五、MongoDB部署分片服务器

5.1、 分片简介 ----类似分布式

5.2、 分片优势

5.3、分片应用环境

5.4、分片架构及组件

5.5、MongoDB分片集群部署

1、所有主机关闭防火墙及Selinux

2、操作系统采用Centos7.x、MongoDB4.0.6

3、主机及端口规划如下:

4、拓扑架构说明

5、部署分片集群

5.1、下载部署MongoDB(3台服务器执行相同操作)

5.2、创建路由、配置、日志、分片等的相关目录与文件(3台服务器执行相同操作)

5.3、准备配置实例配置文件

在/usr/local/mongodb/conf/ 目录创建config.conf配置文件(3台服务器执行相同操作)

5.4、三台服务器分别启动配置服务实例(3台服务器执行相同操作)

5.5、连接mongo构建配置服务复制集

5.6、准备分片服务的相关文件

在/usr/ocal/mongodb/conf/ 目录下创建shard1.conf、shard2.conf、 shard3.conf 分片实例配置文件(3台服务器执行相同操作)。

5.7、将分片配置为复制集

5.8、路由服务部署(3台服务器执行相同操作)

5.9 启动mongos

5.10 启动分片功能

5.11 测试分片功能 --了解

1） Archive模块

2) cmd模块 ---命令、脚本 ----必须会!

3) cp模块

5) dnsutil 模块 --管理host文件的

6) file模块 ---功能非常强大

8) network 模块

9) pkg 模块 ---自动匹配版本系统（ubantu,centos）

10) service 模块

11)其他模块

五、MongoDB部署分片服务器

5.1、 分片简介 ----类似分布式

之前的MongoDB的复制没有实验，因为它没有故障自动切换，只能数据异地同步

复制集类似redis的哨兵集，或者MHA

分片也属于MongoDB集群技术，分片目的为了突破单点数据库服务器的I/O能力限制，对数据库存储进行水平扩展，满足MongoDB数据量大量增长的需求。严格地说，每一个服务器或者实例或者复制集就是一个分片。

MongoDB处理存储海量的数据时，一台机器可能不足以存储数据，也可能不足以提供可接受的读写吞吐量。这时，我们就可以通过在多台机器上分割数据，使得数据库系统能存储和处理更多的数据。

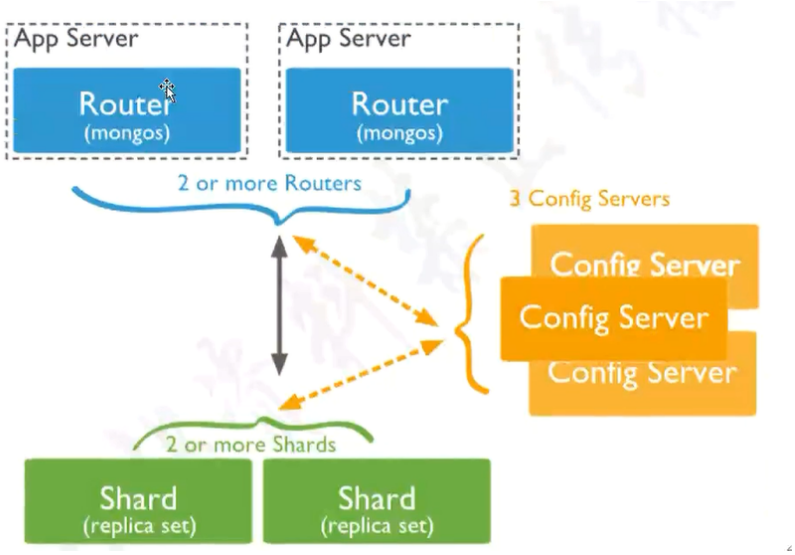
5.2、 分片优势

- 提供类似现行增长架构
  提高数据可用性
  提高大型数据库查询服务器性能
- 单个副本集限制在12个节点
- 垂直扩展价格昂贵

5.3、分片应用环境

- 单点数据库服务器存储成为瓶颈
- 单点数据库服务器的性能成为瓶颈
- 大型应用分散数据库已充分利用内存

5.4、分片架构及组件



分片三个主要组件:

● Shard:

用于存储实际的数据块，实际生产环境中一个shardserver角色可由几台机器组一个replicaSet承担，防止主机单点故障。

● Config Server:

mongod实例，存储了整个Cluster Metadata(元数据)，其中包括chunk (块)信息。

● Query Routers:

前端路由，客户端由此接入，且让整个集群看上去像单一数据库，前端应用可以透明使用。

生成环境实现分片，至少三台路由实例，至少三台配置实例，每一分片都是一个副本集。

5.5、MongoDB分片集群部署

1、所有主机关闭防火墙及Selinux

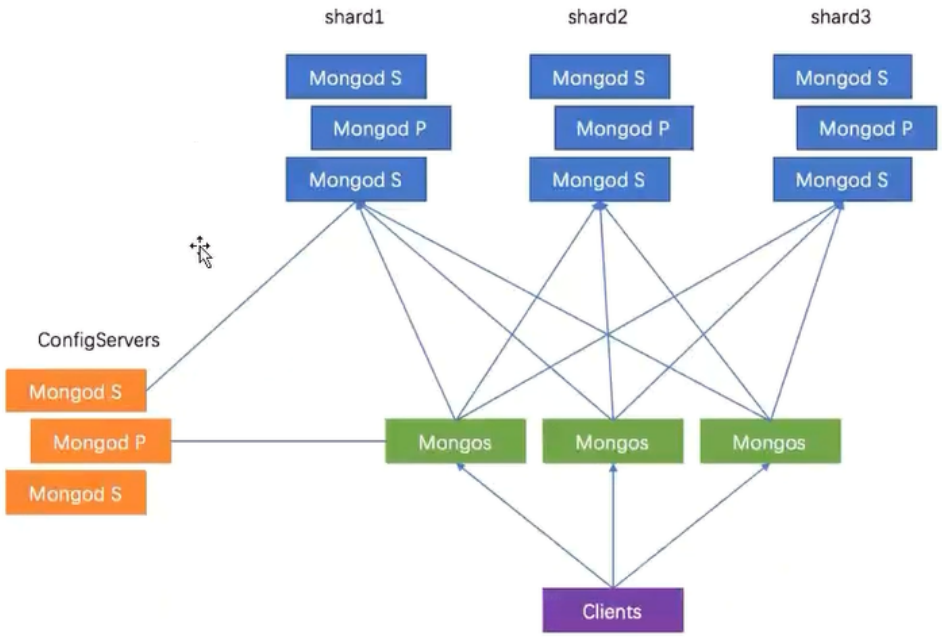
2、操作系统采用Centos7.x、MongoDB4.0.6

3、主机及端口规划如下:

IP地址	路由服务端口	配置服务端口	分片1端口	分片2端口	分片3端口
192.168.200.107	27017	27018	27001	27002	27003
192.168.200.108	27017	27018	27001	27002	27003
192.168.200.109	27017	27018	27001	27002	27003

4、拓扑架构说明

三台机器的配置服务(27018)形成配置服务的复制集，分片1、2、3也在各机器都部署一个实例， 它们之间形成复制集，客户端直接连接3个路由服务与之交互，配置服务和分片服务对客户端是透明的。



5、部署分片集群

5.1、下载部署MongoDB(3台服务器执行相同操作)

设置主机名关闭防火墙和selinuxe

三台都做：

```
[root@localhost ~]# rz
[root@localhost ~]# ls
mongodb-linux-x86_64-rhel70-4.0.6.tgz
[root@localhost ~]# iptables -F
[root@localhost ~]# setenforce 0
setenforce: SELinux is disabled
[root@localhost ~]# systemctl stop firewalld
```

解压到/usr/local/mongodb，设置环境变量:

```
[root@localhost ~]# hostname mongodb1
[root@localhost ~]# bash
[root@localhost ~]# hostname mongodb2
[root@localhost ~]# bash
[root@localhost ~]# hostname mongodb3
[root@localhost ~]# bash
```

解压到/usr/local/mongodb，设置环境变量:

```
[root@mongodb1 ~]# tar xf mongodb-linux-x86_64-rhel70-4.0.6.tgz
[root@mongodb1 ~]# mv mongodb-linux-x86_64-rhel70-4.0.6 /usr/local/mongodb
[root@mongodb1 ~]# cat << END > /etc/profile
> export PATH=$PATH:/usr/local/mongodb/bin/
> END
```

```
[root@mongodb1 ~]# source /etc/profile
```

5.2、创建路由、配置、日志、分片等的相关目录与文件(3台服务器执行相同操作)

创建配置文件存放目录: /usr/local/mongodb/conf

各类日志存放目录: /usr/local/mongodb/logs/

配置服务数据存放目录: /usr/local/mongodb/data/config

分片1服务数据存放目录: /usr/local/mongodb/data/shard1

分片2服务数据存放目录: /usr/local/mongodb/data/shard2

分片3服务数据存放目录: /usr/local/mongodb/data/shard3

配置服务日志存放文件: /us/local/mongodb/logs/config.log

路由服务日志存放文件: /usr/local/mongodb/logs/mongos.loge

分片1服务日志存放文件: /usr/local/mongodb/logs/shard1.log

分片2服务日志存放文件: /usr/local/mongodb/logs/shard2.log

分片3服务日志存放文件: /usr/local/mongodb/logs/shard3.log

三台机器同样的操作：

```
[root@mongodb1 ~]# mkdir -p /usr/local/mongodb/conf
[root@mongodb1 ~]# mkdir -p /usr/local/mongodb/logs/
[root@mongodb1 ~]# mkdir -p /usr/local/mongodb/data/config
[root@mongodb1 ~]# mkdir -p /usr/local/mongodb/data/shard1
[root@mongodb1 ~]# mkdir -p /usr/local/mongodb/data/shard2
[root@mongodb1 ~]# mkdir -p /usr/local/mongodb/data/shard3
[root@mongodb1 ~]# touch /usr/local/mongodb/logs/config.log
[root@mongodb1 ~]# touch /usr/local/mongodb/logs/mongos.log
[root@mongodb1 ~]# touch /usr/local/mongodb/logs/shard1.log
[root@mongodb1 ~]#  touch /usr/local/mongodb/logs/shard2.log
[root@mongodb1 ~]#  touch /usr/local/mongodb/logs/shard3.log
```

5.3、准备配置实例配置文件

在/usr/local/mongodb/conf/ 目录创建config.conf配置文件(3台服务器执行相同操作)

```
[root@mongodb1 ~]# cat << END > /usr/local/mongodb/conf/config.conf
dbpath=/usr/local/mongodb/data/config
logpath=/usr/local/mongodb/logs/config.log
port=27018
logappend=true
fork=true
maxConns=5000
replSet=configs
configsvr=true
```

```
bind_ip=0.0.0.0  # 生产环境中，配有效IP（公网IP）
END
[root@mongodb1 ~]# scp cat /usr/local/mongodb/conf/config.conf
192.168.200.108:/usr/local/mongodb/conf/
[root@mongodb1 ~]# scp /usr/local/mongodb/conf/config.conf
192.168.200.109:/usr/local/mongodb/conf/
```

5.4、三台服务器分别启动配置服务实例(3台服务器执行相同操作)

```
[root@mongodb1 ~]# mongod -f /usr/local/mongodb/conf/config.conf
about to fork child process, waiting until server is ready for connections.
forked process: 1716
child process started successfully, parent exiting
[root@mongodb1 ~]# netstat -lnpt | grep 27018
tcp        0      0 0.0.0.0:27018           0.0.0.0:*               LISTEN
2433/mongod
```

5.5、连接mongo构建配置服务复制集

```
[root@mongodb1 ~]# mongo --host 192.168.200.107 -port 27018
> use admin
switched to db admin
初始化复制集（一步搞定）
> rs.initiate({_id:"configs",members:[{_id:0,host:"192.168.200.107:27018"},
{_id:1,host:"192.168.200.108:27018"},{_id:2,hos
t:"192.168.200.109:27018"}]}){
        "ok" : 1,
        "operationTime" : Timestamp(1590343209, 1),
        "$gleStats" : {
                "lastOpTime" : Timestamp(1590343209, 1),
                "electionId" : ObjectId("000000000000000000000000")
        },
        "lastCommittedOpTime" : Timestamp(0, 0),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1590343209, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

查看状态

```
configs:PRIMARY> rs.status()
{
        "set" : "configs",
        "date" : ISODate("2020-05-24T18:01:16.367Z"),
        "myState" : 1,
        "term" : NumberLong(1),
        "syncingTo" : "",
        "syncSourceHost" : "",
        "syncSourceId" : -1,
        "configsvr" : true,
        "heartbeatIntervalMillis" : NumberLong(2000),
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(1590343272, 1),
                        "t" : NumberLong(1)
                },
                "readConcernMajorityOpTime" : {
                        "ts" : Timestamp(1590343272, 1),
                        "t" : NumberLong(1)
                },
                "appliedOpTime" : {
                        "ts" : Timestamp(1590343272, 1),
                        "t" : NumberLong(1)
                },
                "durableOpTime" : {
                        "ts" : Timestamp(1590343272, 1),
                        "t" : NumberLong(1)
                }
        },
        "lastStableCheckpointTimestamp" : Timestamp(1590343222, 1),
        "members" : [
                {
                        "_id" : 0,
                        "name" : "192.168.200.107:27018",
                        "health" : 1,
                        "state" : 1,
```

```
        "stateStr" : "PRIMARY",
        "uptime" : 3590,
        "optime" : {
                "ts" : Timestamp(1590343272, 1),
                "t" : NumberLong(1)
        },
        "optimeDate" : ISODate("2020-05-24T18:01:12Z"),
        "syncingTo" : "",
        "syncSourceHost" : "",
        "syncSourceId" : -1,
        "infoMessage" : "could not find member to sync from",
        "electionTime" : Timestamp(1590343221, 1),
        "electionDate" : ISODate("2020-05-24T18:00:21Z"),
        "configVersion" : 1,
        "self" : true,
        "lastHeartbeatMessage" : ""
},
{
        "_id" : 1,
        "name" : "192.168.200.108:27018",
        "health" : 1,
        "state" : 2,
        "stateStr" : "SECONDARY",
        "uptime" : 66,
        "optime" : {
                "ts" : Timestamp(1590343272, 1),
                "t" : NumberLong(1)
        },
        "optimeDurable" : {
                "ts" : Timestamp(1590343272, 1),
                "t" : NumberLong(1)
        },
        "optimeDate" : ISODate("2020-05-24T18:01:12Z"),
        "optimeDurableDate" : ISODate("2020-05-24T18:01:12Z"),
        "lastHeartbeat" : ISODate("2020-05-24T18:01:15.360Z"),
        "lastHeartbeatRecv" : ISODate("2020-05-24T18:01:14.772Z"),
        "pingMs" : NumberLong(0),
```

```
                "lastHeartbeatMessage" : "",
                "syncingTo" : "192.168.200.107:27018",
                "syncSourceHost" : "192.168.200.107:27018",
                "syncSourceId" : 0,
                "infoMessage" : "",
                "configVersion" : 1
        },
        {
                "_id" : 2,
                "name" : "192.168.200.109:27018",
                "health" : 1,
                "state" : 2,
                "stateStr" : "SECONDARY",
                "uptime" : 66,
                "optime" : {
                        "ts" : Timestamp(1590343272, 1),
                        "t" : NumberLong(1)
                },
                "optimeDurable" : {
                        "ts" : Timestamp(1590343272, 1),
                        "t" : NumberLong(1)
                },
                "optimeDate" : ISODate("2020-05-24T18:01:12Z"),
                "optimeDurableDate" : ISODate("2020-05-24T18:01:12Z"),
                "lastHeartbeat" : ISODate("2020-05-24T18:01:15.357Z"),
                "lastHeartbeatRecv" : ISODate("2020-05-24T18:01:14.772Z"),
                "pingMs" : NumberLong(0),
                "lastHeartbeatMessage" : "",
                "syncingTo" : "192.168.200.107:27018",
                "syncSourceHost" : "192.168.200.107:27018",
                "syncSourceId" : 0,
                "infoMessage" : "",
                "configVersion" : 1
        }
],
"ok" : 1,
"operationTime" : Timestamp(1590343272, 1),
```

```
        "$gleStats" : {
                "lastOpTime" : Timestamp(1590343209, 1),
                "electionId" : ObjectId("7fffffff0000000000000001")
        },
        "lastCommittedOpTime" : Timestamp(1590343272, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1590343272, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

到此配置服务的复制集就搞定~

三台机器的配置服务就已形成复制集，其中1台（107）为PRIMARY,其他2台（108.109）为SECONDARY。

5.6、准备分片服务的相关文件

在/usr/ocal/mongodb/conf/ 目录下创建shard1.conf、shard2.conf、 shard3.conf 分片实例配置文件(3台服务器执行相同操作)。

```
[root@mongodb1 ~]# vim /usr/local/mongodb/conf/shard1.conf          // 其他实例修改分片名称
dbpath=/usr/local/mongodb/data/shard1
logpath=/usr/local/mongodb/logs/shard1.log
port=27001
logappend=true
fork=true
maxConns=5000
storageEngine=mmapv1
shardsvr=true
replSet=shard1
bind_ip=0.0.0.0
[root@mongodb1 ~]# vim /usr/local/mongodb/conf/shard2.conf
dbpath=/usr/local/mongodb/data/shard2
logpath=/usr/local/mongodb/logs/shard2.log
port=27002
logappend=true
fork=true
```

```
maxConns=5000
storageEngine=mmapv1
shardsvr=true
replSet=shard2
bind_ip=0.0.0.0
[root@mongodb1 ~]# vim /usr/local/mongodb/conf/shard3.conf
dbpath=/usr/local/mongodb/data/shard3
logpath=/usr/local/mongodb/logs/shard3.log
port=27003
logappend=true
fork=true
maxConns=5000
storageEngine=mmapv1
shardsvr=true
replSet=shard3
bind_ip=0.0.0.0
[root@mongodb1 ~]# scp /usr/local/mongodb/conf/shard{1,2,3}.conf
192.168.200.108:/usr/local/mongodb/conf/
root@192.168.200.108's password:
shard1.conf
100%  197    211.8KB/s   00:00
shard2.conf
100%  207    208.3KB/s   00:00
shard3.conf
100%  207    269.0KB/s   00:00
[root@mongodb1 ~]# scp /usr/local/mongodb/conf/shard{1,2,3}.conf
192.168.200.109:/usr/local/mongodb/conf/
root@192.168.200.109's password:
shard1.conf
100%  197    160.2KB/s   00:00
shard2.conf
100%  207    160.7KB/s   00:00
shard3.conf
100%  207    133.9KB/s   00:00
```

以上分片实例端口分别是27001、 27002、 27003， 分别对应了shard1.conf、 shard2.conf、 shard3.conf等文件。稍后会在3台机器的相同端口上形成一个分片服务的复制集，由于3台机器都需要这3个文件，所以根据这9个配置文件分别启动分片服务

三台机器相同的操作：

```
[root@mongodb1 ~]# mongod -f /usr/local/mongodb/conf/shard1.conf
about to fork child process, waiting until server is ready for connections.
forked process: 3265
child process started successfully, parent exiting
[root@mongodb1 ~]# mongod -f /usr/local/mongodb/conf/shard2.conf
about to fork child process, waiting until server is ready for connections.
forked process: 3295
child process started successfully, parent exiting
[root@mongodb1 ~]# mongod -f /usr/local/mongodb/conf/shard3.conf
about to fork child process, waiting until server is ready for connections.
forked process: 3319
child process started successfully, parent exiting
[root@mongodb1 ~]# netstat -lnpt | grep -E "2700[1-3]"
tcp        0      0 0.0.0.0:27001           0.0.0.0:*               LISTEN
3265/mongod
tcp        0      0 0.0.0.0:27002           0.0.0.0:*               LISTEN
3295/mongod
tcp        0      0 0.0.0.0:27003           0.0.0.0:*               LISTEN
3319/mongod
```

到此关于分片的9个实例，都搞定！

5.7、将分片配置为复制集

连接mongo，只需在任意一台机器执行即可，分别连接到27001、27002、27003的端口进行复制集配置

```
[root@mongodb1 ~]# mongo --host 192.168.200.107 --port 27001
```

切换数据库

```
> use admin
switched to db admin
```

初始化复制集

```
> rs.initiate({_id:"shard1",members:[{_id:0,host:"192.168.200.107:27001"},
{_id:1,host:"192.168.200.108:27001"},{_id:2,host:"192.168.200.109:27001"}]})
{
        "ok" : 1,
        "operationTime" : Timestamp(1590387327, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1590387327, 1),
                "signature" : {
```

```
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
shard1:SECONDARY>
shard1:PRIMARY>
shard1:PRIMARY> rs.status()
{
        "set" : "shard1",
        "date" : ISODate("2020-05-25T06:16:39.017Z"),
        "myState" : 1,
        "term" : NumberLong(2),
        "syncingTo" : "",
        "syncSourceHost" : "",
        "syncSourceId" : -1,
        "heartbeatIntervalMillis" : NumberLong(2000),
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(1590387395, 1),
                        "t" : NumberLong(2)
                },
                "readConcernMajorityOpTime" : {
                        "ts" : Timestamp(1590387395, 1),
                        "t" : NumberLong(2)
                },
                "appliedOpTime" : {
                        "ts" : Timestamp(1590387395, 1),
                        "t" : NumberLong(2)
                },
                "durableOpTime" : {
                        "ts" : Timestamp(1590387395, 1),
                        "t" : NumberLong(2)
                }
        },
        "members" : [
                {
```

```
                "_id" : 0,
                "name" : "192.168.200.107:27001",
                "health" : 1,
                "state" : 1,
                "stateStr" : "PRIMARY",
                "uptime" : 42999,
                "optime" : {
                        "ts" : Timestamp(1590387395, 1),
                        "t" : NumberLong(2)
                },
                "optimeDate" : ISODate("2020-05-25T06:16:35Z"),
                "syncingTo" : "",
                "syncSourceHost" : "",
                "syncSourceId" : -1,
                "infoMessage" : "could not find member to sync from",
                "electionTime" : Timestamp(1590387354, 1),
                "electionDate" : ISODate("2020-05-25T06:15:54Z"),
                "configVersion" : 1,
                "self" : true,
                "lastHeartbeatMessage" : ""
        },
        {
                "_id" : 1,
                "name" : "192.168.200.108:27001",
                "health" : 1,
                "state" : 2,
                "stateStr" : "SECONDARY",
                "uptime" : 70,
                "optime" : {
                        "ts" : Timestamp(1590387395, 1),
                        "t" : NumberLong(2)
                },
                "optimeDurable" : {
                        "ts" : Timestamp(1590387395, 1),
                        "t" : NumberLong(2)
                },
                "optimeDate" : ISODate("2020-05-25T06:16:35Z"),
```

```
        "optimeDurableDate" : ISODate("2020-05-25T06:16:35Z"),
        "lastHeartbeat" : ISODate("2020-05-25T06:16:37.201Z"),
        "lastHeartbeatRecv" : ISODate("2020-05-25T06:16:38.046Z"),
        "pingMs" : NumberLong(0),
        "lastHeartbeatMessage" : "",
        "syncingTo" : "192.168.200.107:27001",
        "syncSourceHost" : "192.168.200.107:27001",
        "syncSourceId" : 0,
        "infoMessage" : "",
        "configVersion" : 1
},
{
        "_id" : 2,
        "name" : "192.168.200.109:27001",
        "health" : 1,
        "state" : 2,
        "stateStr" : "SECONDARY",
        "uptime" : 70,
        "optime" : {
                "ts" : Timestamp(1590387395, 1),
                "t" : NumberLong(2)
        },
        "optimeDurable" : {
                "ts" : Timestamp(1590387395, 1),
                "t" : NumberLong(2)
        },
        "optimeDate" : ISODate("2020-05-25T06:16:35Z"),
        "optimeDurableDate" : ISODate("2020-05-25T06:16:35Z"),
        "lastHeartbeat" : ISODate("2020-05-25T06:16:37.454Z"),
        "lastHeartbeatRecv" : ISODate("2020-05-25T06:16:37.683Z"),
        "pingMs" : NumberLong(3),
        "lastHeartbeatMessage" : "",
        "syncingTo" : "192.168.200.107:27001",
        "syncSourceHost" : "192.168.200.107:27001",
        "syncSourceId" : 0,
        "infoMessage" : "",
        "configVersion" : 1
```

```
                }
        ],
        "ok" : 1,
        "operationTime" : Timestamp(1590387395, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1590387395, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

另外两个分片：

```
[root@mongodb1 ~]# mongo --host 192.168.200.107 --port 27002
> use admin
switched to db admin
> rs.initiate({_id:"shard2",members:[{_id:0,host:"192.168.200.107:27002"},
{_id:1,host:"192.168.200.108:27002"},{_id:2,host
:"192.168.200.109:27002"}]}){
        "ok" : 1,
        "operationTime" : Timestamp(1590387735, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1590387735, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
shard2:SECONDARY>
shard2:PRIMARY>
shard2:PRIMARY> rs.status()
{
        "set" : "shard2",
        "date" : ISODate("2020-05-25T06:23:13.415Z"),
        "myState" : 1,
        "term" : NumberLong(1),
```

```
"syncingTo" : "",
"syncSourceHost" : "",
"syncSourceId" : -1,
"heartbeatIntervalMillis" : NumberLong(2000),
"optimes" : {
        "lastCommittedOpTime" : {
                "ts" : Timestamp(1590387785, 1),
                "t" : NumberLong(1)
        },
        "readConcernMajorityOpTime" : {
                "ts" : Timestamp(1590387785, 1),
                "t" : NumberLong(1)
        },
        "appliedOpTime" : {
                "ts" : Timestamp(1590387785, 1),
                "t" : NumberLong(1)
        },
        "durableOpTime" : {
                "ts" : Timestamp(1590387785, 1),
                "t" : NumberLong(1)
        }
},
"members" : [
        {
                "_id" : 0,
                "name" : "192.168.200.107:27002",
                "health" : 1,
                "state" : 1,
                "stateStr" : "PRIMARY",
                "uptime" : 43388,
                "optime" : {
                        "ts" : Timestamp(1590387785, 1),
                        "t" : NumberLong(1)
                },
                "optimeDate" : ISODate("2020-05-25T06:23:05Z"),
                "syncingTo" : "",
                "syncSourceHost" : "",
```

```
        "syncSourceId" : -1,
        "infoMessage" : "could not find member to sync from",
        "electionTime" : Timestamp(1590387751, 1),
        "electionDate" : ISODate("2020-05-25T06:22:31Z"),
        "configVersion" : 1,
        "self" : true,
        "lastHeartbeatMessage" : ""
},
{
        "_id" : 1,
        "name" : "192.168.200.108:27002",
        "health" : 1,
        "state" : 2,
        "stateStr" : "SECONDARY",
        "uptime" : 57,
        "optime" : {
                "ts" : Timestamp(1590387785, 1),
                "t" : NumberLong(1)
        },
        "optimeDurable" : {
                "ts" : Timestamp(1590387785, 1),
                "t" : NumberLong(1)
        },
        "optimeDate" : ISODate("2020-05-25T06:23:05Z"),
        "optimeDurableDate" : ISODate("2020-05-25T06:23:05Z"),
        "lastHeartbeat" : ISODate("2020-05-25T06:23:12.022Z"),
        "lastHeartbeatRecv" : ISODate("2020-05-25T06:23:12.021Z"),
        "pingMs" : NumberLong(1),
        "lastHeartbeatMessage" : "",
        "syncingTo" : "192.168.200.107:27002",
        "syncSourceHost" : "192.168.200.107:27002",
        "syncSourceId" : 0,
        "infoMessage" : "",
        "configVersion" : 1
},
{
        "_id" : 2,
```

```
                        "name" : "192.168.200.109:27002",
                        "health" : 1,
                        "state" : 2,
                        "stateStr" : "SECONDARY",
                        "uptime" : 57,
                        "optime" : {
                                "ts" : Timestamp(1590387785, 1),
                                "t" : NumberLong(1)
                        },
                        "optimeDurable" : {
                                "ts" : Timestamp(1590387785, 1),
                                "t" : NumberLong(1)
                        },
                        "optimeDate" : ISODate("2020-05-25T06:23:05Z"),
                        "optimeDurableDate" : ISODate("2020-05-25T06:23:05Z"),
                        "lastHeartbeat" : ISODate("2020-05-25T06:23:11.999Z"),
                        "lastHeartbeatRecv" : ISODate("2020-05-25T06:23:11.990Z"),
                        "pingMs" : NumberLong(0),
                        "lastHeartbeatMessage" : "",
                        "syncingTo" : "192.168.200.107:27002",
                        "syncSourceHost" : "192.168.200.107:27002",
                        "syncSourceId" : 0,
                        "infoMessage" : "",
                        "configVersion" : 1
                }
        ],
        "ok" : 1,
        "operationTime" : Timestamp(1590387785, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1590387785, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
[root@mongodb1 ~]# mongo --host 192.168.200.107 --port 27003
```

```
> use admin
switched to db admin
> rs.initiate({_id:"shard3",members:[{_id:0,host:"192.168.200.107:27003"},
{_id:1,host:"192.168.200.108:27003"},{_id:2,host
:"192.168.200.109:27003"}]}){
        "ok" : 1,
        "operationTime" : Timestamp(1590387901, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1590387901, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
shard3:PRIMARY>  rs.status()
{
        "set" : "shard3",
        "date" : ISODate("2020-05-25T06:25:20.975Z"),
        "myState" : 1,
        "term" : NumberLong(1),
        "syncingTo" : "",
        "syncSourceHost" : "",
        "syncSourceId" : -1,
        "heartbeatIntervalMillis" : NumberLong(2000),
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(1590387917, 1),
                        "t" : NumberLong(1)
                },
                "readConcernMajorityOpTime" : {
                        "ts" : Timestamp(1590387917, 1),
                        "t" : NumberLong(1)
                },
                "appliedOpTime" : {
                        "ts" : Timestamp(1590387917, 1),
                        "t" : NumberLong(1)
```

```
        },
        "durableOpTime" : {
                "ts" : Timestamp(1590387917, 1),
                "t" : NumberLong(1)
        }
},
"members" : [
        {
                "_id" : 0,
                "name" : "192.168.200.107:27003",
                "health" : 1,
                "state" : 1,
                "stateStr" : "PRIMARY",
                "uptime" : 43507,
                "optime" : {
                        "ts" : Timestamp(1590387917, 1),
                        "t" : NumberLong(1)
                },
                "optimeDate" : ISODate("2020-05-25T06:25:17Z"),
                "syncingTo" : "",
                "syncSourceHost" : "",
                "syncSourceId" : -1,
                "infoMessage" : "could not find member to sync from",
                "electionTime" : Timestamp(1590387916, 1),
                "electionDate" : ISODate("2020-05-25T06:25:16Z"),
                "configVersion" : 1,
                "self" : true,
                "lastHeartbeatMessage" : ""
        },
        {
                "_id" : 1,
                "name" : "192.168.200.108:27003",
                "health" : 1,
                "state" : 2,
                "stateStr" : "SECONDARY",
                "uptime" : 19,
                "optime" : {
```

```
                "ts" : Timestamp(1590387917, 1),
                "t" : NumberLong(1)
        },
        "optimeDurable" : {
                "ts" : Timestamp(1590387917, 1),
                "t" : NumberLong(1)
        },
        "optimeDate" : ISODate("2020-05-25T06:25:17Z"),
        "optimeDurableDate" : ISODate("2020-05-25T06:25:17Z"),
        "lastHeartbeat" : ISODate("2020-05-25T06:25:20.306Z"),
        "lastHeartbeatRecv" : ISODate("2020-05-25T06:25:20.304Z"),
        "pingMs" : NumberLong(11),
        "lastHeartbeatMessage" : "",
        "syncingTo" : "192.168.200.107:27003",
        "syncSourceHost" : "192.168.200.107:27003",
        "syncSourceId" : 0,
        "infoMessage" : "",
        "configVersion" : 1
},
{
        "_id" : 2,
        "name" : "192.168.200.109:27003",
        "health" : 1,
        "state" : 2,
        "stateStr" : "SECONDARY",
        "uptime" : 19,
        "optime" : {
                "ts" : Timestamp(1590387917, 1),
                "t" : NumberLong(1)
        },
        "optimeDurable" : {
                "ts" : Timestamp(1590387917, 1),
                "t" : NumberLong(1)
        },
        "optimeDate" : ISODate("2020-05-25T06:25:17Z"),
        "optimeDurableDate" : ISODate("2020-05-25T06:25:17Z"),
        "lastHeartbeat" : ISODate("2020-05-25T06:25:20.291Z"),
```

```
                        "lastHeartbeatRecv" : ISODate("2020-05-25T06:25:20.132Z"),
                        "pingMs" : NumberLong(8),
                        "lastHeartbeatMessage" : "",
                        "syncingTo" : "192.168.200.107:27003",
                        "syncSourceHost" : "192.168.200.107:27003",
                        "syncSourceId" : 0,
                        "infoMessage" : "",
                        "configVersion" : 1
                }
        ],
        "ok" : 1,
        "operationTime" : Timestamp(1590387917, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1590387917, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```


让3个分片各自形成1主2从的复制集，注意端口及仲裁节点的问题即可，操作完成后3个分片都启动完成，并完成复制集模式。

5.8、路由服务部署(3台服务器执行相同操作)

有些文档用命令去做，这是临时的，建议还是写到配置文件里面变成永久的

在/usr/local/mongodb/conf/目录下创建mongos.conf路由实例文件

```
[root@mongodb1 ~]# cat << END > /usr/local/mongodb/conf/mongos.conf
> logpath=/usr/local/mongodb/logs/mongos.log
> logappend=true
> port=27017
> fork=true
> configdb=configs/192.168.200.107:27018,192.168.200.108:27018,192.168.200.109:27018
> maxConns=20000
> bind_ip=0.0.0.0
> END
[root@mongodb1 ~]# scp /usr/local/mongodb/conf/mongos.conf
192.168.200.108:/usr/local/mongodb/conf/
root@192.168.200.108's password:
```

```
mongos.conf
100%  194    185.2KB/s   00:00
[root@mongodb1 ~]# scp /usr/local/mongodb/conf/mongos.conf
192.168.200.109:/usr/local/mongodb/conf/
root@192.168.200.109's password:
mongos.conf
100%  194     89.9KB/s   00:00
```

5.9 启动mongos

分别在三台服务器启动

```
[root@mongodb1 ~]# mongos -f /usr/local/mongodb/conf/mongos.conf
about to fork child process, waiting until server is ready for connections.
forked process: 5676
child process started successfully, parent exiting
[root@mongodb1 ~]# netstat -lnpt | grep -E "2700[1-3]"
tcp        0      0 0.0.0.0:27001           0.0.0.0:*               LISTEN
3265/mongod
tcp        0      0 0.0.0.0:27002           0.0.0.0:*               LISTEN
3295/mongod
tcp        0      0 0.0.0.0:27003           0.0.0.0:*               LISTEN
3319/mongod
[root@mongodb1 ~]# netstat -lnpt | grep mongo
tcp        0      0 0.0.0.0:27017           0.0.0.0:*               LISTEN
5676/mongos
tcp        0      0 0.0.0.0:27018           0.0.0.0:*               LISTEN
2433/mongod
tcp        0      0 0.0.0.0:27001           0.0.0.0:*               LISTEN
3265/mongod
tcp        0      0 0.0.0.0:27002           0.0.0.0:*               LISTEN
3295/mongod
tcp        0      0 0.0.0.0:27003           0.0.0.0:*               LISTEN
3319/mongod
```

5.10 启动分片功能

连接mongo

```
[root@mongodb1 ~]# mongo --host 192.168.200.107 --port 27017
```

切换数据库

```
mongos> use admin
switched to db admin
```

添加分片，只需在一台机器执行即可

```
mongos>
sh.addShard("shard1/192.168.200.107:27001,192.168.200.108:27001,192.168.200.109:27001")
{
        "shardAdded" : "shard1",
        "ok" : 1,
        "operationTime" : Timestamp(1590389582, 6),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1590389582, 6),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos>
sh.addShard("shard2/192.168.200.107:27002,192.168.200.108:27002,192.168.200.109:27002")
{
        "shardAdded" : "shard2",
        "ok" : 1,
        "operationTime" : Timestamp(1590389662, 2),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1590389662, 2),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos>
sh.addShard("shard3/192.168.200.107:27003,192.168.200.108:27003,192.168.200.109:27003")
{
        "shardAdded" : "shard3",
        "ok" : 1,
        "operationTime" : Timestamp(1590389707, 6),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1590389707, 6),
```

```
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("5ecab63652e09bfe1c9d04e7")
  }
  shards:
        {  "_id" : "shard1",  "host" :
"shard1/192.168.200.107:27001,192.168.200.108:27001,192.168.200.109:27001",  "state"
: 1 }
        {  "_id" : "shard2",  "host" :
"shard2/192.168.200.107:27002,192.168.200.108:27002,192.168.200.109:27002",  "state"
: 1 }
        {  "_id" : "shard3",  "host" :
"shard3/192.168.200.107:27003,192.168.200.108:27003,192.168.200.109:27003",  "state"
: 1 }
  active mongoses:
        "4.0.6" : 3
  autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
        {  "_id" : "config",  "primary" : "config",  "partitioned" : true }
                config.system.sessions
```

```
                        shard key: { "_id" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                shard1  1
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1
} } on : shard1 Timestamp(1, 0)
```

5.11 测试分片功能 --了解

设置分片chunk大小

```
mongos> use config
switched to db config
```

设置块大小为1M测试，不然需要插入海量数据

```
mongos> db.setting.save({"_id":"chunksize","value":1})
WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : "chunksize"
})
```

模拟写入数据

```
mongos> use cloud
switched to db cloud
```

模拟往cloud数据库的user表写入10000数据

```
mongos> for(i=1;i<=10000;i++){db.user.insert({"id":1,"name":"sofia,i love you"+i})}
WriteResult({ "nInserted" : 1 })
```

启用数据库分片

```
mongos> sh.enableSharding("cloud")
{
        "ok" : 1,
        "operationTime" : Timestamp(1590391195, 3),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1590391195, 3),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos> db.user.find()
{ "_id" : ObjectId("5ecb6dd4edfed44d8b1e8113"), "id" : 1, "name" : "sofia,i love
you1" }
```

省略一万行--------

5.12 创建索引，对表进行分片

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("5ecab63652e09bfe1c9d04e7")
  }
  shards:
        {  "_id" : "shard1",  "host" :
"shard1/192.168.200.107:27001,192.168.200.108:27001,192.168.200.109:27001",  "state"
: 1 }
        {  "_id" : "shard2",  "host" :
"shard2/192.168.200.107:27002,192.168.200.108:27002,192.168.200.109:27002",  "state"
: 1 }
        {  "_id" : "shard3",  "host" :
"shard3/192.168.200.107:27003,192.168.200.108:27003,192.168.200.109:27003",  "state"
: 1 }
  active mongoses:
        "4.0.6" : 3
  autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
        {  "_id" : "cloud",  "primary" : "shard2",  "partitioned" : true,  "version"
: {  "uuid" : UUID("72c9c1c3-4c78-42d5-8e17-67e4ec48d09a"),  "lastMod" : 1 } }
        {  "_id" : "config",  "primary" : "config",  "partitioned" : true }
                config.system.sessions
                        shard key: { "_id" : 1 }
                        unique: false
```

```
                        balancing: true
                        chunks:
                                shard1  1
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1
} } on : shard1 Timestamp(1, 0)
mongos> db.user.createIndex({"id":1})            # 以“id”作为索引
{
        "raw" : {

"shard2/192.168.200.107:27002,192.168.200.108:27002,192.168.200.109:27002" : {
                        "createdCollectionAutomatically" : false,
                        "numIndexesBefore" : 1,
                        "numIndexesAfter" : 2,
                        "ok" : 1
                }
        },
        "ok" : 1,
        "operationTime" : Timestamp(1590391464, 1),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1590391464, 1),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
mongos> sh.shardCollection("cloud.user",{"id":1})   # 根据“id”对user表进行分片
{
        "collectionsharded" : "cloud.user",
        "collectionUUID" : UUID("638aa41f-b0f4-421e-b72b-08d8639298cd"),
        "ok" : 1,
        "operationTime" : Timestamp(1590391569, 14),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1590391569, 14),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
```

```
                }
        }
}
mongos> sh.status()            # 查看分片情况
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("5ecab63652e09bfe1c9d04e7")
  }
  shards:
        {  "_id" : "shard1",  "host" :
"shard1/192.168.200.107:27001,192.168.200.108:27001,192.168.200.109:27001",  "state"
: 1 }
        {  "_id" : "shard2",  "host" :
"shard2/192.168.200.107:27002,192.168.200.108:27002,192.168.200.109:27002",  "state"
: 1 }
        {  "_id" : "shard3",  "host" :
"shard3/192.168.200.107:27003,192.168.200.108:27003,192.168.200.109:27003",  "state"
: 1 }
  active mongoses:
        "4.0.6" : 3
  autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
        {  "_id" : "cloud",  "primary" : "shard2",  "partitioned" : true,  "version"
: {  "uuid" : UUID("72c9c1c3-4c78-42d5-8e17-67e4ec48d09a"),  "lastMod" : 1 } }
                cloud.user
                        shard key: { "id" : 1 }
                        unique: false
```

```
                        balancing: true
                        chunks:
                                shard2  1
                        { "id" : { "$minKey" : 1 } } -->> { "id" : { "$maxKey" : 1 }
} on : shard2 Timestamp(1, 0)
        {  "_id" : "config",  "primary" : "config",  "partitioned" : true }
                config.system.sessions
                        shard key: { "_id" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                shard1  1
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1
} } on : shard1 Timestamp(1, 0)
```

到此。MongoDB分布式分片集群已经部署完毕

运维保证的是整个环境的稳定运行

这里数据库的具体的分片操作是DBA的事了

分片的时候最好使用索引来做分片，这样做的效果才会明显，因为查找数据靠的索引

小练习~

导入一组数据sales.txt

-d :数据库目录

-c：导入的集合为sales

-f：列

--file：指定文件

--type：格式

```
[root@mongodb1 ~]# mongoimport -d study -c sales -f id,num,pid,price --file sales.txt
--type=csv
2020-05-25T15:35:31.129+0800    connected to: localhost
2020-05-25T15:35:34.123+0800    [........................] study.sales  20.0KB/2.32MB
(0.8%)
2020-05-25T15:35:37.122+0800    [........................] study.sales  20.0KB/2.32MB
(0.8%)
2020-05-25T15:35:40.124+0800    [#.......................] study.sales  140KB/2.32MB
(5.9%)
2020-05-25T15:35:43.122+0800    [####....................] study.sales  408KB/2.32MB
(17.2%)
```

```
2020-05-25T15:35:46.121+0800    [#########.............] study.sales  896KB/2.32MB
(37.8%)
2020-05-25T15:35:49.121+0800    [###############.........] study.sales  1.43MB/2.32MB
(61.7%)
2020-05-25T15:35:52.190+0800    [####################....] study.sales  1.95MB/2.32MB
(83.9%)
2020-05-25T15:35:53.801+0800    [########################.] study.sales  2.32MB/2.32MB
(100.0%)
2020-05-25T15:35:53.801+0800    imported 121317 documents
[root@mongodb1 ~]# mongo --host 192.168.200.107 --port 27017
mongos> use study
switched to db study
mongos> db.sales.find().limit(3)
{ "_id" : ObjectId("5ecb754357ed495dc201d435"), "id" : 1, "num" : 1, "pid" : 776,
"price" : 2024.994 }
{ "_id" : ObjectId("5ecb754357ed495dc201d436"), "id" : 2, "num" : 3, "pid" : 777,
"price" : 2024.994 }
{ "_id" : ObjectId("5ecb754357ed495dc201d437"), "id" : 3, "num" : 1, "pid" : 778,
"price" : 2024.994 }
创建id列为索引
mongos> db.sales.createIndex({"id":1})
{
        "raw" : {

"shard3/192.168.200.107:27003,192.168.200.108:27003,192.168.200.109:27003" : {
                        "createdCollectionAutomatically" : false,
                        "numIndexesBefore" : 1,
                        "numIndexesAfter" : 2,
                        "ok" : 1
                }
        },
        "ok" : 1,
        "operationTime" : Timestamp(1590392371, 2),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1590392371, 2),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
```

```
                        "keyId" : NumberLong(0)
                }
        }
}
```

对sales表基于id列分片

```
mongos> sh.shardCollection("study.sales",{"id":1})
{
        "ok" : 0,
        "errmsg" : "sharding not enabled for db study",
        "code" : 20,
        "codeName" : "IllegalOperation",
        "operationTime" : Timestamp(1590392490, 2),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1590392490, 2),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

查看分片状态

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("5ecab63652e09bfe1c9d04e7")
  }
  shards:
        {  "_id" : "shard1",  "host" :
"shard1/192.168.200.107:27001,192.168.200.108:27001,192.168.200.109:27001",  "state"
: 1 }
        {  "_id" : "shard2",  "host" :
"shard2/192.168.200.107:27002,192.168.200.108:27002,192.168.200.109:27002",  "state"
: 1 }
```

```
        {  "_id" : "shard3",  "host" :
"shard3/192.168.200.107:27003,192.168.200.108:27003,192.168.200.109:27003",  "state"
: 1 }
  active mongoses:
        "4.0.6" : 3
  autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
        {  "_id" : "cloud",  "primary" : "shard2",  "partitioned" : true,  "version"
: {  "uuid" : UUID("72c9c1c3-4c78-42d5-8e17-67e4ec48d09a"),  "lastMod" : 1 } }
                cloud.user
                        shard key: { "id" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                shard2  1
                        { "id" : { "$minKey" : 1 } } -->> { "id" : { "$maxKey" : 1 }
} on : shard2 Timestamp(1, 0)
        {  "_id" : "config",  "primary" : "config",  "partitioned" : true }
                config.system.sessions
                        shard key: { "_id" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                shard1  1
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : { "$maxKey" : 1
} } on : shard1 Timestamp(1, 0)
        {  "_id" : "study",  "primary" : "shard3",  "partitioned" : false,  "version"
: {  "uuid" : UUID("aaac0d9f-d50c-4869-9a96-faec218a8b05"),  "lastMod" : 1 } }
mongos> db.sales.stats()
{
```

```
        "sharded" : false,
        "primary" : "shard3",
        "paddingFactorNote" : "paddingFactor is unused and unmaintained in 3.0. It
remains hard coded to 1.0 for compatibility only.",
        "userFlags" : 1,
        "capped" : false,
        "ns" : "study.sales",
        "count" : 121317,
        "numExtents" : 7,
        "size" : 13587504,
        "storageSize" : 22507520,
        "totalIndexSize" : 7023184,
        "indexSizes" : {
                "_id_" : 3949008,
                "id_1" : 3074176
        },
        "avgObjSize" : 112,
        "maxSize" : NumberLong(0),
        "nindexes" : 2,
        "nchunks" : 1,
        "shards" : {
                "shard3" : {
                        "ns" : "study.sales",
                        "size" : 13587504,
                        "count" : 121317,
                        "avgObjSize" : 112,
                        "numExtents" : 7,
                        "storageSize" : 22507520,
                        "lastExtentSize" : 11325440,
                        "paddingFactor" : 1,
                        "paddingFactorNote" : "paddingFactor is unused and
unmaintained in 3.0. It remains hard coded to 1.0 for compatibility only.",
                        "userFlags" : 1,
                        "capped" : false,
                        "nindexes" : 2,
                        "indexDetails" : {
```

```
                        },
                        "totalIndexSize" : 7023184,
                        "indexSizes" : {
                                "_id_" : 3949008,
                                "id_1" : 3074176
                        },
                        "ok" : 1,
                        "operationTime" : Timestamp(1590392638, 2),
                        "$gleStats" : {
                                "lastOpTime" : {
                                        "ts" : Timestamp(1590392371, 2),
                                        "t" : NumberLong(1)
                                },
                                "electionId" : ObjectId("7fffffff0000000000000001")
                        },
                        "lastCommittedOpTime" : Timestamp(1590392638, 2),
                        "$configServerState" : {
                                "opTime" : {
                                        "ts" : Timestamp(1590392638, 1),
                                        "t" : NumberLong(2)
                                }
                        },
                        "$clusterTime" : {
                                "clusterTime" : Timestamp(1590392638, 2),
                                "signature" : {
                                        "hash" :
BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                                        "keyId" : NumberLong(0)
                                }
                        }
                }
        },
        "ok" : 1,
        "operationTime" : Timestamp(1590392638, 2),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1590392638, 2),
                "signature" : {
```

```
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
                        "keyId" : NumberLong(0)
                }
        }
}
```

分片几乎是平均分配，选择分片依据的列时，尽量选择不重复的列，不要选择例如性别这种只有两个的。

连接配置实例

```
[root@mongodb3 ~]# mongo --host 192.168.200.109 --port 27018
 configs:PRIMARY>
configs:PRIMARY> show dbs
admin   0.000GB
config  0.001GB
local   0.000GB
configs:PRIMARY> use config
switched to db config
configs:PRIMARY> show collections
changelog
chunks
collections
databases
lockpings
locks
migrations
mongos
setting
shards
tags
transactions
version
configs:PRIMARY> db.chunks.find()       # 查看块的信息
{ "_id" : "config.system.sessions-_id_MinKey", "ns" : "config.system.sessions", "min" : { "_id" : { "$minKey" : 1 } }, "max" : { "_id" : { "$maxKey" : 1 } }, "shard" : "shard1", "lastmod" : Timestamp(1, 0), "last
modEpoch" : ObjectId("5ecb6ba32eefaee56f1f2b70"), "history" : [ { "validAfter" : Timestamp(1590389667, 2), "shard" : "shard1" } ] }{ "_id" : "cloud.user-id_MinKey", "ns" : "cloud.user", "min" : { "id" : { "$minKey" : 1 } }, "max" : { "id" : {
```

"$maxKey" : 1 } }, "shard" : "shard2", "lastmod" : Timestamp(1, 0), "lastmodEpoch" : ObjectId("5ecb7
311778010ceabbcf21b"), "history" : [{ "validAfter" : Timestamp(1590391569, 3),
"shard" : "shard2" }] }

分片管理：

添加及删除分片的方式：

configs:PRIMARY> sh.addShard("127.0.0.1:27004")

configs:PRIMARY> db.runCommand({"removeshard":"127.0.0.1:27004"})

到目前这里主要做的是分片集群的构建

具体怎么分片是DBA的事

六、利用NginxStream代理路由实例

前期呢，应用实例来连接的时候通过路由器有三个入口

```
[root@mongodb1 ~]# mongo --host 192.168.200.107 --port 27017
MongoDB shell version v4.0.6
connecting to: mongodb://192.168.200.107:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("86cb968f-2e16-46cf-9051-a4162c66a97b") }
MongoDB server version: 4.0.6
Server has startup warnings:
2020-05-25T14:36:06.033+0800 I CONTROL  [main]
2020-05-25T14:36:06.033+0800 I CONTROL  [main] ** WARNING: Access control is not
enabled for the database.
2020-05-25T14:36:06.033+0800 I CONTROL  [main] **          Read and write access to
data and configuration is unrestricted.
2020-05-25T14:36:06.033+0800 I CONTROL  [main] ** WARNING: You are running this
process as the root user, which is not recommended.
2020-05-25T14:36:06.033+0800 I CONTROL  [main]
mongos> exit
bye
[root@mongodb1 ~]# mongo --host 192.168.200.108 --port 27017
MongoDB shell version v4.0.6
connecting to: mongodb://192.168.200.108:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("e7f651d4-03a5-47df-b31e-3d34fe56f7d8") }
MongoDB server version: 4.0.6
Server has startup warnings:
2020-05-25T14:38:21.688+0800 I CONTROL  [main]
2020-05-25T14:38:21.688+0800 I CONTROL  [main] ** WARNING: Access control is not
enabled for the database.
```

```
2020-05-25T14:38:21.688+0800 I CONTROL  [main] **          Read and write access to data and configuration is unrestricted.
2020-05-25T14:38:21.688+0800 I CONTROL  [main] ** WARNING: You are running this process as the root user, which is not recommended.
2020-05-25T14:38:21.688+0800 I CONTROL  [main]
mongos> exit
bye
[root@mongodb1 ~]# mongo --host 192.168.200.109 --port 27017
MongoDB shell version v4.0.6
connecting to: mongodb://192.168.200.109:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("d062bfb2-301e-4770-8566-23f59f732c3d") }
MongoDB server version: 4.0.6
Server has startup warnings:
2020-05-25T14:38:42.837+0800 I CONTROL  [main]
2020-05-25T14:38:42.837+0800 I CONTROL  [main] ** WARNING: Access control is not enabled for the database.
2020-05-25T14:38:42.837+0800 I CONTROL  [main] **          Read and write access to data and configuration is unrestricted.
2020-05-25T14:38:42.837+0800 I CONTROL  [main] ** WARNING: You are running this process as the root user, which is not recommended.
2020-05-25T14:38:42.837+0800 I CONTROL  [main]
mongos> exit
bye
```

==========

nginx从1.9.0版本开始，新增了ngx_stream_core_module 模块，使nginx支持四层负载均衡（不是一个web端口，只是一个TCP的协议）。默认编译的时候该模块并未编译进去，需要编译的时候添加--with-stream，使其支持stream代理。

现在很多公司用的都是nginx-1.14或nginx-1.15的居多

源码包安装nginx

```
[root@mongodb1 ~]# yum clean all && yum makecache
[root@mongodb1 ~]# useradd -M -s /sbin/nologin nginx
[root@mongodb1 nginx-1.16.0]# ./configure --prefix=/usr/local/nginx --user=nginx --group=nginx --with-stream && make && make install
```

stream段的配置要与http段在同级目录

```
[root@mongodb1 ~]# vim /usr/local/nginx/conf/nginx.conf
[root@mongodb1 ~]# ln -s /usr/local/nginx/conf/nginx.conf /etc/nginx.conf      # 软链接
```

```
[root@mongodb1 ~]# vim /etc/nginx.conf
stream {
    upstream mongodb {
        server 192.168.200.107:27017 weight=1 max_fails=3 fail_timeout=10s;
        server 192.168.200.108:27017 weight=1 max_fails=3 fail_timeout=10s;
        server 192.168.200.109:27017 weight=1 max_fails=3 fail_timeout=10s;
    }
    server {
        listen 17017;
        proxy_responses 1;
        proxy_timeout 20s;
        proxy_pass mongodb;
    }
}
[root@mongodb1 ~]# /usr/local/nginx/sbin/nginx -t        # 检查语法
nginx: the configuration file /usr/local/nginx/conf/nginx.conf syntax is ok
nginx: configuration file /usr/local/nginx/conf/nginx.conf test is successful
[root@mongodb1 ~]# /usr/local/nginx/sbin/nginx         # 启动生效
[root@mongodb1 ~]# netstat -anpt | grep "17017"
tcp        0      0 0.0.0.0:17017           0.0.0.0:*               LISTEN
10993/nginx: master
[root@mongodb1 ~]# netstat -anpt | grep nginx
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
10993/nginx: master
tcp        0      0 0.0.0.0:17017           0.0.0.0:*               LISTEN
10993/nginx: master
```

连接测试

```
[root@mongodb1 ~]# mongo --host 192.168.200.107 --port 17017
```

在设置日志文件大小的时候，最好设置能把一天的量都装进去

自动化运维工具之Saltstack

类似于ansible

```
[root@salt-master ~]#
[root@salt-minion1 ~]#
[root@salt-minion2 ~]#
```

1. SALTSTACK 的安装

简介

SaltStack (http://www.saltstack.com/)
是一个服务器基础架构集中化管理平台（实现自动化的），
具备配置管理（类似ansible中的playbook）、远程执行、监控等功能，
一般可以理解为简化版的puppet (ttp:/ppetlabs.com/)和加强版的func (https://fedorahosted.org/func/) 和Ansible。

---

puppet最早实现自动化类的一个平台，但是不易上手，现在公司常见的就是saltstack和ansible
ansible更适合于中小型公司的环境
而至于大公司的话，基于机器数量规模特别大，建议使用saltstack

SaltStack及ansible基于Python语言实现，
结合轻量级消息队列(ZeroMQ)与Python第三方模块(Pyzmq、PyCrypto、Pyinjia2、 python-msgpack和PyYAML等)构建。
有如下特性:

- 部署简单、方便;
- 支持大部分UNIX/Linux及Windows环境;
- 主从集中化管理;---可实现主从的集中化管理
- 配置简单、功能强大、扩展性强;--因为拿Python开发的
- 主控端(master) 和被控端(minion) 基于证书认证，安全可靠;
  C/S结构
  C--> client minion agent
  S--> server master
- 支持API及自定义模块，可通过Python轻松扩展。

通过部署SaltStack 环境，我们可以在成千上万台服务器上做到批量执行命令，根据不同业务特性进行配置集中化管理、分发文件、采集服务器数据、操作系统基础及软件包管理等，纲ltStack是运维人员提高工作效率、规范业务配置与操作的利器。目前Saltstack .已经趋向成熟，用户群及社区活跃度都不错，同时官方也开放了不少子项目，具体可访问https://github.com/saltstack获得。

官方文档: http://docs.saltstack.com ----特别牛！写的特别细，就是找的时候不好找
对于很多的产品而言，最好从官方文档趴知识点，在百度上找的都是二手资料
如果想从技术上做提升的话，就要去看官方文档了

中国SaltStack用户组: http://www.saltstack.cn

1.Saltstack的安装

Saltstack的不同角色服务安装非常简单，建议采用yum源方式来实现部署，下面简单介绍具体步骤。

1.1业务环境说明

通过部署两组业务功能服务器来进行演示，相关服务器信息如表。

角色	ID（minion id）	IP	GroupName(组名)
master	master.salt.com	192.168.200.107	
minion	node1.salt.com	192.168.200.108	web1group
minion	node2.salt.com	192.168.200.109	web2group

1.2安装EPEL

由于目前RHEL官网yum源还没有Saltstack的软件包支持，因此先安装EPEL作为部署Saltstack的默认yum源。

```
[root@salt-master ~]# yum clean all && yum makecache fast
[root@salt-master ~]# iptables -F
[root@salt-master ~]# setenforce 0
setenforce: SELinux is disabled
[root@salt-master ~]# systemctl stop firewalld
```

1.3安装SALTSTACK

注意：不同的机器要装不同的软件

saltstack之所以用的不多，是因为它的被管理端需要部署软件（虽然改个配置文件加个两三行，启动一下服务就完事，但因为这几步就让很多公司不来选它），所以它在中小型公司得不到推广；而它在大公司中比较看中的原因是，一是saltstack火的比较早，二是因为它具有消息队列，所以它更适合于大规模的环境。

1)主服务器安装(主控制端)

```
[root@salt-master ~]# yum -y install salt-master
```

2)从服务器安装(被控制端)

```
[root@salt-minion1 yum.repos.d]# yum -y install salt-minion
[root@salt-minion2 yum.repos.d]# yum -y install salt-minion
```

1.4 SALTSTACK防火墙配置

--如果将防火墙禁了的话，就不用考虑下面的配置了

SaltStack master启动后默认监听4505和4506两个端口。

4505（publish_port）为saltstack的消息发布系统。

saltstack在接收管理员指令的时候呢会把指令进行对外发布

4506（ret_port）为saltstack被控端与服务端通信的端口。

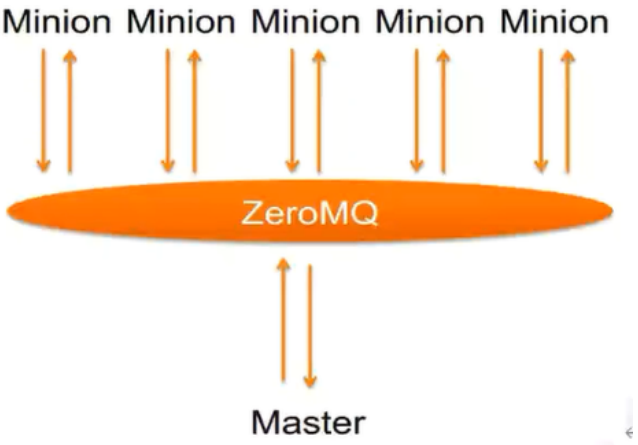
master和minion通信的端口

在主控制端添加TCP 4505，TCP 4506的规则，而在被控端无须配置防火墙，原理是被控端直接与主控端的zeromq建立长连接，接受广播到的任务信息并执行，具体操作市添加两

条iptables规则:

```
[root@salt-master ~]# iptables -I INPUT -m state new -m tcp -p tcp --dport 4505 -j ACCEPT
[root@salt-master ~]# iptables -I INPUT -m state new -m tcp -p tcp --dport 4506 -j ACCEPT
```

如果使用Isof查看4505端口，会发现所有的minion在4505端口持续保持在ESTABLISHED状态。minion 与master之间的通信模式如下:



更适合于高并发

1.5更新SALTSTACK配置及安装校验

SaltStack分两种角色，一种为master（主控端），另一种为minion（被控端），安装完毕后要对两种角色的配置文件进行修改。

1.5.1 master主控端配置

```
[root@salt-master ~]# vim /etc/salt/master
15  interface: 192.168.200.107        # 绑定master通信IP
215 auto_accept: True     #自动认证，避免手动运行salt-key来确认证书信任
406  file_roots:                      #指定Saltstack文件根目录位置；这三行顶格
407    base:
408      - /srv/salt/      # 存放资源的目录，类似于ansible中的roles，之后做的配置管理时编辑好的文件放在这里
[root@salt-master ~]# systemctl start salt-master            # 启动服务
```

1.5.2 minion被控端配置

```
[root@salt-minion1 ~]# vim /etc/salt/minion
 16 master: 192.168.200.107       # 指定master
 78 id: node1.salt.com            # 修改被控端主机识别id，建议使用操作系统知：wq:已名来配置
[root@salt-minion1 ~]# hostname node1.salt.com
[root@salt-minion1 ~]# bash
[root@node1 ~]# systemctl start salt-minion
```

```
[root@node2 ~]# vim /etc/salt/minion
 16 master: 192.168.200.107
 78 id: node2.salt.com
[root@node2 ~]# systemctl start salt-minion
[root@salt-master ~]# hostname master.salt.com
[root@master ~]# vim /etc/hosts        # 加快执行远程命令
192.168.200.107 master.salt.com
192.168.200.108 node1.salt.com
192.168.200.109 node2.salt.com
[root@master ~]# scp /etc/hosts 192.168.200.108:/etc/
root@192.168.200.108's password:
hosts                                                                    100%  252
21.0KB/s   00:00
[root@master ~]# scp /etc/hosts 192.168.200.109:/etc/
root@192.168.200.109's password:
hosts                                                                    100%  252
17.9KB/s   00:00
[root@master ~]# netstat -anpt | grep -E "4505|4506"
# 这里只出现出现两行的原因可能是：
# 1. 406-408那三行没顶格写
# 2. 主机名没改全，好像bash改完名之前的操作没了，需要重新检查一遍
# 3. 以后先改好主机名再修改配置文件
          tcp        0      0 192.168.200.107:4505    0.0.0.0:*               LISTEN
3786/python
          tcp        0      0 192.168.200.107:4506    0.0.0.0:*               LISTEN
3804/python
          tcp        0      0 192.168.200.107:4505    192.168.200.109:53086
ESTABLISHED 3786/python
          tcp        0      0 192.168.200.107:4506    192.168.200.109:57290
ESTABLISHED 3804/python
          tcp        0      0 192.168.200.107:4505    192.168.200.108:42062
ESTABLISHED 3786/python
          tcp        0      0 192.168.200.107:4506    192.168.200.108:52806
ESTABLISHED 3804/python
[root@master ~]# salt-key -L
Accepted Keys:
node1.salt.com
```

```
node2.salt.com
Denied Keys:
Unaccepted Keys:  # 导致这里有记录的原因是之前没有在主配置文件里面顶格写的缘故吧
node1.salt.com
node2.salt.com
Rejected Keys:
```

1.5.3 校验安装结果

通过test模块的ping方法，可以确认指定被控端设备与主控端是否建立信任关系及连通性是否正常，探测所有被控端采用'*'来代替' node1.salt.com'即可

```
[root@master ~]#  salt '*' test.ping
node1.salt.com:
    True
node2.salt.com:
    True
[root@master ~]#  salt 'node1.salt.com' test.ping
node1.salt.com:
    True
```

格式：

```
salt 被控制端 模块.方法 传参
```

1.6提示

当/etc/salt/master没有配置auto accept: True时，

管理端需要通过salt-key 命令来进行证书认证操作，具体操作如下:

```
salt-key -L   显示已经或未认证的被控端id，
Accept Keys 为已认证清单，
Unaccepted Keys为未认证清单;
salt-key -D      删除 所有认证主机id证书
salt-key -d id          删除单个id证书
salt-key -A      接受所有 id证书请求
salt-key -a id   接受单个id证书请求
```

演示一下哈~

```
[root@master ~]# salt-key -L
Accepted Keys:
node1.salt.com
node2.salt.com
```

```
Denied Keys:
Unaccepted Keys:
Rejected Keys:
[root@master ~]# salt-key -d node2.salt.com
The following keys are going to be deleted:
Accepted Keys:
node2.salt.com
Proceed? [N/y] y
Key for minion node2.salt.com deleted
[root@master ~]# vim /etc/salt/master
215 auto_accept: False
[root@master ~]# systemctl restart salt-master.service
[root@master ~]# salt-key -L
[root@master ~]# salt-key -L
Accepted Keys:
node1.salt.com
Denied Keys:
Unaccepted Keys:
node1.salt.com
node2.salt.com
Rejected Keys:
[root@master ~]# salt-key -a node1.salt.com,node2.salt.com
The following keys are going to be accepted:
Unaccepted Keys:
node1.salt.com
node2.salt.com
Proceed? [n/Y] y
Key for minion node2.salt.com accepted.
```

1.7认证流程(密钥)

master/minion数据传输采用AES加密算法

salt支持自动认证---更加安全稳定

在master端：

```
[root@master ~]# ll /etc/salt/pki/master/
总用量 8
-r-------- 1 root root 1675 6月   3 16:51 master.pem       # 私钥
-rw-r--r-- 1 root root  451 6月   3 16:51 master.pub        # 公钥
drwxr-xr-x 2 root root   50 6月   3 17:34 minions
```

```
drwxr-xr-x 2 root root    6 6月   3 16:51 minions_autosign
drwxr-xr-x 2 root root    6 6月   3 16:51 minions_denied
drwxr-xr-x 2 root root    6 6月   3 17:34 minions_pre
drwxr-xr-x 2 root root    6 6月   3 16:51 minions_rejected
[root@master ~]# ll /etc/salt/pki/master/minions
总用量 8
-rw-r--r-- 1 root root 451 6月   3 16:56 node1.salt.com
-rw-r--r-- 1 root root 451 6月   3 17:33 node2.salt.com
```

在minions端：

```
[root@node1 ~]# ll /etc/salt/pki/minion/
总用量 12
-rw-r--r-- 1 root root  451 6月   3 09:11 minion_master.pub
-r-------- 1 root root 1675 6月   3 08:56 minion.pem
-rw-r--r-- 1 root root  451 6月   3 08:56 minion.pub
```

把公钥发给master

```
[root@master ~]# cd /etc/salt/pki/master/
[root@master master]# tree
.
├── master.pem
├── master.pub
├── minions
│   ├── node1.salt.com
│   └── node2.salt.com
├── minions_autosign
├── minions_denied
├── minions_pre
└── minions_rejected
5 directories, 4 files
[root@master master]# netstat -anpt | grep python
tcp        0      0 192.168.200.107:4505    0.0.0.0:*               LISTEN
5020/python
tcp        0      0 192.168.200.107:4506    0.0.0.0:*               LISTEN
5032/python
tcp        0      0 192.168.200.107:4506    192.168.200.108:52818   ESTABLISHED
5032/python
tcp        0      0 192.168.200.107:4505    192.168.200.108:42074   ESTABLISHED
5020/python
```

```
tcp        0      0 192.168.200.107:4506    192.168.200.109:57320   ESTABLISHED
5032/python
tcp        0      0 192.168.200.107:4505    192.168.200.109:53116   ESTABLISHED
5020/python
[root@node1 ~]# netstat -anpt | grep python
tcp        0      0 192.168.200.108:42074   192.168.200.107:4505    ESTABLISHED
9095/python
tcp        0      0 192.168.200.108:52818   192.168.200.107:4506    ESTABLISHED
9095/python
```

2.利用SALTSTACK远程执行命令

saltstack的一个比较突出优势就是具备执行远程命令的功能。

操作方法与func(https:/fedorahosted.org/func/)相似，

可以帮助运维人员完成集中化的操作平台。

官方文档：http://docs.saltstack.cn/topics/targeting/index.html

命令格式: salt '<操作目标>' <模块.方法> [参数]

示例:查看被控制机的内存使用情况

```
[root@master master]# salt '*' cmd.run 'free -m'              （cmd.run这个要记住）
node1.salt.com:
                total        used        free      shared  buff/cache   available
    Mem:            974         220         213           7         540         540
    Swap:          3967           0        3967
node2.salt.com:
                total        used        free      shared  buff/cache   available
    Mem:            974         214         227           7         533         547
    Swap:          3967           0        3967
```

其中针对<操作目标>，saltstack 提供了多种方法对被控制端(id) 进行过滤。下面列举常用的具体参数:。

1) -E: --pcre

通过正则表达式进行匹配。

示例:查看被控制端node1字符开头的主机id名是否连通。

```
[root@master master]# salt -E '^node1.*' test.ping
node1.salt.com:
    True
[root@master master]# salt -E '^node.*' test.ping
node1.salt.com:
    True
node2.salt.com:
```

```
    True
```

2) -L: --list

以主机id名列表的形式进行过滤，格式与Python的列表相似，即不同主机id名称使用逗号分隔

示例:获取主机id名为node1.salt.com,node2.salt.com;获取完整操作系统发行版名称。 θ

```
[root@master master]# salt -L 'node1.salt.com,node2.salt.com' test.ping
node1.salt.com:
    True
node2.salt.com:
    True
```

3) -G: --grain

根据 被控主机的grains信息进行匹配过滤

(grains是saltstack重要组件之一，重要作用是收集被控主机的基本系统信息)，

格式为'<grain value>:<glob expression>'。

比如过滤内核为Linux的主机可以写成'kernel:Linux'，

如果同时需要正则表达式的支持可以切成--grain-pcre.参数来执行。

示例:获取主机发行版本为7.5.1804的Python版本

```
[root@master master]# cat /etc/redhat-release
CentOS Linux release 7.5.1804 (Core)
[root@master master]# salt -G 'osrelease:7.5.1804' test.ping
node2.salt.com:
    True
node1.salt.com:
    True
[root@master master]# salt -G 'osrelease:7.5.1804' cmd.run 'python -V'
node2.salt.com:
    Python 2.7.5
node1.salt.com:
    Python 2.7.5
[root@master master]# salt -G 'kernel:Linux' cmd.run 'python -V'
node2.salt.com:
    Python 2.7.5
node1.salt.com:
    Python 2.7.5
```

4) -I: --pillar

根据被控主机的pillar (作用是定义与被控主机相关的任何数据，定义好的数据可以被其他组件使用)信息进行过滤匹配，格式为'对象名称:对象值'，比如过滤所有具备'apache:httpd' pllr值的主机。

示例:探测具有"nginx:root:/data"信息的主机连通性

```
[root@master master]# salt -l 'nginx:root:/data' test.ping
node2.salt.com:
    True
node1.salt.com:
    True
```

其中pillar属性配置文件如下:

```
nginx:
    root: /data
```

5) -N: --nodegroup

根据主控端master配置文件中的分组名称进行过滤。

如下配置的组信息(主机信息支持正则表达式、grain、 条件运算符等),

通常根据业务类型划分，不同业务具备相同的特点，包括部署环境、应用平台、配置文件等。

L@　　表示后面的主机id格式为列表，即主机id以逗号隔开;

G@　　表示以grain格式描述;

S@　　表示以ip子网或地址格式描述

示例:探测web1group (或web2group)被控主机的连通性

```
[root@master master]# vim /etc/salt/master
713 nodegroups:
714   web1group: 'node1.salt.com'
715   web2group: 'L@node1.salt.com,node2.salt.com'          #  -L: --list
[root@master master]# salt -N web1group cmd.run 'python -V'
node1.salt.com:
    Python 2.7.5
[root@master master]# salt -N web2group cmd.run 'python -V'
node1.salt.com:
    Python 2.7.5
node2.salt.com:
    Python 2.7.5
```

6) -C: --compound

根据条件运算符not、and、 or 去匹配不同规则的主机信息。

示例:探测node1开头并且操作系统为Centos的主机连通性。

```
[root@master master]# salt -C '* and E@node1.*' test.ping
node1.salt.com:
    True
```

not语句不能作为第一个条件执行，不过可以通过以下方法来规避:

示例:探测非node1开头的主机连通性

```
[root@master master]# salt -C '* and not E@node1.*' test.ping
node2.salt.com:
    True
```

7) -S: -ipcidr

根据被控主机的ip地址或ip子网进行匹配。

示例:

```
[root@master master]# salt -S 192.168.200.108 test.ping
node1.salt.com:
    True
[root@master master]# salt -S 192.168.200.109 test.ping
node2.salt.com:
    True
[root@master master]# salt -S 192.168.200.0/24 test.ping
node1.salt.com:
    True
node2.salt.com:
    True
[root@master master]# salt -S 192.168.0.0/16 test.ping
node1.salt.com:
    True
node2.salt.com:
    True
```

3. SALTSTACK 常用模块及API

Saltstack提供了非常丰富的功能模块，涉及操作系统的基础功能、常用工具支持等，

更多模块信息请见:

官方：

https://docs.saltstack.com/en/latest/ref/modules/a/index.html

当然，也可以通过sys模块列出当前版本支持的所有模块:

中国用户做的，国内做的一个网站：翻译了一部分

http://docs.saltstack.cn/ref/modules/al/index.html

```
[root@master master]# salt 'node1.salt.com' sys.list_modules
node1.salt.com:
    - acl
    - aliases
    - alternatives
    - archive
```

```
    - artifactory
    - at
    - augeas
    - blockdev
    - bridge
    - btrfs
----------------------------省略n个
```

接下来抽取出常见的模块进行介绍,并列举模块API的用法。

API原理:通过调用masterclient模块，

实例化一个LocalClient对象，再调用cmd()方法来实现的。

如下是API实现test.ping的示例:

```
import salt.client
client = salt.client.LocalClient()
ret = client.cmd('*,'test.ping")
print ret
```

结果以一个标准的python字典形式的字符串返回吗，可以通过eval()函 数转换成python的字典类型，方便后续的业务逻辑处理，程序运行结果如下:

```
[root@master master]# python
Python 2.7.5 (default, Apr 11 2018, 07:36:10)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import salt.client
>>> client = salt.client.LocalClient()
>>> ret = client.cmd('*','test.ping')
>>> print ret
{'node1.salt.com': True, 'node2.salt.com': True}
>>> exit()
```

注意:

将字符字典转换成python的字典类型，推荐使用ast模块的literal_eval()方法， 可以过滤表达式中的恶意函数。

1）Archive模块

功能: 实现系统层面的压缩包调用，支持gunzip、gzip、 rar、tar、 unrar、 unzip 等

示例1:采用guzip.解压被控制机的/root/sofia.gz包

```
[root@node1 ~]# touch sofia
[root@node1 ~]# gzip sofia
[root@node1 ~]# ls
sofia.gz
```

```
[root@master master]# salt 'node1.salt.com' archive.gunzip /root/sofia.gz   # j解压
node1.salt.com:
[root@node1 ~]# ls
 sofia
[root@master master]# salt 'node1.salt.com' archive.gzip /root/sofia         # 压缩
node1.salt.com:
[root@node1 ~]# ls
sofia.gz
```

tar打包：

```
[root@master master]# salt 'node1.salt.com' archive.tar zcf /tmp/test.tar.gz
/root/sofia
node1.salt.com:
    - tar: Removing leading `/' from member names
[root@node1 ~]# ls /tmp/
test.tar.gz
```

API调用：

```
[root@master master]# python
Python 2.7.5 (default, Apr 11 2018, 07:36:10)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import salt.client
>>> client = salt.client.LocalClient()
>>> ret = client.cmd('node1.salt.com','archive.gzip',['/root/sofia'])
>>> print ret
{'node1.salt.com': []}
>>> ret = client.cmd('node1.salt.com','archive.gunzip',['/root/sofia.gz'])
>>> print ret
{'node1.salt.com': []}
```

2) cmd模块 ---命令、脚本 ----必须会!

功能:实现远程的命令行执行(默认具备root操作权限，使用时需评估风险)。

示例:获取所有主机内存情况

```
[root@master master]# salt 'node1.salt.com' cmd.run "df -Th"
node1.salt.com:
    Filesystem              Type      Size  Used Avail Use% Mounted on
    /dev/mapper/centos-root xfs        50G  4.5G   46G   9% /
    devtmpfs                devtmpfs  471M     0  471M   0% /dev
    tmpfs                   tmpfs     488M   12K  488M   1% /dev/shm
```

```
    tmpfs                   tmpfs     488M  7.9M  480M   2% /run
    tmpfs                   tmpfs     488M     0  488M   0% /sys/fs/cgroup
    /dev/sr0                iso9660   4.2G  4.2G     0 100% /media/cdrom
    /dev/mapper/centos-home xfs        46G   33M   46G   1% /home
    /dev/sda1               xfs      1014M  157M  858M  16% /boot
    tmpfs                   tmpfs      98M     0   98M   0% /run/user/0
[root@master master]# salt 'node1.salt.com' cmd.run "df -Th" | wc -l
11
----------------------------------------
[root@master master]# vim /etc/salt/master
file_roots:
  base:
    - /srv/salt/
[root@master master]# mkdir /srv/salt/script -p            # 放脚本
 [root@master master]# vim /srv/salt/script/test.sh
#!/bin/bash
mkdir /tmp/1111
[root@master master]# salt '*' cmd.script salt://script/test.sh
node1.salt.com:
    ----------
    pid:
        11554
    retcode:
        0
    stderr:
    stdout:
node2.salt.com:
    ----------
    pid:
        5579
    retcode:
        0
    stderr:
    stdout:
[root@node1 ~]# ls /tmp
1111
[root@node2 ~]# ls /tmp
```

1111

该命令会做两个动作:首先同步test.sh 到minion的cache目录(/var/cache/salt/minion/files/base/script/test.sh;其次运行该脚本---它的目录存放的很深这个脚本先传过去，再执行

```
[root@node1 ~]# cat /var/cache/salt/minion/files/base/script/test.sh
#!/bin/bash
mkdir /tmp/1111
```

API调用原理:

```
[root@master master]# python
Python 2.7.5 (default, Apr 11 2018, 07:36:10)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import salt.client                                    # 好像每次都要输入前几行
>>> client = salt.client.LocalClient()
>>> ret = client.cmd('node1.salt.com','cmd.run',['free -m'])
>>> print ret
{'node1.salt.com': '              total        used        free      shared
buff/cache   available\nMem:            974
         220         205           7         548         540\nSwap:          3967
0        3967'}
```

3) cp模块

功能:实现远程文件、目录复制、以及下载URL文件等操作

示例:

①将指定minion的/etc/hosts 文件复制到minion主机的本地的salt cache 目录（没什么用）(/var/cache/salt/minion/localfiles/)

```
[root@master master]# salt '*' cp.cache_local_file /etc/hosts
node1.salt.com:
    /var/cache/salt/minion/localfiles/etc/hosts
node2.salt.com:
    /var/cache/salt/minion/localfiles/etc/hosts
```

②将master的file_roots指定位置下的目录复制到minion上（有点用）

```
[root@master master]# salt '*' cp.get_dir salt://script /tmp
node1.salt.com:
    - /tmp/script/test.sh
node2.salt.com:
    - /tmp/script/test.sh
[root@node1 ~]# ls /tmp/script/
```

test.sh

[root@node2 ~]# ls /tmp/script/

test.sh

③cp.get_url 可以从一个URL地址下载文件，URL可以是msater上的路径(salt://)， 也可以是http网址。（还可以）

[root@master master]# salt '*' cp.get_url http://mirrors.aliyun.com/repo/Centos-7.repo /tmp/aliyun.repo

node1.salt.com:

    /tmp/aliyun.repo

node2.salt.com:

    /tmp/aliyun.repo

[root@node1 ~]# cat /tmp/aliyun.repo

[root@node2 ~]# cat /tmp/aliyun.repo

可以统一去给配阿里云的源

4）cron 模块 ---最鸡肋（格式很麻烦）

功能:实现minion的crontab操作

示例:

查看指定minion、root 用户的crontab清单

[root@master ~]# salt '*' cron.raw_cron root

node2.salt.com:

    * */5 * * * /usr/sbin/ntpdate pool.ntp.org > /dev/null 2>&1

node1.salt.com:

    * */5 * * * /usr/sbin/ntpdate pool.ntp.org > /dev/null 2>&1

为指定minion、root 用户添加作业任务

这里导致没有添加成功的原因是minions的crontab中的第一个计划任务串行了！

[root@master ~]# salt "*" cron.set_job root '*0' '*' '*' '*' '6' '/usr/bin/yum -y update'

node1.salt.com:

    new

node2.salt.com:

    new

[root@master ~]# salt '*' cron.set_job root '0' '0' '*' '*' '*' 'systemctl restart httpd'

node2.salt.com:

    new

node1.salt.com:

    new

删除minion、root用户的crontab的'/usr/bin/yum -y update'任务作业

```
[root@master ~]# salt '*' cron.rm_job root '/usr/bin/yum -y update'
node2.salt.com:
    removed
node1.salt.com:
    removed
[root@master ~]# salt '*' cron.raw_cron root
node2.salt.com:
    * */5 * * * /usr/sbin/ntpdate pool.ntp.org > /dev/null 2>&1
    # Lines below here are managed by Salt, do not edit
    0 0 * * * systemctl restart httpd
node1.salt.com:
    * */5 * * * /usr/sbin/ntpdate pool.ntp.org > /dev/null 2>&1
    # Lines below here are managed by Salt, do not edit
    0 0 * * * systemctl restart httpd
```

注意：批量类的管理工具在管理多台机器的时候是很好的，但是一定不要做误操作，

当然避免误操作的方法是做好异地备份

5) dnsutil 模块 --管理host文件的

功能:实现minion主机通用DNS相关操作

示例:

添加hosts主机配置项

```
[root@master ~]# salt '*' dnsutil.hosts_append /etc/hosts 127.0.0.1 www.sofia.com
node1.salt.com:
    The following line was added to /etc/hosts:
    127.0.0.1 www.sofia.com
node2.salt.com:
    The following line was added to /etc/hosts:
    127.0.0.1 www.sofia.com
[root@node1 ~]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.200.107 master.salt.com
192.168.200.108 node1.salt.com
192.168.200.109 node2.salt.com
127.0.0.1 www.sofia.com
```

删除hosts主机配置项

```
[root@master ~]# salt '*' dnsutil.hosts_remove /etc/hosts www.sofia.com
```

```
node2.salt.com:
    None
node1.salt.com:
    None
```

API调用原理

```
[root@master ~]# python
Python 2.7.5 (default, Apr 11 2018, 07:36:10)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import salt.client
>>> client = salt.client.LocalClient()
>>> ret = client.cmd('*','dnsutil.hosts_append',
['/etc/hosts','127.0.0.1','www.sofia.com'])
>>> print ret
{'node1.salt.com': 'The following line was added to /etc/hosts:\n127.0.0.1
www.sofia.com', 'node2.salt.com': 'The follow
ing line was added to /etc/hosts:\n127.0.0.1 www.sofia.com'}
>>> ret = client.cmd('*','dnsutil.hosts_remove',['/etc/hosts','www.sofia.com'])
>>> print ret
{'node1.salt.com': None, 'node2.salt.com': None}
```

6) file模块 ---功能非常强大

功能:实现minion主机文件常见操作，包括文件读写，权限，查找，校验等

示例:

1.检测文件MD5

```
[root@node1 ~]# md5sum /etc/passwd          # 获取md5值
407b8e3cdd2d3a988861dd93a16aa69a  /etc/passwd
```

校验所有minion主机文件的加密信息，支持md5、sha1、 sha224、 sha256、 sha384、sha512

2.加密算法

```
[root@master ~]# salt '*' file.get_sum /etc/passwd
node2.salt.com:
    d48376a5d48b7f07cfad16a38c3b7f8592109556b93b960970202512f2c78574
node1.salt.com:
    d48376a5d48b7f07cfad16a38c3b7f8592109556b93b960970202512f2c78574
```

3.修改文件所属用户

```
[root@master ~]# salt '*' file.chown /etc/passwd root root
node2.salt.com:
    None
```

```
node1.salt.com:
    None
```

4.复制所有minion的/path/to/src文件到/path/to/dst

```
[root@master ~]# salt '*' file.copy /etc/passwd /tmp/passwd.bak
node2.salt.com:
    True
node1.salt.com:
    True
```

5.检测所有minion的/etc目录是否存在，检测文件使用filefile _exists

```
[root@master ~]# salt '*' file.directory_exists /etc
node2.salt.com:
    True
node1.salt.com:
    True
```

6.获取所有minion的stats信息

```
[root@master ~]# salt '*' file.stats /etc/passwd
node1.salt.com:
    ----------
    atime:
        1591164001.29
    ctime:
        1591163913.97
    gid:
        0
    group:
        root
    inode:
        69545931
    mode:
        0644
    mtime:
        1591157976.96
    size:
        2349
    target:
        /etc/passwd
    type:
```

```
        file
    uid:
        0
    user:
        root
node2.salt.com:
    ----------
    atime:
        1591164001.76
    ctime:
        1591163913.97
    gid:
        0
    group:
        root
    inode:
        67195285
    mode:
        0644
    mtime:
        1586978834.43
    size:
        2349
    target:
        /etc/passwd
    type:
        file
    uid:
        0
    user:
        root
```

7.获取所有minion的/etc/passwd的权限mode

```
[root@master ~]# salt '*' file.get_mode /etc/passwd
node2.salt.com:
    0644
node1.salt.com:
    0644
```

8.修改所有minion的/etc/passwd的权限mode为0644

```
[root@master ~]# salt '*' file.set_mode /etc/passwd 0644
node2.salt.com:
    0644
node1.salt.com:
    0644
```

9.在所有minion上创建目录/opt/test

```
[root@master ~]# salt '*' file.mkdir /opt/test/
node2.salt.com:
    None
node1.salt.com:
    None
```

10.将所有minion主机上的httpd.conf文件的LogLevel warn改成info

```
[root@master ~]# salt '*' file.sed /etc/httpd/httpd.conf 'LogLevel warn' 'LogLevel
info'
node1.salt.com:
    False
node2.salt.com:
    False
```

11.将所有minion的/tmp/test/test.conf文件后面追加maxclient 100

```
[root@master ~]# salt '*' file.append /tmp/test/test.conf "maxclient 100"
node1.salt.com:
    Wrote 1 lines to "/tmp/test/test.conf"
node2.salt.com:
    Wrote 1 lines to "/tmp/test/test.conf"
```

12.删除所有minion的/tmp/foo文件

```
[root@master tmp]# salt '*' file.remove /tmp/foo
node1.salt.com:
    True
node2.salt.com:
    True
```

API调用原理:

```
[root@master tmp]# python
Python 2.7.5 (default, Apr 11 2018, 07:36:10)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import salt.client
```

```
>>> client = salt.client.LocalClient()
>>> ret = client.cmd('*','file.remove',['/tmp/foo'])
>>> print ret
{'node1.salt.com': True, 'node2.salt.com': True}
```

7) iptables 模块

功能:minion的iptables主持

示例:

在所有minion主机追加规则

```
[root@master ~]# salt '*' iptables.append filter INPUT rule='-m state --state
RELATED,ESTABLISHED -j ACCEPT'
node2.salt.com:
    True
node1.salt.com:
    True
```

删除所有minion上的指定链编号为3的规则

```
[root@master ~]# salt '*' iptables.delete filter INPUT position=1
node1.salt.com:
node2.salt.com:
```

保存所有minion上的规则

```
[root@master ~]# salt '*' iptables.save /etc/sysconfig/iptables
node2.salt.com:
    Wrote 1 lines to "/etc/sysconfig/iptables"
node1.salt.com:
    Wrote 1 lines to "/etc/sysconfig/iptables"
```

API调用原理:

```
[root@master ~]# python
Python 2.7.5 (default, Apr 11 2018, 07:36:10)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import salt.client
>>> client = salt.client.LocalClient()
>>> ret = client.cmd('*','iptables.append',['filter','INPUT','rule=\'-p tcp --dport
80 -j ACCEPT\''])
>>> ret = client.cmd('*','iptables.append',['filter','INPUT','rule=\'-p tcp --sport
80 -j ACCEPT\''])
>>> print ret
```

{'node1.salt.com': True, 'node2.salt.com': True}

8) network 模块

功能:返回minion的网络信息

示例:

在minion上获取dig、ping、traceroute 目录域名信息

```
[root@master ~]# salt 'node1.salt.com' network.dig www.qq.com
node1.salt.com:
    ; <<>> DiG 9.9.4-RedHat-9.9.4-61.el7 <<>> www.qq.com
    ;; global options: +cmd
    ;; Got answer:
    ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 5529
    ;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 3, ADDITIONAL: 11

    ;; QUESTION SECTION:
    ;www.qq.com.                    IN      A

    ;; ANSWER SECTION:
    www.qq.com.             121     IN      CNAME   public-v6.sparta.mig.tencent-
cloud.net.
    public-v6.sparta.mig.tencent-cloud.net.      31 IN A 123.150.76.178
    public-v6.sparta.mig.tencent-cloud.net.      31 IN A 123.150.76.177

    ;; AUTHORITY SECTION:
    tencent-cloud.net.   121582  IN      NS      ns-open2.qq.com.
    tencent-cloud.net.   121582  IN      NS      ns-open1.qq.com.
    tencent-cloud.net.   121582  IN      NS      ns-open3.qq.com.

    ;; ADDITIONAL SECTION:
    ns-open1.qq.com.     3295    IN      A       59.36.132.139
    ns-open1.qq.com.     3295    IN      A       120.204.1.100
    ns-open1.qq.com.     3295    IN      A       203.205.236.176
    ns-open2.qq.com.     1182    IN      A       182.254.48.197
    ns-open2.qq.com.     1182    IN      A       182.254.116.5
    ns-open2.qq.com.     1182    IN      A       58.251.121.188
    ns-open3.qq.com.     463     IN      A       203.205.220.25
    ns-open3.qq.com.     463     IN      A       58.246.221.60
```

```
    ns-open3.qq.com.      463     IN      A       121.51.167.100
    ns-open3.qq.com.      463     IN      A       125.39.46.36
    ns-open3.qq.com.      463     IN      A       180.163.22.39

    ;; Query time: 23 msec
    ;; SERVER: 192.168.1.1#53(192.168.1.1)
    ;; WHEN: Wed Jun 03 16:15:43 CST 2020
    ;; MSG SIZE  rcvd: 357
[root@master ~]# salt 'node1.salt.com' network.ping www.qq.com
node1.salt.com:
    PING public-v6.sparta.mig.tencent-cloud.net (123.150.76.178) 56(84) bytes of
data.
    64 bytes from 123.150.76.178 (123.150.76.178): icmp_seq=1 ttl=128 time=42.7 ms
    64 bytes from 123.150.76.178 (123.150.76.178): icmp_seq=2 ttl=128 time=43.9 ms
    64 bytes from 123.150.76.178 (123.150.76.178): icmp_seq=3 ttl=128 time=41.1 ms
    64 bytes from 123.150.76.178 (123.150.76.178): icmp_seq=4 ttl=128 time=42.1 ms

    --- public-v6.sparta.mig.tencent-cloud.net ping statistics ---
    4 packets transmitted, 4 received, 0% packet loss, time 3004ms
    rtt min/avg/max/mdev = 41.154/42.496/43.926/1.021 ms
[root@master ~]# salt 'node1.salt.com' network.traceroute www.qq.com
node1.salt.com:
    |_
      ----------
      count:
          1
      hostname:
          192.168.200.1
      ip:
          192.168.200.1
      ms1:
          0.217
      ms2:
          0.169
      ms3:
          0.142
    |_
```

```
      ----------
      count:
          2
      hostname:
          *
    |_
      ----------
      count:
          3
      hostname:
          *
    |_
      ----------
      count:
          4
      hostname:
          *
------------------------------省略
```

获取minion的mac地址

```
[root@master ~]# salt 'node1.salt.com' network.hwaddr ens32
node1.salt.com:
    00:0c:29:cd:9b:a8
[root@master ~]# salt 'node2.salt.com' network.hwaddr ens32
node2.salt.com:
    00:0c:29:af:e0:c4
```

检测minion是否属于10.0.0.0/16 这个子网

```
[root@master ~]# salt 'node1.salt.com' network.in_subnet 192.168.0.0/16
node1.salt.com:
    True
[root@master ~]# salt 'node2.salt.com' network.in_subnet 192.168.0.0/16
node2.salt.com:
    True
```

获取minion的网卡配置信息

```
[root@master ~]# salt 'node1.salt.com' network.interfaces
node1.salt.com:
    ----------
    ens32:
```

```
    ----------
    hwaddr:
        00:0c:29:cd:9b:a8
    inet:
        |_
          ----------
          address:
              192.168.200.108
          broadcast:
              192.168.200.255
          label:
              ens32
          netmask:
              255.255.255.0
    inet6:
        |_
          ----------
          address:
              fe80::4754:d238:a6ef:c9aa
          prefixlen:
              64
          scope:
              link
    up:
        True
lo:
    ----------
    hwaddr:
        00:00:00:00:00:00
    inet:
        |_
          ----------
          address:
              127.0.0.1
          broadcast:
              None
          label:
```

```
            lo
          netmask:
            255.0.0.0
    inet6:
        |_
          ----------
          address:
            ::1
          prefixlen:
            128
          scope:
            host
    up:
        True
virbr0:
    ----------
    hwaddr:
        52:54:00:76:88:ad
    inet:
        |_
          ----------
          address:
            192.168.122.1
          broadcast:
            192.168.122.255
          label:
            virbr0
          netmask:
            255.255.255.0
    up:
        True
virbr0-nic:
    ----------
    hwaddr:
        52:54:00:76:88:ad
    up:
        False
```

获取minion的ip地址信息

```
[root@master ~]# salt 'node1.salt.com' network.ip_addrs
node1.salt.com:
    - 192.168.122.1
    - 192.168.200.108
[root@master ~]# salt 'node2.salt.com' network.ip_addrs
node2.salt.com:
    - 192.168.122.1
    - 192.168.200.109
```

获取minion的子网信息

```
[root@master ~]# salt 'node2.salt.com' network.subnets
node2.salt.com:
    - 192.168.200.0/24
    - 192.168.122.0/24
[root@master ~]# salt 'node1.salt.com' network.subnets
node1.salt.com:
    - 192.168.200.0/24
    - 192.168.122.0/24
```

API调用原理:

```
[root@master ~]# python
Python 2.7.5 (default, Apr 11 2018, 07:36:10)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import salt.client
>>> client = salt.client.LocalClient()
>>> ret = client.cmd('node1.salt.com','network.ip_addrs')
>>> print ret
{'node1.salt.com': ['192.168.122.1', '192.168.200.108']}
```

9) pkg 模块 ---自动匹配版本系统（ubantu,centos)

功能:minion程序包管理，如yum、apt-get

示例:

所有minion安装php

```
[root@master ~]# salt '*' pkg.install php
node1.salt.com:
    ----------
    libzip:
        ----------
```

```
        new:
            0.10.1-8.el7
        old:
    php:
        ----------
        new:
            5.4.16-48.el7
        old:
    php-cli:
        ----------
        new:
            5.4.16-48.el7
        old:
    php-common:
        ----------
        new:
            5.4.16-48.el7
        old:
node2.salt.com:
    ----------
    libzip:
        ----------
        new:
            0.10.1-8.el7
        old:
    php:
        ----------
        new:
            5.4.16-48.el7
        old:
    php-cli:
        ----------
        new:
            5.4.16-48.el7
        old:
    php-common:
        ----------
```

```
        new:
            5.4.16-48.el7
        old:
```

所有minion卸载php

```
[root@master ~]# salt '*' pkg.remove php
node2.salt.com:
    ----------
    php:
        ----------
        new:
        old:
            5.4.16-48.el7
node1.salt.com:
    ----------
    php:
        ----------
        new:
        old:
            5.4.16-48.el7
```

升级所有minion.上所有的软件包

```
[root@master ~]# salt '*' pkg.upgrade          # 最好别轻易使用
```

API调用原理:

```
[root@master ~]# python
Python 2.7.5 (default, Apr 11 2018, 07:36:10)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import salt.client
>>> client = salt.client.LocalClient()
>>> ret = client.cmd('*','pkg.install',['php'])
>>> ret = client.cmd('*','pkg.remove',['php'])
>>> print ret
{'node1.salt.com': {}, 'node2.salt.com': {}}
```

10) service 模块

这里可以插一个故障，哈哈，好像有挺多这样的，开机就没有了

saltstack中的master无法连接minion

原因经过一晚后，minion自动关闭了

```
[root@node1 ~]# systemctl start salt-minion
```

需要开启才能连接

功能:管理minion的服务

示例:

针对httpd服务的开机自启动、禁用httpd、reload、restart、 start、 stop、 status 操作

```
[root@master ~]# salt '*' service.enable httpd
node2.salt.com:
    True
node1.salt.com:
    True
[root@node1 ~]# systemctl is-enabled httpd
enabled
[root@node1 ~]# systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset:
disabled
)   Active: inactive (dead)
     Docs: man:httpd(8)
           man:apachectl(8)
[root@master ~]# salt '*' service.start httpd
node2.salt.com:
    True
node1.salt.com:
    True
```

API调用原理:

```
[root@master ~]# python
Python 2.7.5 (default, Apr 11 2018, 07:36:10)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-28)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import salt.client
>>> client = salt.client.LocalClient()
>>> ret = client.cmd('*','service.stop',['httpd'])
>>> print ret
{'node1.salt.com': True, 'node2.salt.com': True}
```

注意：

当我们后续搭建环境的时候，我们用到的比较重要的模块有

pkg模块：装软件包

file模块：file.cp

service模块：控制服务的启动

11)其他模块

saltstack还提供了user(系统用户模块)、group(系统组模块)、partition(系统分区模块)、puppet(puppet管理模块)、system(系统重启、关机模块)、timezone(市区管理模块)、nginx(nginx管理模块)、mount(文件系统挂载模块)等等，当然我们也可以通过Python扩展功能来满足需求。

```
[root@master ~]# salt '*' mount.umount /media/cdrom /dev/sr0
node1.salt.com:
    True
node2.salt.com:
    True
[root@master ~]# salt '*' mount.mount /media/cdrom /dev/sr0 true
node1.salt.com:
    True
node2.salt.com:
    True
```

提醒：

模块这块的话，平时没事多看看，saltstack和ansible一样

一般来说，对于saltstack和ansible来说，很多公司必须要做一个二选一

公司中ansible用的比较多，所以saltstack只是掩饰一下，不会讲太多

后面的远程执行只是介绍一下