密码)。

表3-5 涉及的配置字段表

|  |  |
| --- | --- |
| 配置字段 | 说明 |
| validate\_password\_policy | 密码限制策略 |
| validate\_password\_length | 密码长度限制 |
| log-error | 日志存放路径 |

|  |
| --- |
| [root@mysql51 ~]# vim /etc/my.cnf  [mysqld]  validate\_password\_policy=0  validate\_password\_length=6  log-error=/var/lib/mysql/mysqld.log  [root@mysql51 ~]# systemctl restart mysqld  [root@mysql51 ~]# mysql -uroot -p'C.3w;T)<vx1K'  mysql> alter user user() identified by "123456";  Query OK, 0 rows affected (0.00 sec) |

* + 1. 添加远程登录权限

默认状态下，mysql只允许本机登录，想要通过网络登录，需要提前进行授权，否则管理节点无法连接到数据库服务。授权通过"grant"命令字段实现。而实验目的是搭建MHA集群，所以还需要实现主从同步，因此，还需要给库添加从库的repluser同步用户。授权完后，我们可以通过查询mysql库的user表来验证。

|  |
| --- |
| mysql> grant all on \*.\* to root@'%' identified by '123456';  Query OK, 0 rows affected, 1 warning (0.00 sec)  mysql> grant replication slave on \*.\* to repluser@"%" identified by '123456';  Query OK, 0 rows affected, 1 warning (0.00 sec)  mysql> select \* from mysql.user where user="root"\G  mysql> select \* from mysql.user where user="repluser"\G |

至此，我们的mysql配置完成。

* + 1. 生成mysql51快照，并以做链接克隆52-56

与生成模板机快照的方式相同，生成mysql51的快照，并且根据快照生成链接克隆52-56。为之后的实验做准备。

* 1. 本章小结

本章主要进行了虚拟环境的搭建和软件的部署配置。

1. 镜像的封装使用和MHA部署
   1. mysql镜像
      1. 封装mysql镜像

mysql镜像我们在centos基础上进行构建，新创建一个目录，在下面写一个mysql的启动脚本start.sh，然后将repos目录下的yum配置文件复制到当前目录下，写Dockerfile构建镜像文件，最后使用"docker build"命令生成镜像"mysql"。

|  |
| --- |
| [root@mysql51 ~]# mkdir -p dockerFiles/mysql  [root@mysql51 ~]# cp repos/\* dockerFiles/mysql/  [root@mysql51 ~]# cd dockerFiles/mysql  [root@mysql51 mysql]# vim start.sh  #!/bin/bash  /usr/sbin/mysqld --user mysql  [root@mysql51 mysql]# vim Dockerfile  FROM centos:latest  MAINTAINER XiDong Xu  RUN rm -f /etc/yum.repos.d/\*  ADD CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo  ADD else.repo /etc/yum.repos.d/else.repo  ADD start.sh /usr/local/bin/start.sh  RUN yum install -y net-tools psmisc vim bash-completion mysql-community-\*  EXPOSE 3306  CMD ["/usr/local/bin/start.sh"]  [root@mysql51 mysql]# docker build -t mysql:01 . |

* 1. 搭建 MHA主库
     1. 软件包安装

MHA环境需要安装软件包，需要注意的是，数据节点和管理节点用到的软件包是不同的。但相同的是，它们都需要安装MHA\_nodes，而MHA整个软件系统是由Perl语言书写的，因此需要perl相关的软件包依赖，下面是所用到的依赖包，利用之前搭建好的ftp服务安装软件包。

表4-1 perl软件包列表

|  |  |
| --- | --- |
| 需要安装的主机 | 安装的软件包 |
| mysql51 | perl-Config-Tiny-2.14-7.el7.noarch |
| mysql52 | perl-Email-Date-Format-1.002-15.el7.noarch |
| mysql53 | perl-Log-Dispatch-2.41-1.el7.1.noarch |
| mysql54 | perl-Mail-Sender-0.8.23-1.el7.noarch |
| mysql55 | perl-Mail-Sendmail-0.79-21.el7.art.noarch |
| mysql56 | perl-MIME-Lite-3.030-1.el7.noarch |
| —— | perl-MIME-Types-1.38-2.el7.noarch |
| —— | perl-Parallel-ForkManager-1.18-2.el7.noarch |

* + 1. 制作MHA镜像

修改mysql配置文件"/etc/my.cnf"，启用binlog日志，加载半同步模块，启用半同步模块。

先在yum源中添加MHA模块，然后利用"createrepo"制作yum源，再创建mysql51镜像文件夹，复制my.cnf和/var/lib/mysql配置文件到目录下，然后制作perl依赖包清单，完成后继续制作运行脚本。在运行的过程中，会出现版本冲突问题，因为容器的centos版本为7.6，会有rpm包冲突，在这里需要使用"yum downgrade"命令给"openssl-lib"降版本。至此，准备工作完成，开始制作镜像。

制作Dockerfile文件，将上面的相关组件放置到对应位置，然后通过docker build创建镜像mysql:MHA。

|  |
| --- |
| [root@mysql51 docker]# vim ~/repos/else.repo  [MHA]  name=MHA  baseurl=ftp://192.168.1.51/software/MHA  enabled=1  gpgcheck=0  [root@mysql51 ~]# vim dockerFiles/mysql51/my.cnf  log\_bin=master51  server\_id=51  relay\_log\_purge=off  plugin-load="rpl\_semi\_sync\_master=semisync\_master.so;rpl\_semi\_sync\_slave=semisync\_slave.so"  rpl\_semi\_sync\_master\_enabled=1;  rpl\_semi\_sync\_slave\_enabled=1;  [root@mysql51 docker]# createrepo /var/ftp/software/MHA/  [root@mysql51 ~]# mkdir -p /root/dockerFiles/mysql51/  [root@mysql51 ~]# cd /root/dockerFiles/mysql51  [root@mysql51 mysql51]# cp -rp /etc/my.cnf .  [root@mysql51 mysql51]# cp -rp /var/lib/mysql .  [root@mysql51 docker]# ls /var/ftp/software/MHA/ > MHARPMS  [root@mysql51 docker]# sed -i 's/.rpm$//g' MHARPMS  [root@mysql51 docker]# vim yuminstall.sh  #!/bin/bash  yum downgrade openssl-libs -y  for i in `cat /root/MHARPMS`  do  yum install -y $i  done  [root@mysql51 docker]# vim Dockerfile  FROM mysql:01  ADD MHARPMS /root/MHARPMS  ADD else.repo /etc/yum.repos.d/else.repo  ADD yuminstall.sh /root/yuminstall.sh  RUN /root/yuminstall.sh  [root@mysql51 docker]# docker build -t mysql:MHA . |

* + 1. 用主库文件生成mysql52、mysql53目录

创建并复制对应文件到mysql52和mysql53，修改对应配置文件，类似于mysql51。

|  |
| --- |
| [root@mysql51 dockerFiles]# cp -rp mysql51 mysql52  [root@mysql51 dockerFiles]# cp -rp mysql51 mysql53  [root@mysql51 dockerFiles]# vim mysql52/my.cnf  [mysqld]  log\_bin=master52  server\_id=52  relay\_log\_purge=off  plugin-load="rpl\_semi\_sync\_master=semisync\_master.so;rpl\_semi\_sync\_slave=semisync\_slave.so"  rpl\_semi\_sync\_master\_enabled=1;  rpl\_semi\_sync\_slave\_enabled=1;  [root@mysql51 dockerFiles]# vim mysql53/my.cnf  [mysqld]  log\_bin=master53  server\_id=53  relay\_log\_purge=off  plugin-load="rpl\_semi\_sync\_master=semisync\_master.so;rpl\_semi\_sync\_slave=semisync\_slave.so"  rpl\_semi\_sync\_master\_enabled=1;  rpl\_semi\_sync\_slave\_enabled=1; |

* 1. 搭建MHA集群从库
     1. 创建mysql54从库目录

创建mysql54目录，将配置文件和数据库内容复制到当前目录，创建镜像目录，然后根据mysql54生成mysql55和mysql56。这样数据节点就准备好了。接下来开始搭建MHA环境，准备运行。

|  |
| --- |
| [root@mysql51 ~]# mkdir dockerFiles/mysql54  [root@mysql51 ~]# cd dockerFiles/mysql54  [root@mysql51 mysql54]# cp -rp ../mysql51/my.cnf ../mysql51/mysql .  [root@mysql51 mysql54]# mkdir docker/  [root@mysql51 ~]# cp -rp dockerFiles/mysql54 dockerFiles/mysql55 |

* 1. 搭建MHA基础环境
     1. 免密配置

修改ssh配置免验证，第一步利用ssh-key-gen生成密钥，然后将传输密钥到其他服务器上。

|  |
| --- |
| [root@mysql51 mysql51]# vim /etc/ssh/ssh\_config  StrictHostKeyChecking no  [root@mysql51 mysql51]# ssh-keygen -t rsa -b 2048 -N '' -f /root/.ssh/id\_rsa  [root@mysql51 script]# vim initENV.sh  #!/bin/bash  for i in mysql5{1..6}  do  ssh-copy-id -f $i  scp /etc/hosts $i:/etc/hosts &> /dev/null  scp /root/.ssh/id\_rsa $i:/root/.ssh/id\_rsa &> /dev/null  scp /root/repos/\* $i:/etc/yum.repos.d/ &> /dev/null  done |

* 1. 配置MHA-Manager管理主机
     1. 创建一个新的MHA容器，制作MHA-manager镜像

创建一个新的MHA容器MHAManager，在容器中配置MHA-manager。从FTP服务器上下载MHA-manager包。需要注意的是MHA-manager的安装需要使用perl-ExtUtils-\*和perl-CPAN-\*，此处解决他们的依赖关系，需要给libdb包降低版本，注意此处不能只降单个包的版本，要连依赖一起，否则会显示降级失败，没有这两个包会出现编译错误。

安装完依赖后perl编译并安装MHA-manager。待安装完成后开始配置MHA-manager。

创建配置目录"/etc/MHA\_manager/"，将配置文件模板复制到对应目录下，编写配置文件。将高可用脚本复制到MHA-manager配置目录下并进行修改，当主库出现故障的时候，MHA会根据ip\_failover中的配置将主库更换为备用主库，防止单点故障，实现高可用。

因MHA启动较为麻烦，写启动关闭脚本,通过start、stop、check、ssh、repl字段来控制MHA管理节点。最后提交容器为新的mysql:manager镜像，完成制作。

|  |
| --- |
| [root@mysql51 docker]# docker run -it --name MHAManager -v /root/dockerFiles/mysql56/docker/my.cnf:/etc/my.cnf -v /root/dockerFiles/mysql56/docker/mysql/:/var/lib/mysql mysql:MHA bash  [root@03fae928ea04 /]# yum -y install wget  [root@03fae928ea04 /]# wget ftp://192.168.1.51/software/MHA/MHA4mysql-manager-0.56.tar.gz  [root@03fae928ea04 /]# tar -xf MHA4mysql-manager-0.56.tar.gz  [root@03fae928ea04 /]# yum downgrade -y libdb\*  [root@03fae928ea04 /]# vim /etc/yum.repos.d/CentOS-Base.repo  [glibc\_common]  name=glibc-common  baseurl=ftp://192.168.1.51/software/CentOS7  enabled=0  gpgcheck=0  [root@03fae928ea04 MHA4mysql-manager-0.56]# yum -y install perl-ExtUtils-\* perl-CPAN-\*  [root@03fae928ea04 MHA4mysql-manager-0.56]# perl Makefile.PL  [root@03fae928ea04 MHA4mysql-manager-0.56]# make && make install  [root@03fae928ea04 MHA4mysql-manager-0.56]# mkdir /etc/MHA\_manager  [root@03fae928ea04 MHA4mysql-manager-0.56]# cp samples/conf/app1.cnf /etc/MHA\_manager/  [root@03fae928ea04 MHA4mysql-manager-0.56]# vim /etc/MHA\_manager/app1.cnf  [server default]  manager\_workdir=/etc/MHA\_manager  manager\_log=/etc/MHA\_manager/manager.log  master\_ip\_failover\_script=/etc/MHA\_manager/master\_ip\_failover  ssh\_user=root  ssh\_port=22  repl\_user=repluser  repl\_password=123456  user=root  password=123456  [server1]  hostname=192.168.1.51  candidate\_master=1  [server2]  hostname=192.168.1.52  candidate\_master=1  [server3]  hostname=192.168.1.53  candidate\_master=1  [server4]  hostname=192.168.1.54  no\_master=1  [server5]  hostname=192.168.1.55  no\_master=1  [root@03fae928ea04 MHA4mysql-manager-0.56]# cp samples/scripts/master\_ip\_failover /etc/MHA\_manager/  [root@03fae928ea04 MHA\_manager]# vim master\_ip\_failover  ...  my $vip = '192.168.1.100/24'; # Virtual IP  my $key = "1";  my $ssh\_start\_vip = "/sbin/ifconfig eth0:$key $vip";  my $ssh\_stop\_vip = "/sbin/ifconfig eth0:$key down";  ...  [root@mysql51 ~]# docker stop MHAManager  [root@mysql51 ~]# docker commit MHAManager test:manager |

* 1. 部署MHA环境(其二)
     1. 给所有主机安装nfs-utils工具

实验中模拟生产环境中存储集群提供存储服务，所有的数据库数据全部在mysql51，由mysql51提供共享给其他服务器使用。nfs工具的使用需要nfs-utils工具。

|  |
| --- |
| [root@mysql51 ~]# for i in mysql{51..56} ; do ssh $i "yum install nfs-utils.x86\_64 -y" ; ssh $i "systemctl start rpcbind ; systemctl enable rpcbind ; systemctl restart nfs ; systemctl enable nfs " ; done |

* + 1. 配置mysql51上的NFS

修改mysql51的NFS服务器配置，配置文件路径为" /etc/exports"，配置字段值为"/root/myDB 192.168.1.0/24(rw,no\_root\_squash)"，将myDB目录共享给192.168.1.0网段，使它们都具有读写权限，且不降权。

配置完毕后，在mysql52上进行校验，通过"showmount -e 192.168.1.51"命令查看是否检测到网络中的nfs挂载服务。成功后，将dockerfile里的相关数据文件移动到myDB中。

检查无误后，在所有主机上挂载mysql51上的myDB。

|  |
| --- |
| [root@mysql51 ~]# vim /etc/exports  /root/myDB 192.168.1.0/24(rw,no\_root\_squash)  [root@mysql52 ~]# showmount -e 192.168.1.51  Export list for 192.168.1.51:  /root/myDB 192.168.1.0/24  [root@mysql51 ~]# for i in mysql{52..56} ; do ssh root@$i "mkdir -p /root/myDB" ; done  [root@mysql51 ~]# vim upmount.sh  mount 192.168.1.51:/root/myDB /root/myDB &> /dev/null  [root@mysql51 ~]# for i in mysql{52..56} ; do scp upmount.sh root@$i:/etc/profile.d/ ; ssh root@$i "bash /etc/profile.d/upmount.sh" ; done  [root@mysql52 ~]# ls myDB/  mysql51 mysql52 mysql53 mysql54 mysql55  [root@mysql51 ~]# for i in mysql{51..56} ; do mv dockerFiles/$i/my\* myDB/$i/ ; done |

* 1. 搭建并部署自定义镜像仓库
     1. 在mysql51上搭建自定义镜像仓库

修改docker配置文件/etc/daemon.json，配置私有镜像，字段值为 "insecure-registries" : ["192.168.1.51:5000"]，修改后重启docker服务使配置生效。此时可以开始制作自定义镜像仓库。

因为私有镜像仓库需要registry镜像来实现，所以我们从docker hub上下拉并启动registry镜像仓库，将其映射到5000端口。此时自定义镜像仓库已经启用，我们将镜像重新打标签并且上传到私有镜像仓库中。

|  |
| --- |
| [root@mysql51 ~]# vim /etc/docker/daemon.json  {  "insecure-registries" : ["192.168.1.51:5000"]  }  [root@mysql51 ~]# systemctl restart docker  [root@mysql51 ~]# docker search registry  [root@mysql51 ~]# docker pull docker.io/registry  [root@mysql51 ~]# docker run -d -p 5000:5000 docker.io/registry:latest  [root@mysql51 ~]# docker tag mysql:MHA 192.168.1.51:5000/mysql:MHA  [root@mysql51 ~]# docker tag test:manager 192.168.1.51:5000/mysql:manager  [root@mysql51 ~]# docker push 192.168.1.51:5000/mysql:MHA  [root@mysql51 ~]# docker push 192.168.1.51:5000/mysql:manager |

* + 1. 在mysql52-mysql56上使用私有镜像

和mysql51中相同，想要使用自定义镜像仓库，需要修改配置文件、重启docker服务，并且为了方便配置和搭建，我们设置开机自启docker。成功后在MHA服务器中下拉镜像"192.168.1.51:5000/mysql:MHA"，MHA管理机下拉镜像"192.168.1.51:5000/mysql:manager"。

|  |
| --- |
| [root@mysql51 ~]# for i in mysql{52..55} ; do ssh $i "docker pull 192.168.1.51:5000/mysql:MHA" ; done  [root@mysql51 ~]# ssh mysql56 "docker pull 192.168.1.51:5000/mysql:manager" |

* 1. 部署MHA环境(其二)
     1. 部署主库mysql51

现在万事俱备，可以测试MHA集群的使用了，确认mysql51的配置文件，无误后启动mysql51的docker容器，映射端口为3306，映射配置文件和数据文件。

首先验证是否能登录，注意本地登录因为sock文件变化需要用-S指定sock文件，用"mysql -uroot -p123456 -h192.168.1.51"验证远程登录。

|  |
| --- |
| [root@mysql51 mysql51]# vim my.cnf  [mysqld]  log\_bin=master51  server\_id=51  relay\_log\_purge=off  plugin-load="rpl\_semi\_sync\_master=semisync\_master.so;rpl\_semi\_sync\_slave=semisync\_slave.so" /\* 加载主库插件和从库插件 \*/  rpl\_semi\_sync\_master\_enabled=1; /\* 启用主库插件 \*/  rpl\_semi\_sync\_slave\_enabled=1; /\* 启用从库插件 \*/  validate\_password\_policy=0  validate\_password\_length=6  [root@mysql51 ~]# docker run --name mysql51 -d -v /root/myDB/mysql51/my.cnf:/etc/my.cnf -v /root/myDB/mysql51/mysql:/var/lib/mysql -p3306:3306 mysql:MHA  [root@mysql51 mysql]# mysql -uroot -p123456 -S /root/myDB/mysql51/mysql/mysql.sock  [root@mysql52 ~]# mysql -uroot -p123456 -h192.168.1.51 |

* + 1. 部署备份主库mysql52和mysql53

同上，确认mysql52配置文件，无误后启动mysql52容器，映射端口为3306，映射配置文件和数据文件，此时出现报错：

|  |
| --- |
| /usr/bin/docker-current: Error response from daemon: driver failed programming external connectivity on endpoint mysql52 (bf96470d010244d9acd058e0f24c50a599c9c32999fa156a0c4a7341b1f5dc5d): Error starting userland proxy: listen tcp 0.0.0.0:3306: bind: address already in use. |

查询到mysqld进程正在运行导致的端口占用，关闭mysqld服务解决问题，重新启动docker容器，确认mysql52能进行登录后，配置并启动mysql52主从。配置字段和值如下表所示。

表4-2 主从配置字段

|  |  |
| --- | --- |
| 字段 | 值 |
| master\_host | '192.168.1.51' |
| master\_user | 'repluser' |
| master\_password | '123456' |
| master\_log\_file | 'master51.000001' |
| master\_log\_pos | 154 |

|  |
| --- |
| [root@mysql51 mysql]# mysql -uroot -p123456 -hmysql51  mysql> show master status\G  Position: 154  [root@mysql52 ~]# mysql -uroot -p123456 -hmysql52  mysql> change master to  master\_host='192.168.1.51',  master\_user='repluser',  master\_password='123456',  master\_log\_file='master51.000001',  master\_log\_pos=154;  Query OK, 0 rows affected, 1 warning (0.06 sec)  mysql> start slave;  Query OK, 0 rows affected (0.01 sec)  #启动从库配置后，验证mysql51和mysql52的主从  mysql> show slave status\G  Slave\_IO\_Running: Yes  Slave\_SQL\_Running: Yes  #可以看到IO线程和SQL线程均正常运行，说明主从已经配置正确 |

mysql53的确认和mysql52步骤相同，不再赘述。

* + 1. 部署从库mysql54/mysql55

首先依然是确认mysql54和mysql55的配置正确。

|  |
| --- |
| [root@mysql54 ~]# vim myDB/mysql54/my.cnf  [mysqld]  log\_bin=slave54  server\_id=54  relay\_log\_purge=off  plugin-load=rpl\_semi\_sync\_slave=semisync\_slave.so  rpl\_semi\_sync\_slave\_enabled=1;  #因为只做从库，所以不加载主库插件  validate\_password\_policy=0  validate\_password\_length=6  [root@mysql55 ~]# vim myDB/mysql55/my.cnf  [mysqld]  log\_bin=slave55  server\_id=55  relay\_log\_purge=off  plugin-load=rpl\_semi\_sync\_slave=semisync\_slave.so  rpl\_semi\_sync\_slave\_enabled=1;  validate\_password\_policy=0  validate\_password\_length=6 |

确认配置没有问题后，部署mysql54 / 55容器。验证登录，配置主从并验证。因mysql55与mysql54步骤相同，不做赘述。

|  |
| --- |
| [root@mysql54 ~]# i=mysql54;docker run --name $i -d -v /root/myDB/$i/my.cnf:/etc/my.cnf -v /root/myDB/$i/mysql:/var/lib/mysql -p3306:3306 192.168.1.51:5000/mysql:MHA  [root@mysql54 ~]# mysql -uroot -p123456 -hmysql54  mysql> change master to  -> master\_host='192.168.1.51',  -> master\_user='repluser',  -> master\_password='123456',  -> master\_log\_file='master51.000001',  -> master\_log\_pos=154;  Query OK, 0 rows affected, 1 warning (0.10 sec)  mysql> start slave;  mysql> show slave status\G  Slave\_IO\_Running: Yes  Slave\_SQL\_Running: Yes  # IO线程和SQL线程均正常运行，说明主从已经配置正确 |

* + 1. 部署MHA管理机mysql56

启动MHA-manager的docker镜像，并运用写好的脚本进行环境检查。

|  |
| --- |
| [root@mysql56 ~]# docker run --name MHAManager -it 192.168.1.51:5000/mysql:manager bash  [root@2692ffd523e2 /]# MHAstart.sh ssh |

第一次检查失败，排查问题后，发现是ssh脚本调用和连接问题。在容器中安装ssh服务且将免密和本地解析传输到对应位置，再次检查，ssh检查通过。然后进行主从校验，此时主从校验可以成功，通过脚本启动MHA。

|  |
| --- |
| [root@2692ffd523e2 /]# yum install -y openssh-clients-7.4p1-11.el7.x86\_64  [root@2692ffd523e2 /]# MHAstart.sh ssh  [root@2692ffd523e2 /]# MHAstart.sh repl  [root@2692ffd523e2 /]# MHAstart.sh start |

* + 1. 脚本布置

在实验过程中，我们发现有一些步骤可以简化，使用脚本可以让我们更快的进行实验，于是在这里放出使用的脚本和其功能与内容。

1. 因为每次启动docker都需要输入大量选项，因此我们在各个MHA-Node节点中部署启动docker的脚本startdocker.sh。

|  |
| --- |
| [root@mysql51 ~]# vim /usr/local/bin/startdocker.sh  #!/bin/bash  #定义主机名变量  hst=mysql$1  #mysql51的运行  start51(){  docker stop $hst  docker rm $hst  docker run --name $hst -d -v /root/myDB/$hst/my.cnf:/etc/my.cnf -v /root/myDB/$hst/mysql:/var/lib/mysql -p3306:3306 192.168.1.51:5000/mysql:MHA  }  #mysql{52..55}的docker运行  startelse(){  docker stop $hst  docker rm $hst  docker run --name $hst -d -v /root/myDB/$hst/my.cnf:/etc/my.cnf -v /root/myDB/$hst/mysql:/var/lib/mysql -p3306:3306 192.168.1.51:5000/mysql:MHA  sleep 3  mysql -uroot -p123456 -h$hst -e "reset slave;start slave;"  }  case $1 in  51)  start51  ;;  52)  startelse  ;;  53)  startelse  ;;  54)  startelse  ;;  55)  startelse  ;;  \*)  echo "put the number of the host" >&2  ;;  Esac |

1. 各个MHA-Node节点中挂载mysql51存储的脚本upmount.sh

|  |
| --- |
| [root@mysql52 ~]# vim /etc/profile.d/upmount.sh  #!/bin/bash  mount 192.168.1.51:/root/myDB /root/myDB &> /dev/null  mount -o bind /root/myDB/$HOSTNAME/mysql /var/lib/mysql &> /dev/null |

1. MHA-Manager中对MHA进行快速管理的脚本MHAstart.sh，在MHA-manager部署步骤有写，不做赘述。
   1. 使用通常虚拟机进行MHA部署
      1. 停止容器并启动各个节点的mysqld服务

|  |
| --- |
| [root@mysql51 ~]# docker stop $(docker ps)  [root@mysql51 ~]# systemctl start mysqld |

* + 1. MHA管理节点进行健康检查

首先进行ssh检查，确认通过后进行repl主从检查失败，推测为从库变更，需要还原数据库初始状态，经检查，发现是因为my.cnf配置文件位置的变化导致的配置不同步问题，将文件复制并覆盖后，重启服务，再次检查成功，启动MHA。

* 1. 实验中出现意外的问题
     1. 问题描述

在实验中，我遇到了以前没有遇到过的问题。那就是在多次调用MHA脚本的时候，无论是执行健康检查也好，还是执行主从检查也好，还是启用MHA也好，都会导致数据节点的崩溃。

* + 1. 问题探究和讨论

在测试的过程中崩溃症状表现为IO直接宕死，无响应。刚开始，我认为这是虚拟机运行久了导致的偶然性卡死。但随着次数增加，我逐渐感觉不太对。而当虚拟机界面上出现"内存溢出"的时候，我才发现问题没有那么简单。刚开始我认为是内存不足，但实际发现启用镜像后内存占用仅仅达到300+MB，而MHA跑起来后也没有出问题。

当第二次调用时，内存迅速由600+变成400,200,100,65,随即在卡死的边缘游走。经过这次测试后，我认为可能是内存不足导致的，于是将虚拟机的内存调到了2G，6台一起，此时物理机的负荷已经比较高了。而结果很明显，只是再次重复了上面的现象罢了。

因此我认为MHA的调用通过docker后产生了这一系列的反应。于是我想要将MHA环境跑在各数据节点上，而非docker容器中。

但是，现实再一次地证明了猜想的错误，即便是停止docker容器的运行，在虚拟机上直接运行MHA，依然会出现IO宕死的情况。

于是，我只能猜测是NFS或是VMWare的底层对MHA产生了影响。为了排除，我尝试新建了多台机器，最后的测试结果令我意外：裸搭和NFS都没有对MHA产生影响。

* + 1. 问题的解决

最后只能通过新建的裸搭环境和mysql51上的各种服务结合，排查出来是Mysql51机器出现了问题，当遇到多次调用的时候，原本的mysql51和mysql52总是会出现宕死的情况，果真是运维有三宝，重启重装换电脑。

* 1. 关于Docker的泛用性讨论
     1. MHA数据节点不能部署在docker平台上

最初因为MHA的架构比较庞大，且需要有一定拓展性，因此希望通过docker进行封装并且快速部署。于是开始做这个实验。但做到途中的时候发现，将MHA数据节点部署到docker中的时候，出现了调用的问题。

在启动MHA之前，各种试想实现得都非常顺利，但是在最后一步中失败，分析原因，得出以下结论：

MHA是一个组件型工具，通过ssh协议对集群中的计算机进行管理和分配，但是将MHA封装在docker内后，MHA对组件的调用会失败，因为MHA并不是一个通过端口映射从内部提供服务的方案，而是通过ssh协议连接后进行一系列组件和脚本的调用，因此，会出现下面的报错。将报错总结后，画出了下面的原理图：

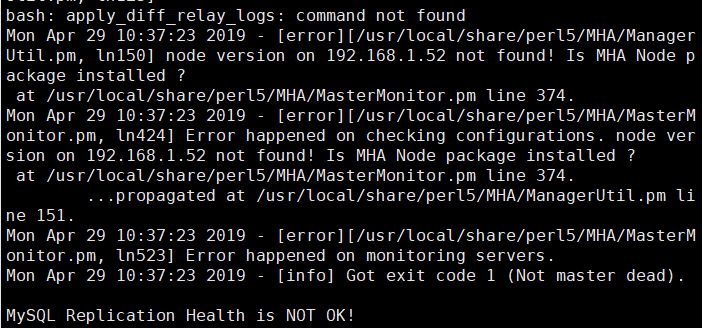


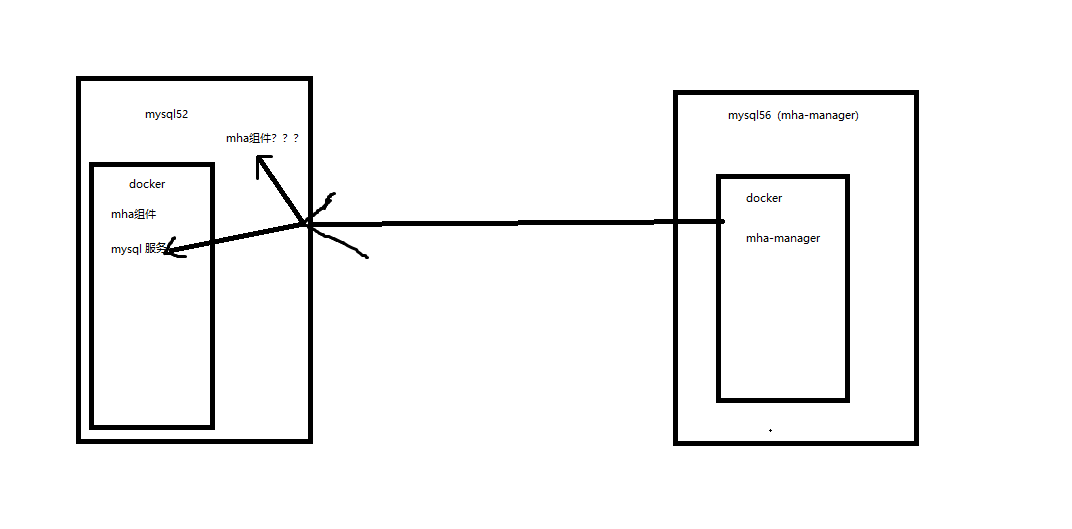
图4-1报错内容

图4-2 报错原理示意图

* + 1. docker技术的泛用性讨论

最初，认为docker的泛用性会十分广阔，凡事只要封装一个镜像后启动即可，但现在发现虚拟化对比docker所拥有的高隔离性在某些情况下拥有无可替代的作用。

docker技术会更加适合使用在服务和程序上，而像MHA这样通过ssh连接运行调用相应组件来使用的工具封装docker则会无法使用。

需要将docker和虚拟化技术结合起来，才能够最大程度上地优化架构的运行和使用，需要ansible辅助控制实现。

* + 1. docker镜像启动过程中出现的问题

在制作镜像的过程中，因为CentOS版本不同导致的包冲突问题进行了一段时间的研究，还有就是从头搭环境，配置文件过多导致思维混乱。

* 1. 本章小结

本章主要论述了主体实验步骤和MHA集群的部署测试。

1. 实验结果论述
   1. 关于Docker和VM虚拟化的对比

首先，虚拟化需要一个VM环境，无论是用KVM还是VMWare还是Hyper V都是依赖于GuestHost。而Docker可以直接作用于程序，这样可以节省大量的系统资源，更加利于环境的部署。如下表所示，对比单台机器使用docker和纯虚拟化对资源的占用：

表5-1 docker和虚拟化资源占用比

|  |  |  |
| --- | --- | --- |
|  | 磁盘 | 内存 |
| 虚拟化 | 9GB | 512MB+ |
| Docker | 3.3GB | 200-300MB |

因为我们需要6台，因此磁盘占用上需要乘上倍数，这样会放大他们的差距，虚拟化的资源占用会更多。因此，在运行的过程中，我们可以发现docker更加的节省资源。

经过第四章中的论述，我们已经知道了docker的局限性，MHA的节点数据是不能够封装在docker中的，这样一来，docker的容量会进一步减小，并且，因为时间的原因，我并没有去精简centos，如果去掉大部分的系统相关文件，镜像的大小会进一步减小。

但与此同时，需要注意的是，使用docker需要先准备好我们的配置文件和数据文件，以及MHA的相关安装文件，在启动docker的时候进行映射即可。

参考文献

[1] 个人Github(MHA高可用集群、Docke部分)

<https://github.com/xxd59366/StudyNote/tree/master/MyText/MyNote.docx>

[2] 《云计算架构技术与实践》 顾炯炯 清华大学出版社 2016-09-01

[3] 《云计算通俗讲义(第二版)》 王良明 电子工业出版社 2017-05-01

[4] VMware虚拟化与云计算应用案例详解（第2版）》 王春海 中国铁道出版社 2016-06-01

[5] 《云计算时代：本质、技术、创新、战略》 刘黎明 电子工业出版社 2014-06-01

[6]《Docker 容器与容器云（第2版）》 浙江大学软件工程实验室 人民邮电出版社 2016-10-01

[7]《Docker Documentation》 Docker官方文档

[8]《BGDMdocker: a Docker workflow for data mining and visualization of bacterial pan-genomes and biosynthetic gene clusters》 PeerJ Inc. 2017-11-30

[9]《源码搭建k8s-v1.10.5高可用集群》 仇宏帅 微信交流群 2019-2-22

[10]《Docker技术入门与实战 第3版》 杨保华 戴王剑 曹亚仑 机械工业出版社 2018-09-14

致谢

首先，感谢指导老师梁老师的帮助和建议，在老师敬业的态度和有效的建议下我才能将论文完成。其次，要感谢我实习时的前辈郭玉璞先生，在Linux运维方面很多的问题都是通过他了解并学习的，为这篇论文打下了理论基础。另外，感谢父母亲对我的关心，在这段日子里，他们多次询问我是否完成了毕业设计，没有他们的督促，疏懒的我可能到现在还没有开始。

一年前，自己萌生了学习服务端的想法，于是开始学习Linux，开始研究各种架构，父母没有多说什么，为我提供了钱和设备，母亲更是希望我能考出系统架构师。老郭在学习和工作中都给我相当多的帮助和支持。

这次的设计，原本只是想要将传统意义上的开发和服务端相结合，因此命名时也比较取巧，因为当时并没有决定到底要将哪一块结合起来。在学习服务端的过程中，我逐渐认识到自己以前的无知。在我的概念中，数据库就是数据库，却完全没有想过：想要承载高访问量需要面对的问题有哪些。之后在学习MHA的过程中，从主从同步到读写分离，我惊讶于架构的神奇和有趣。而MHA的过程相对来说较为繁琐，因此萌生了这么一个想法——将MHA封装为docker是否可以实现一步到位呢？

而到最后，我认识到了docker的局限性，不是什么东西都可以用docker封装后来使用的。整个过程中虽然充满了各种各样的问题，但是都慢慢探索出了问题所在，寻找出了解决方案。对自己而言，也是一次宝贵的实践经验和学习体会。

最后，再次感谢各位的陪伴和支持。