

# 实验 05：基于 Docker 部署

## 一、实验目的

- 1、了解 Docker;
- 2、了解 MongoDB 数据库;
- 3、掌握基于 Docker 部署 MongoDB 数据库服务;
- 4、掌握基于 Docker 实现 MongoDB 数据库集群;
- 5、掌握使用 MongoDB Compass 管理 MongoDB 数据库集群。

## 二、实验学时

2 学时

## 三、实验类型

综合性



## 四、实验需求

### 1、硬件

每人配备计算机 1 台。

### 2、软件

安装 VMware WorkStation Pro 或 Oracle VM VirtualBox 软件，安装 Mobaxterm 软件。

### 3、网络

本地主机与虚拟机能够访问互联网，虚拟机网络不使用 DHCP 服务。

Docker 能够同步访问网络。

### 4、工具

无。

## 五、实验任务

- 1、完成 MongoDB 数据库的安装;
- 2、完成 MongoDB 数据库集群的部署;
- 3、完成使用 MongoDB Compass 管理 MongoDB 数据库集群。

## 六、实验内容及步骤

- 1、本实验需要 VM 1 台。
- 2、本实验 VM 配置信息如下表所示。

| 虚拟机配置  | 操作系统配置  |
|--|---|
| 虚拟机名称: VM-Lab-05-Task-01-172.31.0.51<br>内存: 4GB<br>CPU: 4 颗, 1 核心<br>虚拟磁盘: 100GB<br>网卡: 1 块, NAT | 主机名: Lab-05-Task-01<br>IP 地址: 172.31.0.51<br>子网掩码: 255.255.255.0<br>网关: 172.31.0.254<br>DNS: 172.31.0.254 |

3、本实验拓扑图。

无。

4、本实验操作演示视频。

本实验操作演示视频为视频集的第 5 集：

<https://www.bilibili.com/video/BV1b1421t7aa?p=5>

## 七、实验内容步骤

### 1、安装 Docker

(1) 使用 VMware WorkStation Pro 创建实验所需虚拟机，并完成 openEuler 操作系统安装与基本配置（配置网络、开启远程连接），具体操作步骤请参考《实验 01：安装与基本配置》。

(2) 配置防火墙策略。

```
# 查看防火墙 Firewalld 服务状态
[root@Lab-05-Task-01 ~]# systemctl status firewalld
# 添加本地客户端允许远程连接 MongoDB 数据库
# mongo1 数据库通过 27017/tcp 连接
[root@Lab-05-Task-01 ~]# firewall-cmd --zone=public --add-port=27017/tcp --permanent
# mongo2 数据库通过 27018/tcp 连接
[root@Lab-05-Task-01 ~]# firewall-cmd --zone=public --add-port=27018/tcp --permanent
# mongo3 数据库通过 27019/tcp 连接
[root@Lab-05-Task-01 ~]# firewall-cmd --zone=public --add-port=27019/tcp --permanent
# 重新载入防火墙配置使其生效
[root@Lab-05-Task-01 ~]# firewall-cmd --reload
[root@Lab-05-Task-01 ~]# firewall-cmd --list-all
```

注意：

openEuler 操作系统默认安装 Firewalld 防火墙，并创建 firewalld 服务，该服务已开启且已配置为开机自启动。

(3) 安装 Docker 服务。

```
# 安装 dokcer
[root@Lab-05-Task-01 ~]# yum install -y docker
# 查看 Docker 版本信息
[root@Lab-05-Task-01 ~]# docker --version
```

(4) 开启 Docker 服务并设置服务开机自启。

```
# 启动 docker 服务
```

```
[root@Lab-05-Task-01 ~]# systemctl start docker
# 设置 docker 服务为开机自启动
[root@Lab-05-Task-01 ~]# systemctl enable docker
# 查看 docker 服务状态
[root@Lab-05-Task-01 ~]# systemctl status docker
```

## 2、安装 MongoDB 数据库

- (1) 使用 docker pull 命令拉取 MongoDB 镜像。

```
# 拉取 MongoDB 镜像
[root@Lab-05-Task-01 ~]# docker pull mongo
```

- (2) 使用 docker run 命令创建 3 个 MongoDB 容器。

```
# 创建并启动第 1 个 MongoDB 容器, 将容器 27017 端口映射到宿主机的 27017 端口
[root@Lab-05-Task-01 ~]# docker run -d --name mongo1 -v /usr/local/mongodb/datadb1:/data/db -v /usr/local/mongodb/key:/data/key -v /etc/localtime:/etc/localtime -p 27017:27017 mongo --replSet mongoddb-cluster
# 创建并启动第 2 个 MongoDB 容器, 将容器 27017 端口映射到宿主机的 27018 端口
[root@Lab-05-Task-01 ~]# docker run -d --name mongo2 -v /usr/local/mongodb/datadb2:/data/db -v /usr/local/mongodb/key:/data/key -v /etc/localtime:/etc/localtime -p 27018:27017 mongo --replSet mongoddb-cluster
# 创建并启动第 2 个 MongoDB 容器, 将容器 27017 端口映射到宿主机的 27019 端口
[root@Lab-05-Task-01 ~]# docker run -d --name mongo3 -v /usr/local/mongodb/datadb3:/data/db -v /usr/local/mongodb/key:/data/key -v /etc/localtime:/etc/localtime -p 27019:27017 mongo --replSet mongoddb-cluster
#查看容器是否创建
[root@Lab-05-Task-01 ~]# docker ps
```

| CONTAINER ID   | IMAGE         | COMMAND                  |
|----------------|---------------|--------------------------|
| CREATED        | STATUS        | PORTS                    |
| NAMES          |               |                          |
| 0d35ccb3c547   | mongo         | "docker-entrypoint.s..." |
| 7 seconds ago  | Up 6 seconds  | 0.0.0.0:27019->27017/tcp |
| mongo3         |               |                          |
| 54d66763668b   | mongo         | "docker-entrypoint.s..." |
| 12 seconds ago | Up 11 seconds | 0.0.0.0:27018->27017/tcp |
| mongo2         |               |                          |
| 632602ef0fa1   | mongo         | "docker-entrypoint.s..." |
| 21 seconds ago | Up 20 seconds | 0.0.0.0:27017->27017/tcp |
| mongo1         |               |                          |

注意:

若初次拉取 mongo 镜像失败, 出现错误:

```
error pulling image configuration: Get "https://production.cloudflare.docker.com/registry-v2/docker/registry/v2/blobs/sha256/a3/a31b196b
```

```
207d 768e78f2af331869e91d13443f691080d3b93e8009a53391eeaa/da
ta?verify=1721361822-bxHi6ad80PXuX8pRRws7gQDXVYU%3D": dial t
cp 168.143.171.189:443: c onnect: connection refused
```

需配置 Docker Hub 镜像加速器, Docker 官方和国内很多云服务商都提供了国内加速器服务, 具体参考配置如下:

①创建或修改 /etc/docker/daemon.json, 添加或修改内容:

```
{
  "data-root": "/data/dockerData",
  "registry-mirrors": [
    "https://registry.docker-cn.com",
    "http://hub-mirror.c.163.com",
    "https://dockerhub.azk8s.cn",
    "https://mirror.ccs.tencentyun.com",
    "https://registry.cn-hangzhou.aliyuncs.com",
    "https://docker.mirrors.ustc.edu.cn",
    "https://docker.m.daocloud.io",
    "https://noohub.ru",
    "https://huecker.io",
    "https://dockerhub.timeweb.cloud"
  ]
}
```

②保存重载 Docker 配置, 重启 Docker 服务

```
systemctl daemon-reload
```

```
systemctl restart docker
```

③配置完成后, 重新执行拉取镜像命令即可继续实验。

### 3、生成副本集密钥

进入到“/usr/local/mongodb/key”目录下, 生成 MongoDB 的副本集密钥。

```
[root@Lab-05-Task-01 ~]# cd /usr/local/mongodb/key
[root@Lab-05-Task-01 key]# openssl rand -base64 756 > mongodb.
key
```

### 4、在容器 mongo1 内配置副本集

(1) 使用 docker exec 命令进入到容器名为“mongo1”的容器内部, 并在容器内部安装 vim 编辑器。

```
# 进入到容器名为 “mongo1” 的容器内部
[root@Lab-05-Task-01 key]# docker exec -it mongo1 bash
# 安装 vim 编辑器
root@50080eb44de2:/# apt-get update
root@50080eb44de2:/# apt-get install -y vim
```

(2) 在容器名为“mongo1”的容器内部修改 mongdb 数据库配置文件。

```
root@50080eb44de2:/# cp /etc/mongod.conf.orig /etc/mongod.conf
# 使用 vim 命令编辑/etc/mongod.conf 文件
root@50080eb44de2:/# vim /etc/mongod.conf
# -----/etc/mongod.conf 文件-----
# 找到内容修改或添加
net:
port: 27017
```

```
bindIp: 0.0.0.0
security:
keyFile: /data/key/mongodb.key
replication:
replSetName: "mongodb-cluster"
# -----/etc/mongod.conf 文件-----
```

(3) 退出当前容器, 使用 `docker restart` 重启容器名为“mongo1”的 docker 容器, 确保配置生效。

```
# exit 命令退出当前容器
root@50080eb44de2:/# exit
# 重启 mongo1 容器
[root@Lab-05-Task-01 key]# docker restart mongo1
```

## 5、在容器 mongo2 内配置副本集

(1) 使用 `docker exec` 命令进入到容器名为“mongo2”的容器内部, 并在容器内部安装 vim 编辑器。

```
# 进入到容器名为 “mongo2” 的容器内部
[root@Lab-05-Task-01 key]# docker exec -it mongo2 bash
# 安装 vim 编辑器
root@9710b49a6a41:/# apt-get update
root@9710b49a6a41:/# apt-get install -y vim
```

(2) 在容器名为“mongo2”的容器内部修改 `mongodb` 数据库配置文件。

```
root@9710b49a6a41:/# cp /etc/mongod.conf.orig /etc/mongod.conf
# 使用 vim 命令编辑/etc/mongod.conf 文件
root@9710b49a6a41:/# vim /etc/mongod.conf
# -----/etc/mongod.conf 文件-----
# 找到内容修改或添加
net:
port: 27017
bindIp: 0.0.0.0
security:
keyFile: /data/key/mongodb.key
replication:
replSetName: "mongodb-cluster"
# -----/etc/mongod.conf 文件-----
```

(3) 退出当前容器, 使用 `docker restart` 重启容器名为“mongo2”的 docker 容器, 确保配置生效。

```
# exit 命令退出当前容器
root@9710b49a6a41:/# exit
# 重启 mongo2 容器
[root@Lab-05-Task-01 key]# docker restart mongo2
```

## 6、在容器 mongo3 内配置副本集

(1) 使用 `docker exec` 命令进入到容器名为“mongo3”的容器内部, 并在容器内部安装 vim 编辑器。

```
# 进入到容器名为 “mongo3” 的容器内部
```

```
[root@Lab-05-Task-01 key]# docker exec -it mongo3 bash
# 安装 vim 编辑器
root@30cefbae3722:/# apt-get update
root@30cefbae3722:/# apt-get install -y vim

root@30cefbae3722:/# cp /etc/mongod.conf.orig /etc/mongod.conf
```

- (2) 在容器名为“mongo3”的容器内部修改 mongod.conf 数据库配置文件。

```
# 使用 vim 命令编辑/etc/mongod.conf 文件
root@30cefbae3722:/# vim /etc/mongod.conf
# -----/etc/mongod.conf 文件-----
# 找到内容修改或添加
net:
port: 27017
bindIp: 0.0.0.0
security:
keyFile: /data/key/mongodb.key
replication:
replSetName: "mongodb-cluster"
# -----/etc/mongod.conf 文件-----
```

- (3) 退出当前容器，使用 docker restart 重启容器名为“mongo3”的 docker 容器，确保配置生效。

```
# exit 命令退出当前容器
root@30cefbae3722:/# exit
# 重启 mongo3 容器
[root@Lab-05-Task-01 key]# docker restart mongo3
```

## 7、在容器 mongo1 内初始化副本集

- (1) 使用 docker exec -it mongo1 mongosh 命令连接到容器名为“mongo1”的 MongoDB 客户端，初始化副本集，查看副本集状态并退出当前连接。

```
# 使用 docker exec -it mongo1 mongosh 命令连接 MongoDB 客户端
[root@Lab-05-Task-01 ~]# docker exec -it mongo1 mongosh
# 初始化副本集
> rs.initiate( {
  _id : "mongodb-cluster",
  members: [
    { _id:0, priority:2, host:"172.31.0.51:27017"},
    { _id:1, host:"172.31.0.51:27018"},
    { _id:2, host:"172.31.0.51:27019"}
  ]
})
#查看副本集状态
mongodb-cluster [direct: other] test> rs.status()
# 退出当前连接(primary 为主节点)
mongodb-cluster [direct: primary] test> quit()
```

- (2) 使用 docker exec -it mongo1 mongosh 命令连接 MongoDB 客户端，为副本集创建用户，并退出当前连接。

```
# 使用 docker exec -it mongo1 mongosh 命令连接 MongoDB 客户端
```

```
[root@Lab-05-Task-01 ~]# docker exec -it mongo1 mongosh
# 为副本集创建用户
mongodb-cluster [direct: primary] test> use admin
mongodb-cluster [direct: primary] admin> db.getSiblingDB("admin").c
reateUser(
{
  user: "mongodblab",
  pwd: "mongodblab#PWD",
  roles: [{role: "clusterAdmin",db: "admin"},"readWriteAnyDatabase"]
}
)
# 退出当前连接
mongodb-cluster [direct: primary] admin> quit()
```

注意:

通过上述 `rs.status()` 命令查看副本集状态, `mongo1 (172.31.0.51:27017)` 为主节点, `mongo2 (172.31.0.51:27018)`、`mongo3 (172.31.0.51:27019)` 为副本节点。

## 8、使用 MongoDB Compass 管理 MongoDB 数据库集群

- (1) 从 MongoDB Compass 的官方网站 (<https://www.mongodb.com>) 获取可执行程序。
- (2) 打开 MongoDB Compass 软件, 填写数据库连接信息如: `mongodb://172.31.0.51:27017`, 单击“connect”, 依次完成 3 个数据库的连接, 如图 5-1、5-2 所示。

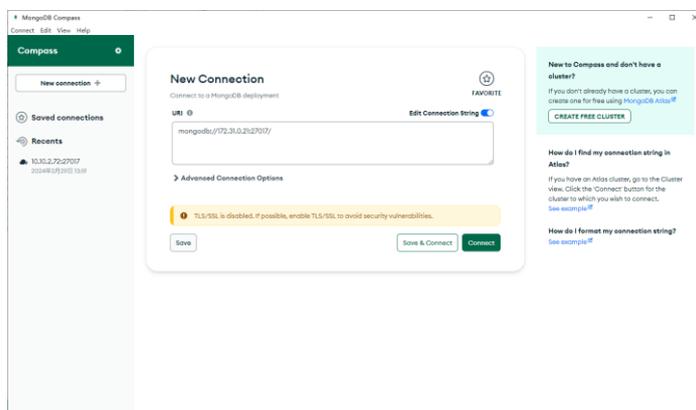


图 5-1 填写数据库连接信息

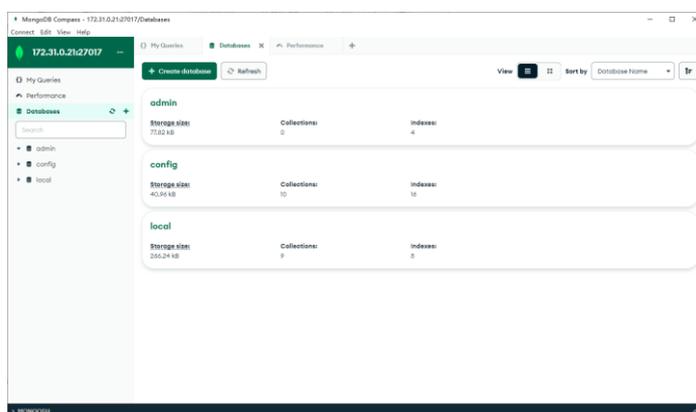


图 5-2 连接数据库

(3) 通过 MongoDB Compass 监控查看 3 个 MongoDB 数据库的信息。

单击“Performance”依次查看数据库的详细连接信息，具体监控参数本实验不再解释，可私下自主了解，监控数据如图 5-3 所示。

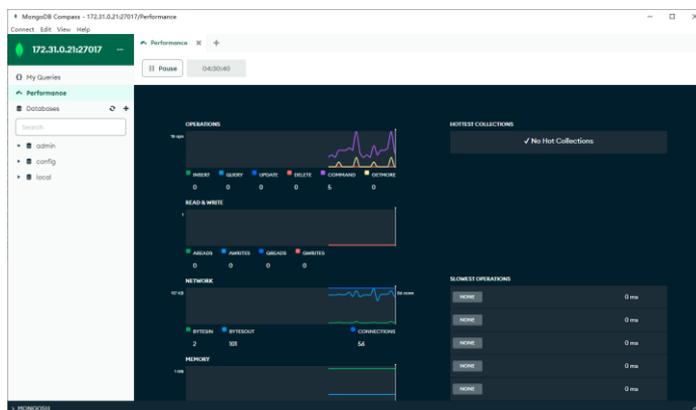


图 5-3 查看数据库监控信息

## 9、测试 MongoDB 副本集

### 9.1 测试方案

设计不同场景来测试 MongoDB 集群副本集。

| 场景  | 测试步骤                        |
|-----|-----------------------------|
| 场景一 | 为主节点数据库添加数据，查看从节点数据库是否新增数据。 |
| 场景二 | 从主节点数据库删除数据，查看从节点数据库是否删除数据。 |
| 场景三 | 模拟主节点宕机，查看新主节点是否存在。         |
| 场景四 | 恢复原主节点，查看宕机期间未同步的数据是否已同步    |

### 9.2 测试步骤

(1) 主节点增加数据，从节点同步增加。

使用 MongoDB Compass 通过用户“mongodblab”连接到主节点（172.31.0.51:27017）的 MongoDB 数据库。

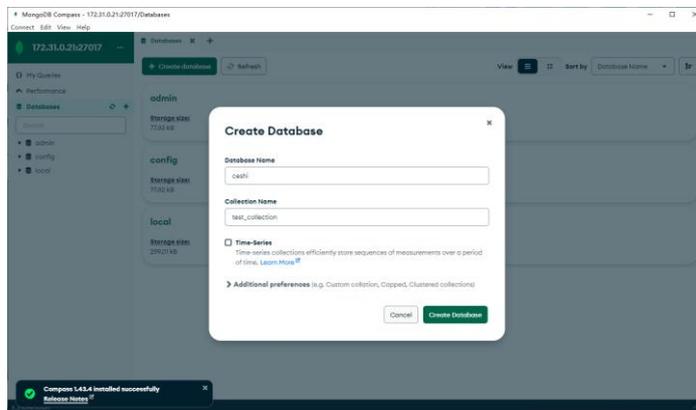


图 5-4 创建数据库和集合

## # 连接 URL

```
mongodb://mongodblab:mongodblab%23PWD@172.31.0.51:27017
```

创建数据库“ceshi”和集合“test\_collection”，如图 5-4 所示。

单击“ADD DATA”，选择“Insert Document”，添加默认数据（本实验不再具体解释数据添加过程，可私下自主学习），如图 5-5 所示。

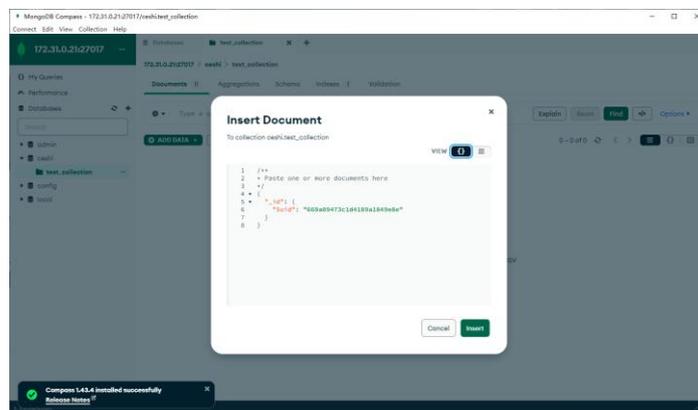


图 5-5 添加数据

添加完成，依次使用 MongoDB Compass 通过用户“mongodblab”连接到副本节点（172.31.0.51:27018 / 172.31.0.51:27019）的 MongoDB 数据库，查看在主节点上创建的数据库、集合以及添加的数据，是否存在。

(2) 主节点删除数据，从节点同步删除。

参照上述步骤使用 MongoDB Compass 连接主节点数据库，选中需删除数据右侧的删除按钮，执行数据删除操作，如图 5-6 所示。

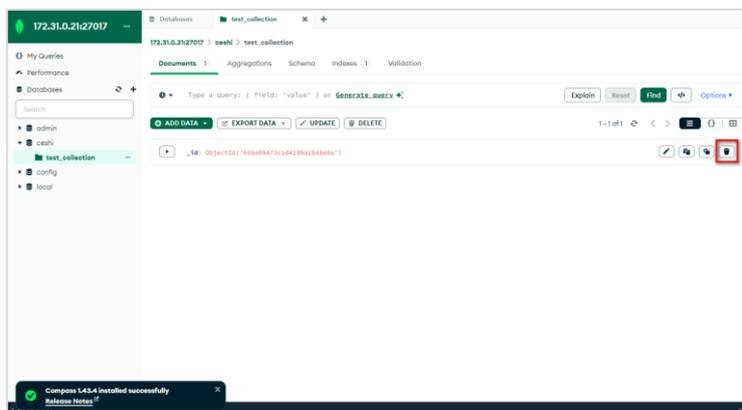


图 5-6 删除数据

删除完成，依次使用 MongoDB Compass 通过用户“mongodblab”连接到副本节点（172.31.0.51:27018 / 172.31.0.51:27019）的 MongoDB 数据库，查看在主节点上删除的数据，是否在本节点已删除。

(3) 主节点宕机，集群新选举主节点。

关闭主节点容器，模拟主节点宕机故障。

依次使用 `docker exec -it mongo2 mongosh` 命令连接副本节点（172.31.0.51:27018 / 172.31.0.51:27019）的 MongoDB 数据库客户端，查看当前主节点所在的容器

```
# 停止主节点对应容器
[root@Lab-05-Task-01 ~]# docker stop mongo1
# 连接 MongoDB 客户端
[root@Lab-05-Task-01 ~]# docker exec -it mongo2 mongosh
# 查看当前主节点所在的容器
mongodb-cluster [direct: secondary] test> rs.status()
```

(4) 原主节点恢复正常，宕机期间的数据实现同步。

使用 MongoDB Compass 通过用户“mongodblab”连接到现主节点 (172.31.0.51:27018)，按照上述步骤添加 1 条数据。

启动原主节点对应容器，模拟原主节点 (172.31.0.51:27017) 恢复正常。

使用 `docker exec -it mongo1 mongosh` 命令连接原主节点，并查看副本集状态。

使用 MongoDB Compass 通过用户“mongodblab”连接到原主节点 (172.31.0.51:27017)，查看宕机期间未同步的数据是否已同步。

```
# 启动原主节点对应容器
[root@Lab-05-Task-01 ~]# docker start mongo1
# 连接 MongoDB 客户端
[root@Lab-05-Task-01 ~]# docker exec -it mongo1 mongosh
# 查看副本集状态
mongodb-cluster [direct: secondary] test> rs.status()
```

### 9.3 测试结果

| 场景  | 测试步骤                        | 真实测试结果      |
|-----|-----------------------------|-------------|
| 场景一 | 为主节点数据库添加数据，查看从节点数据库是否新增数据。 | 从节点实现数据同步增加 |
| 场景二 | 从主节点数据库删除数据，查看从节点数据库是否删除数据。 | 从节点实现数据同步删除 |
| 场景三 | 模拟主节点宕机，查看新主节点是否存在。         | 集群重新选取主节点   |
| 场景四 | 恢复原主节点，查看宕机期间未同步的数据是否已同步。   | 数据已同步       |

## 八、实验考核

实验考核分为【实验随堂查】和【实验线上考】两个部分。

实验随堂查：每个实验设置 2-5 考核点。完成实验任务后，任课教师随机选择一个考核点，学生现场进行演示和汇报讲解。

实验线上考：每个实验设置 10 道客观题。通过线上考核平台（如课堂派）进行作答。

### 1、实验随堂查

本实验随堂查设置 3 个考核点，具体如下：

考核点 1：使用 MongoDB Compass 管理 MongoDB

考核点 2：解读 MongoDB Compass 监控信息

考核点 3：实现 MongoDB 集群数据同步

### 2、实验线上考

本实验线上考共 10 题，题型为单选、多选、判断、填空等题型。