

# Kafka 2.3.1 实战

大眼鱼叔叔



# 目 录

封面

前言

目录

Kafka 安装部署

单节点环境安装

伪分布式环境安装

多节点环境安装

consumer 开发

消费者参数

Kafka Connect

插件

JDBC Connector

Oracle Connector

示例

Oracle 数据库表备份

从 Oracle 同步数据到 MongoDB

从 MongoDB 同步数据到 Oracle

附录 A

新建 VMWare 虚拟机

安装 CentOS 7.7

安装 Docker

安装 Oracle JDK 1.8

附录 B

Chrony

MobaXterm

Kafka Tool

# 封面

Java Developer 系列

# Kafka 2.3

## 开发指南

大眼鱼叔叔 ( QQ : 82414029 )

# 前言

---

# 目录

---

## 目录

---

- [Kafka 安装](#)
  - [开发环境](#)
    - [单节点环境](#)
    - [单节点伪分布式环境](#)
    - [多节点环境](#)
  - [生产环境](#)
- [附录 A](#)
  - [新建 VMWare 虚拟机](#)
  - [安装 CentOS 7.7](#)
  - [安装 Oracle JDK 1.8](#)
- [附录 B](#)
  - [MobaXterm](#)
  - [Kafka Tool 2.0.6](#)

# Kafka 安装部署

---

[单节点环境安装](#)

[伪分布式环境安装](#)

[多节点环境安装](#)

# 单节点环境安装

- [在 CentOS 7 上安装 Kafka 2.3.1 单节点环境](#)
  - [前提](#)
  - [安装](#)
  - [测试](#)
  - [关闭](#)
  - [调优](#)
    - [操作系统调优](#)
    - [JVM 调优](#)
- [参考资料](#)

## 在 CentOS 7 上安装 Kafka 2.3.1 单节点环境

### 前提

假设我们已经满足了以下条件，如果没有的话，请参考 附录A：

- 一台2核4G的虚拟机，IP 地址192.168.80.81
- 已安装 CentOS 7.7 64 位操作系统
- 具备 root 用户权限
- 已安装 JDK 1.8.0\_221

### 安装

#### 1. 下载 Kafka

从 [Apache Kafka](#) 官网下载推荐的二进制包 [kafka\\_2.12-2.3.1.tgz](#)，并上传到 /opt 目录解压缩：

```
# cd /opt
# tar -xzf kafka_2.12-2.3.1.tgz
# chown -R lemon:oper /opt/kafka_2.12-2.3.1
# cd /opt/kafka_2.12-2.3.1
```

#### 1. 启动服务器

Kafka 使用 [ZooKeeper](#)，如果你还没有 ZooKeeper 服务器，你需要先启动一个 ZooKeeper 服务器。您可以通过与 kafka 打包在一起的便捷脚本来快速简单地创建一个单节点 ZooKeeper 实例。

```
$ nohup bin/zookeeper-server-start.sh config/zookeeper.properties > zookeeper.log &
...
[2020-01-11 21:35:36,457] INFO binding to port 0.0.0.0/0.0.0.0:2181 (org.apache.zookeeper.server.NIOServerCnxnFactory)
```

稍等片刻，待 Zookeeper 启动完成后，再启动 Kafka 服务器：

```
$ export JMX_PORT=9988
$ nohup bin/kafka-server-start.sh config/server.properties > kafka-server.log &
...
[2020-01-11 21:37:22,229] INFO [KafkaServer id=0] started (kafka.server.KafkaServer)
```

## 测试

### 1. 创建一个 topic

让我们创建一个名为 “test” 的 topic，它有一个分区和一个副本：

```
$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test
Created topic test.
```

现在我们可以运行 list (列表) 命令来查看这个 topic：

```
$ bin/kafka-topics.sh --list --zookeeper localhost:2181
test
```

### 1. 发送一些消息

Kafka 自带一个命令行客户端，它从文件或标准输入中获取输入，并将其作为 message (消息) 发送到 Kafka 集群。默认情况下，每行将作为单独的 message 发送。

使用另一个终端运行 producer，然后在控制台输入一些消息以发送到服务器。

```
$ cd /opt/kafka_2.12-2.3.1/
$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test
```



```
> Hello World!
> Hello China!
^C
```

## 1. 启动一个 consumer

Kafka 还有一个命令行 consumer（消费者），将消息转储到标准输出。

```
$ cd /opt/kafka_2.12-2.3.1/
$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test --from-beginning
Hello World!
Hello China!
^C
```

如果您将上述命令在不同的终端中运行，那么现在就可以将消息输入到生产者终端中，并将它们在消费终端中显示出来。

所有的命令行工具都有其他选项；运行不带任何参数的命令将显示更加详细的使用信息。

## 2. 生产者吞吐量测试

kafka-producer-perf-test 脚本是 Kafka 提供的用于测试 producer 性能的脚本，该脚本可以很方便地计算出 producer 在一段时间内的吞吐量和平均延时

```
$ bin/kafka-producer-perf-test.sh --topic test --num-records 500000 --record-size 200 --throughput -1 --producer-props bootstrap.servers=localhost:9092 acks=-1
221506 records sent, 44063.3 records/sec (8.40 MB/sec), 2382.8 ms avg latency, 3356.0 ms max latency.
500000 records sent, 64102.564103 records/sec (12.23 MB/sec), 2078.40 ms avg latency, 3356.00 ms max latency, 1841 ms 50th, 3250 ms 95th, 3343 ms 99th, 3354 ms 99.9th.
```

输出结果表明在这台测试机上运行一个 kafka producer，平均每秒发送 64102 条记录，平均吞吐量是每秒 12.23MB（占用 97.84Mb/s 左右的宽带），平均延时 2078 毫秒，最大延时 3356 毫秒，50% 的消息发送需 1841 毫秒，95% 的消息发送需 3250 毫秒，99% 的消息发送需 3343 毫秒，99.9 的消息发送需 3354 毫秒。

## 3. 消费者吞吐量测试

和 kafka-producer-perf-test 脚本类似，Kafka 为 consumer 也提供了方便、便捷的性能测试脚本，即

kafka-consumer-perf-test 脚本。我们首先用它在刚刚搭建的 Kafka 集群环境中测试一下新版本 consumer 的吞吐量。

```
$ bin/kafka-consumer-perf-test.sh --broker-list localhost:9092 --messages 500000 --topic test
start.time, end.time, data.consumed.in.MB, MB.sec, data.consumed.in.nMsg, nMsg.sec, rebalance.time.ms, fetch.time.ms, fetch.MB.sec, fetch.nMsg.sec
2020-01-13 12:52:48:112, 2020-01-13 12:52:49:295, 95.3675, 80.6149, 500002, 422655.9594, 25, 1158, 82.3553, 431780.6563
```

## 关闭

1. 首先使用 kafka-server-stop 脚本关闭 kafka 集群：

```
$ bin/kafka-server-stop.sh
```

1. 然后稍等片刻，使用 zookeeper-server-stop 脚本关闭 zookeeper：

```
$ bin/zookeeper-server-stop.sh
```

## 调优

### 操作系统调优

1. 调整最大文件描述符上限：

```
# ulimit -n 1000000
```

使用 Kafka 的用户有时候会碰到 “too many files open” 的错误，这就需要为 broker 所在机器调优最大文件描述符上限。调优可参考这样的公式：broker 上可能的最大分区数 \*（每个分区平均数据量 / 平均的日志段大小 + 3）。实际一般将该值设置得很大，比如 1000000。

1. 关闭 swap：

```
# sysctl vm.swappiness=1
# vim /etc/sysctl.conf
vm.swappiness=1
```

关闭 swap 是很多使用磁盘的应用程序的常规调优手段，将 `vm.swappiness` 调整为 1 个较小的数，即大幅降低对 swap 空间的使用，以免极大地拉低性能。可以使用 `free -m` 命令验证。

## 1. 优化 /data 分区：

调整 /data 分区设置，`vim /etc/fstab`，在 defaults 后面增加 `,noatime,largeio`：

```
/dev/mapper/centos-data /data xfs defaults,noatime,largeio 0 0
```

禁止 atime 更新：由于 Kafka 大量使用物理磁盘进行消息持久化，故文件系统的选择是重要的调优步骤。对于 Linux 系统上的任何文件系统，Kafka 都推荐用户在挂载文件系统（mount）时设置 noatime 选项，即取消文件 atime（最新访问时间）属性的更新——禁掉 atime 更新避免了 inode 访问时间的写入操作，因此极大地减少了文件系统写操作数，从而提升了集群性能。Kafka 并没有使用 atime，因此禁掉它是安全的操作。可以使用 `ls -l --time=atime` 命令验证。

largeio 参数将影响 stat 调用返回的 I/O 大小。对于大数据量的磁盘写入操作而言，它能够提升一定的性能。

重新挂载 /data 分区：

```
# mount -o remount /data
```

## 1. 生产者吞吐量测试

```
$ cd /opt/kafka_2.12-2.3.1
$ bin/kafka-producer-perf-test.sh --topic test --num-records 500000 --record-size 200 --throughput -1 --producer-props bootstrap.servers=localhost:9092 acks=-1
442836 records sent, 88496.4 records/sec (16.88 MB/sec), 1311.3 ms avg latency, 1937.0 ms max latency.
500000 records sent, 91642.228739 records/sec (17.48 MB/sec), 1328.20 ms avg latency, 1937.00 ms max latency, 1368 ms 50th, 1886 ms 95th, 1930 ms 99th, 1935 ms 99.9th.
```

## 1. 消费者吞吐量测试

和 kafka-producer-perf-test 脚本类似，Kafka 为 consumer 也提供了方便、便捷的性能测试脚本，即 kafka-consumer-perf-test 脚本。我们首先用它在刚刚搭建的 Kafka 集群环境中测试一下新版本

consumer 的吞吐量。

```
$ bin/kafka-consumer-perf-test.sh --broker-list localhost:9092 --messages 500000 --topic test
start.time, end.time, data.consumed.in.MB, MB.sec, data.consumed.in.nMsg, nMsg.sec,
rebalance.time.ms, fetch.time.ms, fetch.MB.sec, fetch.nMsg.sec
2020-01-13 13:17:34:483, 2020-01-13 13:17:35:580, 95.4016, 86.9659, 500181,
455953.5096, 20, 1077, 88.5809, 464420.6128
```

## JVM 调优

### 1. 调整 JVM 启动参数：

编辑个人文件 `vim ~/.bash_profile`，将以下内容添加到文件中：

```
export KAFKA_HEAP_OPTS="-Xmx1g -Xms1g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=128m -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupancyPercent=35 -XX:G1HeapRegionSize=16M -XX:MinMetaspaceFreeRatio=50 -XX:MaxMetaspaceFreeRatio=85"
```

编译 .bash\_profile

```
source ~/.bash_profile
```

### 1. 生产者吞吐量测试

```
$ cd /opt/kafka_2.12-2.3.1
$ bin/kafka-producer-perf-test.sh --topic test --num-records 500000 --record-size 200 --throughput -1 --print-metrics --producer-props bootstrap.servers=localhost:9092 acks=-1
377763 records sent, 75552.6 records/sec (14.41 MB/sec), 1507.6 ms avg latency, 1979.0 ms max latency.
500000 records sent, 84160.915671 records/sec (16.05 MB/sec), 1466.62 ms avg latency, 1979.00 ms max latency, 1507 ms 50th, 1952 ms 95th, 1965 ms 99th, 1978 ms 99.9th.
Metric Name
      Value
app-info:commit-id:{client-id=producer-1}
      : 18a913733fb71c01
app-info:start-time-ms:{client-id=producer-1}
      : 1579140833567
app-info:version:{client-id=producer-1}
      : 2.3.1
```

```

kafka-metrics-count:count:{client-id=producer-1}
    : 102.000
producer-metrics:batch-size-avg:{client-id=producer-1}
    : 16337.068
producer-metrics:batch-size-max:{client-id=producer-1}
    : 16377.000
producer-metrics:batch-split-rate:{client-id=producer-1}
    : 0.000
producer-metrics:batch-split-total:{client-id=producer-1}
    : 0.000
producer-metrics:buffer-available-bytes:{client-id=producer-1}
    : 33554432.000
producer-metrics:buffer-exhausted-rate:{client-id=producer-1}
    : 0.000
producer-metrics:buffer-exhausted-total:{client-id=producer-1}
    : 0.000
producer-metrics:buffer-total-bytes:{client-id=producer-1}
    : 33554432.000
producer-metrics:bufferpool-wait-ratio:{client-id=producer-1}
    : 0.081
producer-metrics:bufferpool-wait-time-total:{client-id=producer-1}
    : 2834023273.000
producer-metrics:compression-rate-avg:{client-id=producer-1}
    : 1.000
producer-metrics:connection-close-rate:{client-id=producer-1}
    : 0.000
producer-metrics:connection-close-total:{client-id=producer-1}
    : 0.000
producer-metrics:connection-count:{client-id=producer-1}
    : 2.000
producer-metrics:connection-creation-rate:{client-id=producer-1}
    : 0.056
producer-metrics:connection-creation-total:{client-id=producer-1}
    : 2.000
producer-metrics:failed-authentication-rate:{client-id=producer-1}
    : 0.000
producer-metrics:failed-authentication-total:{client-id=producer-1}
    : 0.000
producer-metrics:failed-reauthentication-rate:{client-id=producer-1}
    : 0.000
producer-metrics:failed-reauthentication-total:{client-id=producer-1}
    : 0.000
producer-metrics:incoming-byte-rate:{client-id=producer-1}
    : 10062.678
producer-metrics:incoming-byte-total:{client-id=producer-1}
    : 360586.000
producer-metrics:io-ratio:{client-id=producer-1}
    : 0.012
producer-metrics:io-time-ns-avg:{client-id=producer-1}
    : 23895.015

```

```

producer-metrics:io-wait-ratio:{client-id=producer-1}
: 0.098
producer-metrics:io-wait-time-ns-avg:{client-id=producer-1}
: 204193.407
producer-metrics:io-waittime-total:{client-id=producer-1}
: 3537650773.000
producer-metrics:iotime-total:{client-id=producer-1}
: 413981132.000
producer-metrics:metadata-age:{client-id=producer-1}
: 5.829
producer-metrics:network-io-rate:{client-id=producer-1}
: 358.811
producer-metrics:network-io-total:{client-id=producer-1}
: 12858.000
producer-metrics:outgoing-byte-rate:{client-id=producer-1}
: 2939361.696
producer-metrics:outgoing-byte-total:{client-id=producer-1}
: 105329087.000
producer-metrics:produce-throttle-time-avg:{client-id=producer-1}
: 0.000
producer-metrics:produce-throttle-time-max:{client-id=producer-1}
: 0.000
producer-metrics:reauthentication-latency-avg:{client-id=producer-1}
: NaN
producer-metrics:reauthentication-latency-max:{client-id=producer-1}
: NaN
producer-metrics:record-error-rate:{client-id=producer-1}
: 0.000
producer-metrics:record-error-total:{client-id=producer-1}
: 0.000
producer-metrics:record-queue-time-avg:{client-id=producer-1}
: 1458.931
producer-metrics:record-queue-time-max:{client-id=producer-1}
: 1977.000
producer-metrics:record-retry-rate:{client-id=producer-1}
: 0.000
producer-metrics:record-retry-total:{client-id=producer-1}
: 0.000
producer-metrics:record-send-rate:{client-id=producer-1}
: 13975.068
producer-metrics:record-send-total:{client-id=producer-1}
: 500000.000
producer-metrics:record-size-avg:{client-id=producer-1}
: 286.000
producer-metrics:record-size-max:{client-id=producer-1}
: 286.000
producer-metrics:records-per-request-avg:{client-id=producer-1}
: 77.809
producer-metrics:request-latency-avg:{client-id=producer-1}
: 4.382

```

```

producer-metrics:request-latency-max:{client-id=producer-1}
    : 136.000
producer-metrics:request-rate:{client-id=producer-1}
    : 179.406
producer-metrics:request-size-avg:{client-id=producer-1}
    : 16383.432
producer-metrics:request-size-max:{client-id=producer-1}
    : 16431.000
producer-metrics:request-total:{client-id=producer-1}
    : 6429.000
producer-metrics:requests-in-flight:{client-id=producer-1}
    : 0.000
producer-metrics:response-rate:{client-id=producer-1}
    : 179.411
producer-metrics:response-total:{client-id=producer-1}
    : 6429.000
producer-metrics:select-rate:{client-id=producer-1}
    : 481.330
producer-metrics:select-total:{client-id=producer-1}
    : 17325.000
producer-metrics:successful-authentication-no-reauth-total:{client-id=producer-1} : 0.000
producer-metrics:successful-authentication-rate:{client-id=producer-1}
    : 0.000
producer-metrics:successful-authentication-total:{client-id=producer-1}
    : 0.000
producer-metrics:successful-reauthentication-rate:{client-id=producer-1}
    : 0.000
producer-metrics:successful-reauthentication-total:{client-id=producer-1}
    : 0.000
producer-metrics:waiting-threads:{client-id=producer-1}
    : 0.000
producer-node-metrics:incoming-byte-rate:{client-id=producer-1, node-id=node--1} : 12.335
producer-node-metrics:incoming-byte-rate:{client-id=producer-1, node-id=node-0} : 10064.949
producer-node-metrics:incoming-byte-total:{client-id=producer-1, node-id=node--1} : 442.000
producer-node-metrics:incoming-byte-total:{client-id=producer-1, node-id=node-0} : 360144.000
producer-node-metrics:outgoing-byte-rate:{client-id=producer-1, node-id=node--1} : 1.702
producer-node-metrics:outgoing-byte-rate:{client-id=producer-1, node-id=node-0} : 2943220.331
producer-node-metrics:outgoing-byte-total:{client-id=producer-1, node-id=node--1} : 61.000
producer-node-metrics:outgoing-byte-total:{client-id=producer-1, node-id=node-0} : 105329026.000
producer-node-metrics:request-latency-avg:{client-id=producer-1, node-id=node--1} : NaN

```

```

producer-node-metrics:request-latency-avg:{client-id=producer-1, node-id=node-0} : 4.382
producer-node-metrics:request-latency-max:{client-id=producer-1, node-id=node-1} : NaN
producer-node-metrics:request-latency-max:{client-id=producer-1, node-id=node-0} : 136.000
producer-node-metrics:request-rate:{client-id=producer-1, node-id=node-1} : 0.056
producer-node-metrics:request-rate:{client-id=producer-1, node-id=node-0} : 179.590
producer-node-metrics:request-size-avg:{client-id=producer-1, node-id=node-1} : 30.500
producer-node-metrics:request-size-avg:{client-id=producer-1, node-id=node-0} : 16388.521
producer-node-metrics:request-size-max:{client-id=producer-1, node-id=node-1} : 37.000
producer-node-metrics:request-size-max:{client-id=producer-1, node-id=node-0} : 16431.000
producer-node-metrics:request-total:{client-id=producer-1, node-id=node-1} : 2.000
producer-node-metrics:request-total:{client-id=producer-1, node-id=node-0} : 6427.000
producer-node-metrics:response-rate:{client-id=producer-1, node-id=node-1} : 0.056
producer-node-metrics:response-rate:{client-id=producer-1, node-id=node-0} : 179.615
producer-node-metrics:response-total:{client-id=producer-1, node-id=node-1} : 2.000
producer-node-metrics:response-total:{client-id=producer-1, node-id=node-0} : 6427.000
producer-topic-metrics:byte-rate:{client-id=producer-1, topic=test} : 2934343.237
producer-topic-metrics:byte-total:{client-id=producer-1, topic=test} : 104981998.000
producer-topic-metrics:compression-rate:{client-id=producer-1, topic=test} : 1.000
producer-topic-metrics:record-error-rate:{client-id=producer-1, topic=test} : 0.000
producer-topic-metrics:record-error-total:{client-id=producer-1, topic=test} : 0.000
producer-topic-metrics:record-retry-rate:{client-id=producer-1, topic=test} : 0.000
producer-topic-metrics:record-retry-total:{client-id=producer-1, topic=test} : 0.000
producer-topic-metrics:record-send-rate:{client-id=producer-1, topic=test} : 13975.459
producer-topic-metrics:record-send-total:{client-id=producer-1, topic=test} : 500000.000

```



throughput：用来进行限流控制，当设定的值小于 0 时不限流，当设定的值大于 0 时，如果发送的吞吐量大于该值时就会被阻塞一段时间。

print-metrics：指定了这个参数时会在测试完成之后打印很多指标信息，对很多测试任务而言具有一定的参考价值。

## 1. 消费者吞吐量测试

和 kafka-producer-perf-test 脚本类似，Kafka 为 consumer 也提供了方便、便捷的性能测试脚本，即 kafka-consumer-perf-test 脚本。我们首先用它在刚刚搭建的 Kafka 集群环境中测试一下新版本 consumer 的吞吐量。

```
$ bin/kafka-consumer-perf-test.sh --broker-list localhost:9092 --messages 500000 --topic test
start.time, end.time, data.consumed.in.MB, MB.sec, data.consumed.in.nMsg, nMsg.sec, rebalance.time.ms, fetch.time.ms, fetch.MB.sec, fetch.nMsg.sec
2020-01-13 13:24:56:267, 2020-01-13 13:24:57:259, 95.4016, 96.1710, 500181, 504214.7177, 26, 966, 98.7594, 517785.7143
```

## 参考资料

- [Apache Kafka QuickStart](#)

# 伪分布式环境安装

- 在 CentOS 7 上安装 Kafka 2.3.1 伪分布式环境
  - 安装
  - 测试
- 参考资料

## 在 CentOS 7 上安装 Kafka 2.3.1 伪分布式环境

到目前为止，我们一直在使用单个 broker，这并不好玩。对 Kafka 来说，单个 broker 只是一个大小为一的集群。为了深入了解它，让我们把集群扩展到三个节点（仍然在一台机器上）。

提示：在开始本节之前，请先安装单节点环境，然后调整 JVM 启动参数。

编辑个人文件 `vim ~/.bash_profile`，调整 JVM 参数：

```
export KAFKA_HEAP_OPTS="-Xmx512m -Xms512m -XX:MetaspaceSize=128m -XX:MaxMeta
spaceSize=128m -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupa
ncyPercent=35 -XX:G1HeapRegionSize=16M -XX:MinMetaspaceFreeRatio=50 -XX:MaxM
etaspaceFreeRatio=85"
```

编译 .bash\_profile，`source ~/.bash_profile`。

## 安装

1. 首先为每个 broker 创建一个配置文件：

```
$ cd /opt/kafka_2.12-2.3.1
$ cp config/server.properties config/server-1.properties
$ cp config/server.properties config/server-2.properties
```

1. 编辑 broker 1 的配置文件 `vim config/server-1.properties`，并设置如下属性：

```
broker.id=1
listeners=PLAINTEXT://:9093
log.dir=/data/kafka/kafka-logs-1
```

`broker.id` 属性是集群中每个节点的名称，这一名称是唯一的。我们必须重写监听端口和日志目录，因为我们在同一台机器上运行这些，我们不希望所有的 broker 尝试在同一个端口注册，或者覆盖彼此的数据。

1. 编辑代理2的配置文件 `vim config/server-2.properties`，并设置如下属性：

```
broker.id=2
listeners=PLAINTEXT://:9094
log.dir=/data/kafka/kafka-logs-2
```

1. 我们已经建立 Zookeeper 和一个单节点了，现在我们只需要启动两个新的节点：

```
$ bin/kafka-server-start.sh config/server-1.properties &
...
[2020-01-11 21:54:39,662] INFO [KafkaServer id=1] started (kafka.server.KafkaServer)
```

```
$ bin/kafka-server-start.sh config/server-2.properties &
...
[2020-01-11 21:55:09,813] INFO [KafkaServer id=2] started (kafka.server.KafkaServer)
```

## 测试

1. 现在创建一个副本为 3 的新 topic：

```
$ bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 3 --partitions 1 --topic my-replicated-topic
```

1. Good，现在我们有一个集群，但是我们怎么才能知道那些代理在做什么呢？运行 `describe topics` 命令来查看：

```
$ bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic my-replicated-topic
Topic:my-replicated-topic      PartitionCount:1      ReplicationFactor:3
Configs:
    Topic: my-replicated-topic      Partition: 0      Leader: 2      Rep
```

```
licas: 2,1,0 Isr: 2,1,0
```

第一行给出了所有分区的摘要，下面的每行都给出了一个分区的信息。因为我们只有一个分区，所以只有一行。

- “leader” 是负责给定分区所有读写操作的节点。每个节点都是随机选择的部分分区的领导者。
- “replicas” 是复制分区日志的节点列表，不管这些节点是 leader 还是仅仅活着。
- “isr” 是一组 “同步” replicas，是 replicas 列表的子集，它活着并被指到 leader。

1. 我们可以在已创建的原始主题上运行相同的命令来查看它的位置：

```
$ bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic test
Topic:test          PartitionCount:1      ReplicationFactor:1    Configs:
      Topic: test      Partition: 0      Leader: 0      Replicas: 0      Isr:
0
```

这没什么大不了，原来的主题没有副本且在服务器 0 上。我们创建集群时，这是唯一的服务器。

1. 让我们发表一些信息给我们的新 topic：

```
$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic my-replicated-topic
...
my test message 1
my test message 2
^C
```

1. 现在我们来消费这些消息：

```
$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --from-beginning --topic my-replicated-topic
...
my test message 1
my test message 2
^C
```

1. 让我们来测试一下容错性。Broker 1 现在是 leader，让我们来杀了它（注意要切换到 root 用户）：

```
# ps aux | grep server-1.properties
```

```
root      12452  2.5   9.6 3707296 371960 pts/0  Sl   22:53   0:07 /opt/jdk1.8.0_231/bin/java -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupancyPercent=35 -XX:+ExplicitGCInvokesConcurrent...  
# kill -9 12452
```

领导权已经切换到一个从属节点，而且节点1也不在同步副本集中了：

```
# bin/kafka-topics.sh --describe --zookeeper localhost:2181 --topic my-repl  
icated-topic  
Topic:my-replicated-topic      PartitionCount:1      ReplicationFactor:3  
    Configs:  
      Topic: my-replicated-topic      Partition: 0      Leader: 2      Rep  
licas: 2,1,0 Isr: 2,0
```

2. 不过，即便原先写入消息的 leader 已经不在，这些消息仍可用于消费：

```
$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --from-be  
ginning --topic my-replicated-topic  
my test message 1  
my test message 2  
^C
```

## 参考资料

- [Apache Kafka QuickStart](#)

# 多节点环境安装

## 检查硬件要求

### 内存要求

- 最小内存：2G
- 推荐内存：4G 或 4G 以上

使用以下命令可以确认内存大小：

```
# grep MemTotal /proc/meminfo
MemTotal:      3861364 kB
```

假如你的内存小于最低要求，还是建议您先考虑一下增加内存再继续下面的内容。

你还可以使用以下命令查看可用内存：

```
# free
```

	total	used	free	shared	buff/cache	available
Mem:	3861364	163436	3290712	11888	407216	3466676
Swap:	4063228	0	4063228			

## 系统架构

推荐使用 64 位系统运行软件。输入以下命令，确定系统架构：

```
# uname -m
x86_64
```

## 检查软件要求

### 操作系统要求

理论上 Kafka 支持大部分主流的 Linux 版本。以下是经过本人测试的操作系统：

- CentOS 7.6
- CentOS 7.7

## 附件软件要求

只需要你安装了 JDK 即可。以下是经过本人测试的 JDK 版本：

- Oracle JDK 1.8.0\_221

# consumer 开发

---

[消费者参数](#)



# 消费者参数

- [重要的消费者参数](#)

## 重要的消费者参数

### fetch.min.bytes

该参数用来配置Consumer在一次拉取请求（调用poll()方法）中能从Kafka中拉取的最小数据量，默认值为1（B）。Kafka在收到Consumer的拉取请求时，如果返回给Consumer的数据量小于这个参数所配置的值，那么它就需要进行等待，直到数据量满足这个参数的配置大小。可以适当调大这个参数的值以提高一定的吞吐量，不过也会造成额外的延迟（latency），对于延迟敏感的应用可能就不可取了。

### fetch.max.wait.ms

这个参数也和fetch.min.bytes参数有关，如果kafka仅仅参考fetch.min.bytes参数的要求，那么有可能会一直阻塞等待而无法发送响应给Consumer，显然这是不合理的。fetch.max.wait.ms参数用于指定Kafka的等待时间，默认值为500（ms）。

### max.poll.records

这个参数用来配置Consumer在一次拉取请求中拉取的最大消息数，默认值为500（条），如果消息的大小都比较小，则可以适当调大这个参数值来提升一定的消费速度。

### request.timeout.ms

这个参数用来配置Consumer等待请求响应的最长时间，默认值为30000（ms）。

### metadata.max.age.ms

这个参数用来配置元数据的过期时间，默认值为300000（ms），即5分钟。如果元数据在此参数所限定的时间范围内没有进行更新，则会被强制更新，即使没有任何分区变化或有新的broker加入。

### reconnect.backoff.ms

这个参数用来配置尝试重新连接指定主机之前的等待时间（也称为退避时间），避免频繁地连接主机，默认值为50（ms）。这种机制适用于消费者向broker发送的所有请求。

## `retry.backoff.ms`

这个参数用来配置尝试重新发送失败的请求到指定的主题分区之前的等待（退避）时间，避免在某些故障情况下频繁地重复发送，默认值为100（ms）。

## `isolation.level`

这个参数用来配置消费者的事务隔离级别。字符串类型，有效值

为“`read_uncommitted`”和“`read_committed`”，表示消费者所消费到的位置，如果设置

为“`read_committed`”，那么消费者就会忽略事务未提交的消息，即只能消费到LSO（`LastStableOffset`）的位置，默认情况下为“`read_uncommitted`”，既可以消费到HW（`High Watermark`）处的位置。

# Kafka Connect

---

[插件](#)

[示例](#)

# 插件

---

JDBC Connector

Oracle Connector

# JDBC Connector

- [JDBC Connector](#)
  - [安装 JDBC Connector](#)
  - [安装 JDBC 驱动程序](#)

## JDBC Connector

### 安装 JDBC Connector

1. 在 [Confluent Hub](#) 上下载 ZIP 文件。
2. 解压缩 ZIP 文件并且将内容复制到你想要的目录。例如，你可以创建一个名为 `share/kafka/plugins/` 目录，并将 Connector 插件内容复制到该目录下。

```
# mkdir -p /usr/local/share/kafka/plugins/  
# unzip confluentinc-kafka-connect-jdbc-5.4.0.zip  
# chown -R kafka:oper /usr/local/share/kafka/
```

3. 将这个路径添加到你的 Kafka Connect 属性文件中,Kafka Connect 使用插件路径查找插件，一个插件路径是一个以逗号分隔的目录列表。

```
$ vim config/connect-distributed.properties  
plugin.path=/usr/local/share/kafka/plugins/confluentinc-kafka-connect-jdbc-  
5.4.0/lib
```

4. 使用配置文件启动 Connet Worker，Connect 将通过这些插件发现所有的连接器。

```
$ nohup bin/connect-distributed.sh config/connect-distributed.properties &  
... ..  
[2020-01-19 15:09:33,463] INFO [Worker clientId=connect-1, groupId=connect-  
cluster] Starting connectors and tasks using config offset 35 (org.apache.k  
afka.connect.runtime.distributed.DistributedHerder:1000)  
[2020-01-19 15:09:33,463] INFO [Worker clientId=connect-1, groupId=connect-  
cluster] Finished starting connectors and tasks (org.apache.kafka.connect.r  
untime.distributed.DistributedHerder:1021)
```

5. 在运行 Connect Worker 的每台机器上重复这些步骤，以保证每个 Worker 上的连接器都已经被激活。
6. 验证插件是否被成功加载。

```
$ curl http://localhost:8083/connector-plugins | jq  
[
```

```
{
  "class": "io.confluent.connect.jdbc.JdbcSinkConnector",
  "type": "sink",
  "version": "5.4.0"
},
{
  "class": "io.confluent.connect.jdbc.JdbcSourceConnector",
  "type": "source",
  "version": "5.4.0"
},
{
  "class": "org.apache.kafka.connect.file.FileStreamSinkConnector",
  "type": "sink",
  "version": "2.3.1"
},
{
  "class": "org.apache.kafka.connect.file.FileStreamSourceConnector",
  "type": "source",
  "version": "2.3.1"
}
]
```

你应该可以看到 `JdbcSourceConnector` 、 `JdbcSinkConnector` 两个 Connector 插件

#### 7. 列出所有活动的 Connector :

```
$ curl localhost:8083/connectors
[]
```

## 安装 JDBC 驱动程序

1. 查找适合的 JDBC 4.0 的驱动 JAR 包。
  - [JDBC drivers for Oracle](#)
2. 将这些 JAR 包移动到 `share/java/kafka-connect-jdbc` 目录。

```
# mkdir -p /usr/local/share/java/kafka-connect-jdbc/
# chown -R lemon:oper /usr/local/share/java/kafka-connect-jdbc/
```

#### 3. 设置环境变量 :

```
$ export CLASSPATH=./usr/local/share/java/kafka-connect-jdbc/ojdbc8.jar
```

#### 4. 重启 Connect Worker 节点。例如 :

```
$ jps -l
87537 org.apache.kafka.connect.cli.ConnectDistributed
```

```
$ kill -9 87537  
$ nohup bin/connect-distributed.sh config/connect-distributed.properties &
```

5. 验证插件是否被成功加载。

```
$ curl http://localhost:8083/connector-plugins | jq
```

# Oracle Connector

- [Kafka Source Connector For Oracle](#)
  - [安装 Oracle Connector](#)
  - [安装 Oracle JDBC 驱动程序](#)

## Kafka Source Connector For Oracle

### 安装 Oracle Connector

1. 下载 [kafka-connect-oracle](#) 源代码。
2. 编译源代码。

```
mvn clean package -Dmaven.test.skip=true
mvn dependency:copy-dependencies
```

```
# mkdir -p /usr/local/share/kafka/plugins/erdemcer-kafka-connect-oracle-1.0/
# chown -R lemon:oper /usr/local/share/kafka/
```

```
$ vim config/connect-distributed.properties
plugin.path=/usr/local/share/kafka/plugins/erdemcer-kafka-connect-oracle-1.0
```

```
[
{
"class": "com.ecer.kafka.connect.oracle.OracleSourceConnector",
"type": "source",
"version": "null"
},
{
"class": "org.apache.kafka.connect.file.FileStreamSinkConnector",
"type": "sink",
"version": "2.3.1"
```



```
},  
{  
  "class": "org.apache.kafka.connect.file.FileStreamSourceConnector",  
  "type": "source",  
  "version": "2.3.1"  
}  
]
```

## 安装 Oracle JDBC 驱动程序

---

# 示例

---

Oracle 数据库表备份

从 Oracle 同步数据到 MongoDB

从 MongoDB 同步数据到 Oracle

# Oracle 数据库表备份

- Oracle 数据库表备份
  - 准备 Oracle 数据库实例
  - 安装 JDBC Connector 插件
  - 创建 Source/Sink Connector
  - 测试
  - 清理测试环境

## Oracle 数据库表备份

这次我们将一个 Oracle 数据库实例上的 LOGIN 表的数据导入到一个 Kafka 主题上，再将它同步到另一个 Oracle 数据库实例上的 LOGIN 表。

### 准备 Oracle 数据库实例

1. 设置环境变量：

```
$ export CLASSPATH=./usr/local/share/java/kafka-connect-jdbc/ojdbc6.jar
```

2. 使用 Docker 快速安装并运行 3 个 Oracle 11g 的实例，用户名/初始密码：SYSTEM / oracle。

```
$ docker run --name oracle11g -d -p 1521:1521 -p 8080:8080 -e ORACLE_ALLOW_REMOTE=true oracleinanutshell/oracle-xe-11g
$ docker run --name oracle11g-1 -d -p 11521:1521 -p 18080:8080 -e ORACLE_ALLOW_REMOTE=true oracleinanutshell/oracle-xe-11g
```

修改密码：

```
$ docker exec -it oracle11g bash
```

```
$ sqlplus
```

```
Enter user-name: system
```

```
Enter password: oracle
```

```
SQL> alter user system identified by admin123;
```

```
SQL> exit;
```

```
$ exit
```

在每个实例下创建 TEST 用户和 LOGIN 表：

```
-- USER SQL
CREATE USER C##TEST IDENTIFIED BY "test1234"
DEFAULT TABLESPACE "USERS"
TEMPORARY TABLESPACE "TEMP";
GRANT "DBA" TO C##TEST ;
GRANT "CONNECT" TO C##TEST ;
GRANT "RESOURCE" TO C##TEST ;

-- TABLES
CREATE TABLE C##TEST.LOGIN (
  ID NUMBER NOT NULL
, USERNAME VARCHAR2(30) NOT NULL
, LOGIN_TIME DATE NOT NULL
, CONSTRAINT LOGIN_PK PRIMARY KEY (ID) ENABLE
);
CREATE SEQUENCE C##TEST.SEQ_LOGIN CACHE 20;
CREATE INDEX C##TEST.IDX_LOGIN_LT ON C##TEST.LOGIN (LOGIN_TIME DESC);
```

## 安装 JDBC Connector 插件

参见 [JDBC Connector](#)

## 创建 Source/Sink Connector

### 1. 创建 JDBC Source Connector :

```
$ echo '{"name":"oracle-jdbc-source","config":{"connector.class":"io.confluent.connect.jdbc.JdbcSourceConnector","tasks.max":"1","dialect.name":"OracleDatabaseDialect","connection.url":"jdbc:oracle:thin:@localhost:1521:xe","connection.user":"TEST","connection.password":"test1234","mode":"timestamp","table.whitelist":"LOGIN","validate.non.null":false,"timestamp.column.name":"LOGIN_TIME","topic.prefix":"oracle.test.}}}' | curl -X POST -d @- http://localhost:8083/connectors --header "Content-Type:application/json"
```

mode：表加载模式，支持的模式有 `bulk`，每次轮询时都要对整个表进行批量加载；

`incrementing`，在每个表上使用递增列以仅检测新增的行。请注意，这不会检测到对现有行的修改或删除；`timestamp`，使用时间戳（或类似时间戳的列）来检测新行和修改过的行。假设每次写入都会更新该列，并且值是单调递增的，不需要保证唯一。`timestamp+incrementing`，使用两列，用于检测新行和已修改行的时间戳列，以及用于为更新提供全局唯一ID的递增列，以便可以为每一行分配唯一的偏移量。

validate.non.null：默认情况下，JDBC 连接器将验证所有增量表和时间戳表是否将 ID 和时间戳列设置

为 NOT NULL。如果没有，则 JDBC 连接器将无法启动。将此设置为 false 将禁用这些检查。

**incrementing.column.name**：递增的列的名称，用于检测新行。空值表示自动侦测递增列。该列不能为空。

**timestamp.column.name**：用一个或多个时间戳列的逗号分隔列表，使用 COALESCE SQL 函数检测新行或修改过的行。每次轮询都会发现第一个非空时间戳值大于看到的最大先前时间戳值的行。至少一列不能为空。

## 2. 创建 JDBC Sink Connector：

```
$ echo '{"name":"oracle-jdbc-sink","config":{"connector.class":"io.confluent.connect.jdbc.JdbcSinkConnector","topics":"oracle.test.LOGIN","tasks.max":"1","connection.url":"jdbc:oracle:thin:@localhost:11521:xe","connection.user":"TEST","connection.password":"test1234","auto.create":"false","insert.mode":"insert","table.name.format":"LOGIN"}}' | curl -X POST -d @- http://localhost:8083/connectors --header "Content-Type:application/json"
```

insert.mode：要使用的插入模式，支持的模式有， insert 、 upsert 、 update 。

pk.mode：主键模式，支持的模式有 none 、 kafka 、 record\_key 、 record\_value 。

## 3. 列出所有活动的 Connector：

```
$ curl localhost:8083/connectors
["oracle-jdbc-source","oracle-jdbc-sink"]
```

# 测试

## 1. 通过 SQL 脚本在数据库实例 1 上新增测试数据：

```
DECLARE
    i NUMBER;
BEGIN
    i:=0;
    LOOP
        i:=i+1;
        INSERT INTO TEST.LOGIN(ID, USERNAME, LOGIN_TIME)
            VALUES (TEST.SEQ_LOGIN.NEXTVAL,to_char(systimestamp, 'yyyymmddhh24miss'), SYSDATE);
```

```

        IF i > 100000 THEN
            EXIT;
        END IF;
    END LOOP;
END;

```

#### 1. 在数据库实例 2 上检查数据是否已同步

```

select count(1) from test.login;
select * from test.login order by login_time desc,id desc;

```

#### 1. SQL 监控分析：

```

SELECT ADDRESS, HASH_VALUE, SQL_ID, SQL_FULLTEXT, PARSE_CALLS, EXECUTIONS,
       APPLICATION_WAIT_TIME, CONCURRENCY_WAIT_TIME/1000000 CONCURRENCY_WAIT
_TIME, CLUSTER_WAIT_TIME, USER_IO_WAIT_TIME,
       OPTIMIZER_COST, CPU_TIME/1000000 CPU_TIME, ELAPSED_TIME/1000000 ELAP
SED_TIME,
       DECODE(EXECUTIONS, 0, 0, ROUND(CPU_TIME/1000000/EXECUTIONS, 5)) AV_CPU_T
IME,
       DECODE(EXECUTIONS, 0, 0, ROUND(ELAPSED_TIME/1000000/EXECUTIONS, 5)) AV_E
LAPSED_TIME,
       SHARABLE_MEM, PERSISTENT_MEM, RUNTIME_MEM,
       DISK_READS, DIRECT_WRITES, BUFFER_GETS,
       ROWS_PROCESSED,
       FIRST_LOAD_TIME, TO_CHAR(LAST_ACTIVE_TIME, 'YYYY-MM-DD/HH24:MI:SS') L
AST_ACTIVE_TIME
FROM V$SQLAREA
WHERE PARSING_SCHEMA_NAME='TEST' AND LAST_ACTIVE_TIME >=SYSDATE-1/24
AND SQL_FULLTEXT NOT LIKE '%sys.%'
ORDER BY EXECUTIONS DESC;

```

## 清理测试环境

#### 1. 删除连接器：

```

$ curl -X DELETE http://localhost:8083/connectors/oracle-jdbc-source
$ curl -X DELETE http://localhost:8083/connectors/oracle-jdbc-sink

```

#### 1. 清空数据表：

```
TRUNCATE TABLE LOGIN;
```

1. 删除主题 oracle.test.LOGIN

# 从 Oracle 同步数据到 MongoDB

---



# 从 MongoDB 同步数据到 Oracle

---

# 附录 A

---

[新建 VMWare 虚拟机](#)

[安装 CentOS 7.7](#)

[安装 Docker](#)

[安装 Oracle JDK 1.8](#)

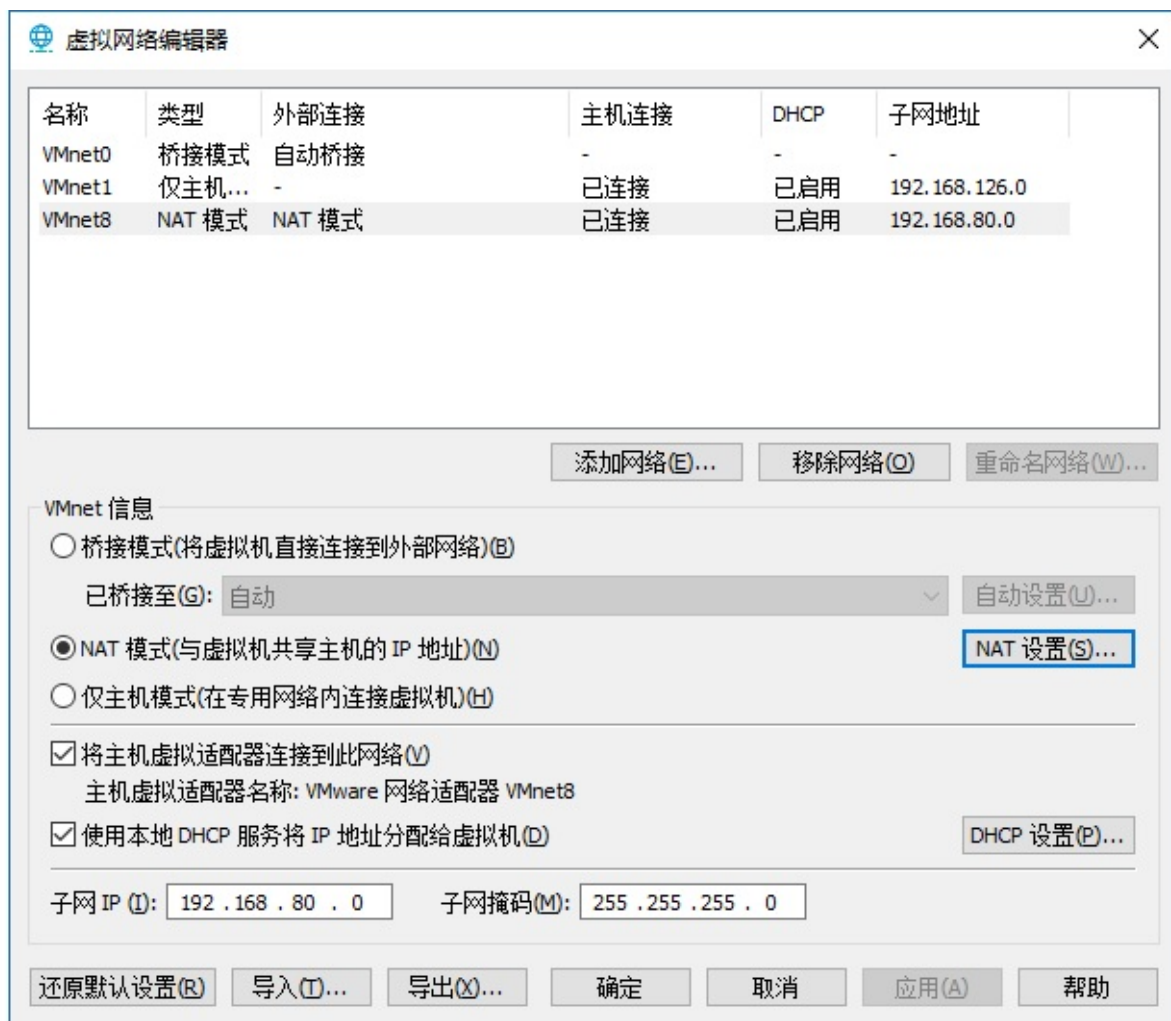
# 新建 VMWare 虚拟机

- 在 VMWare Workstation Pro 15.5.1 中新建虚拟机
  - 设置虚拟网络
  - 新建虚拟机

## 在 VMWare Workstation Pro 15.5.1 中新建虚拟机

### 设置虚拟网络

- 打开 虚拟网络编辑器 对话框，将 NAT 模式的虚拟子网地址改为 192.168.80.0：



为方便学员交流，开发环境统一设置为：192.168.80

2. 打开 NAT 设置 对话框，添加 端口转发，比如端口 22，这样其他主机就可以通过你的宿主机的地址访问虚拟机了：

NAT 设置

网络: vmnet8

子网 IP: 192.168.80.0

子网掩码: 255.255.255.0

网关 IP(G): 192.168.80.2

端口转发(E)

主机端口	类型	虚拟机 IP 地址	描述
22	TCP	192.168.80.81: 22	

添加(A)...

移除(R)

属性(P)

高级

☒ 允许活动的 FTP(F)

☒ 允许任何组织唯一标识符(O)

UDP 超时(以秒为单位)(U): 30

配置端口(C): 0

☐ 启用 IPv6(E)

IPv6 前缀(G): fd15:4ba5:5a2b:1008::/64

DNS 设置(D)...

NetBIOS 设置(N)...

确定

取消

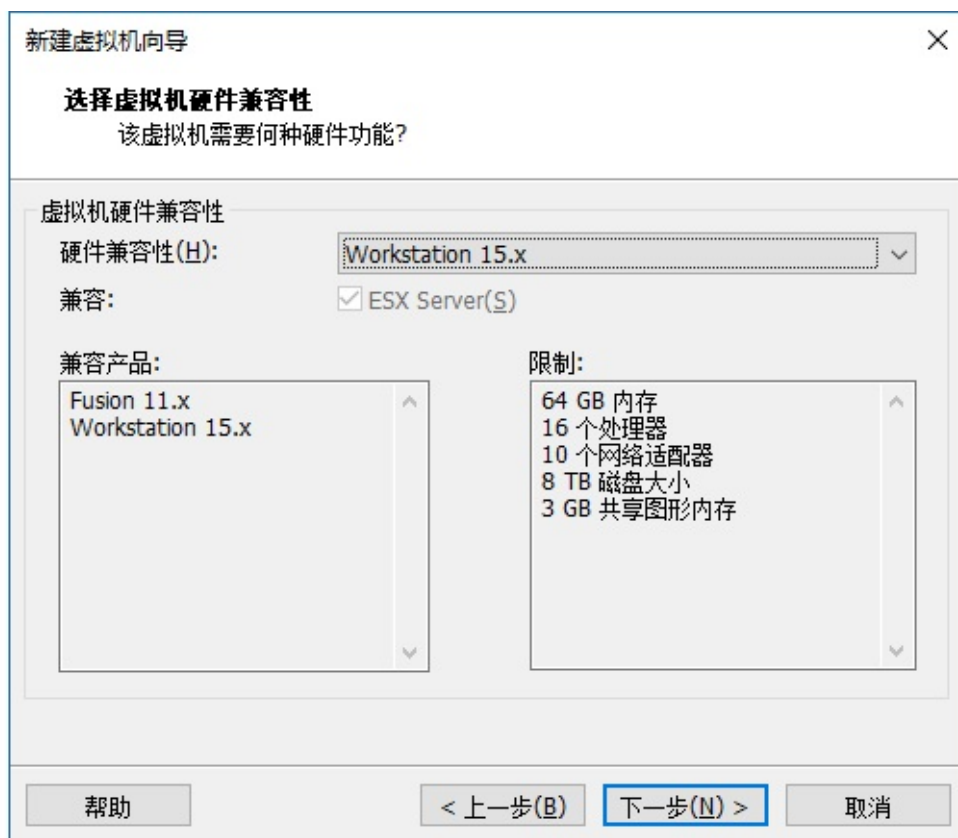
帮助

## 新建虚拟机

1. 打开 新建虚拟机向导 新建虚拟机，选择 自定义（高级）：



2. 在 选择虚拟机硬件兼容性 对话框，默认选择 Workstation 15.x：



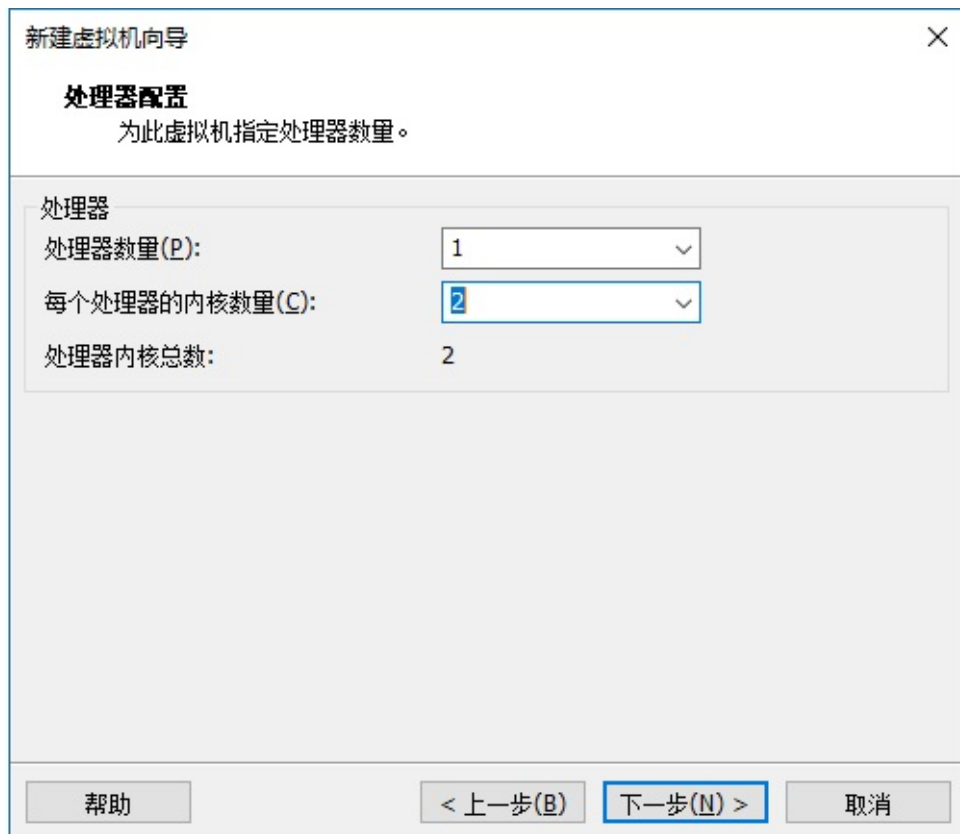
3. 在 安装客户机操作系统 对话框，选择 iso 文件：



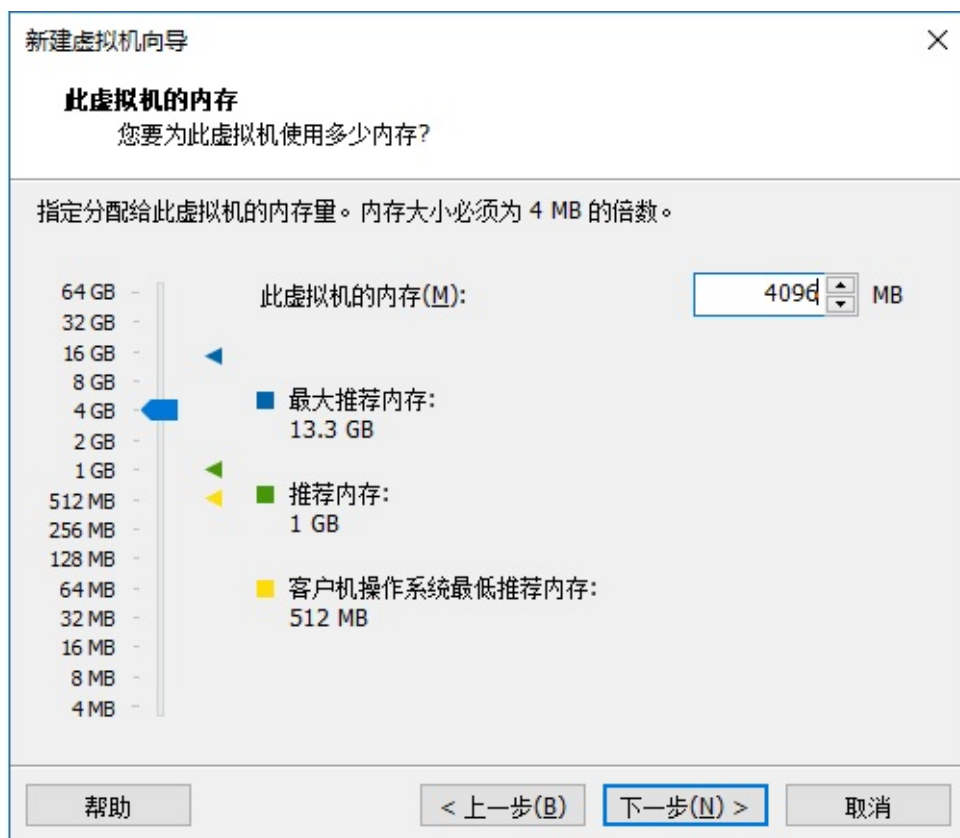
4. 在 命名虚拟机 对话框，设置虚拟机名称和位置：



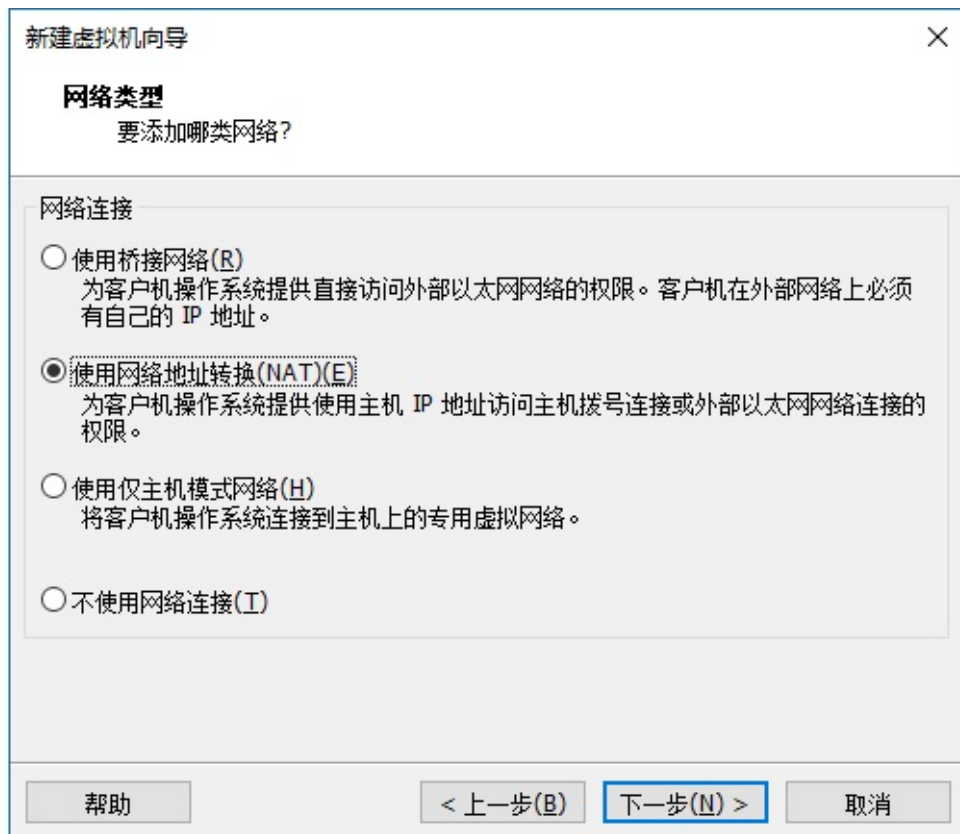
5. 在 处理器配置 对话框，选择内核数量为 2：



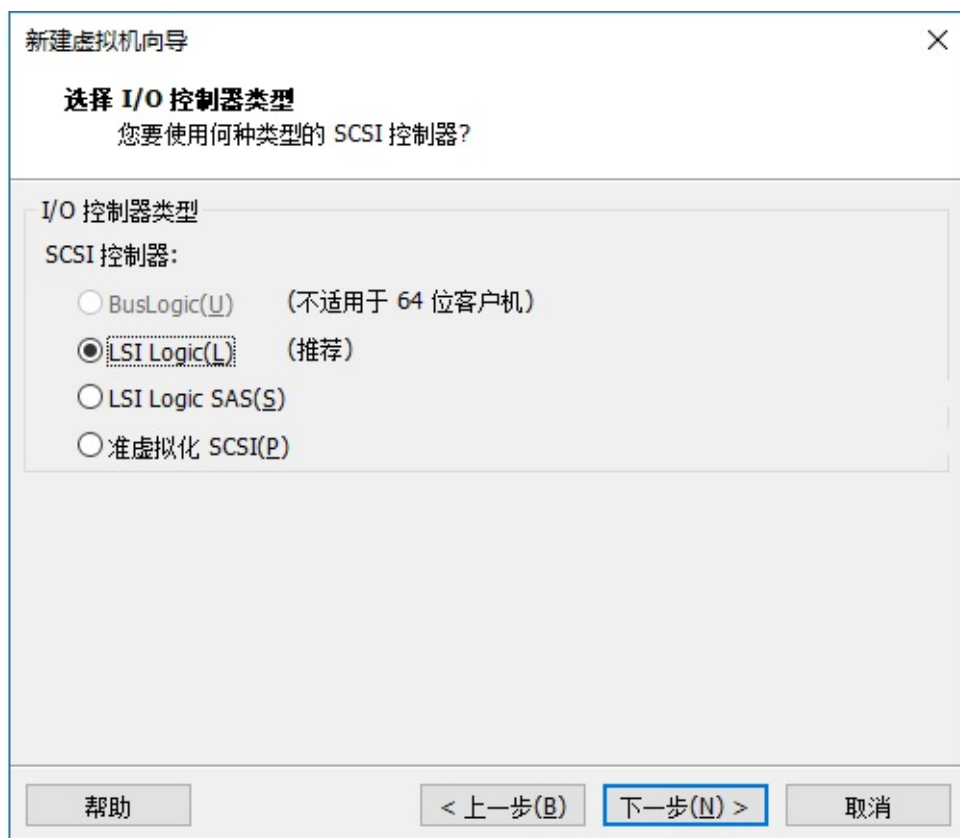
6. 在此虚拟机的内存对话框，设置此虚拟机的内存为 4096 MB：



7. 在网络类型对话框，默认选择 NAT（就是刚刚的 192.168.80 网段）：



8. 在 选择 I/O 控制器类型 对话框，默认选择 LSI Logic：

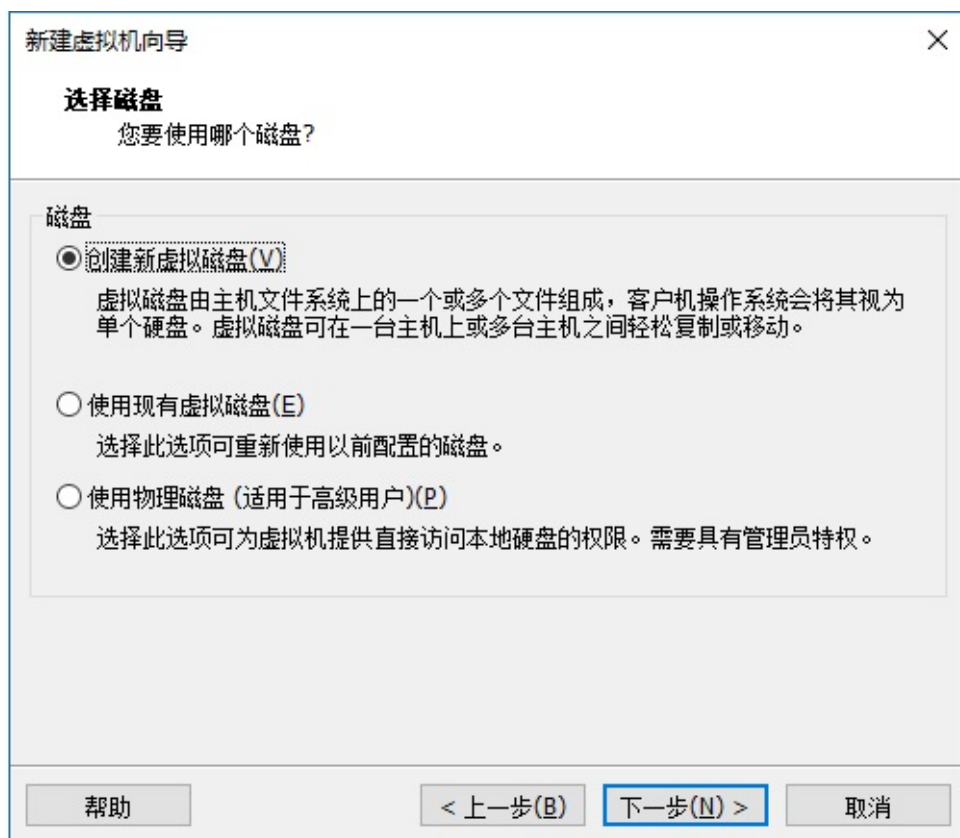


9. 在 选择磁盘类型 对话框，默认选择 SCSI：

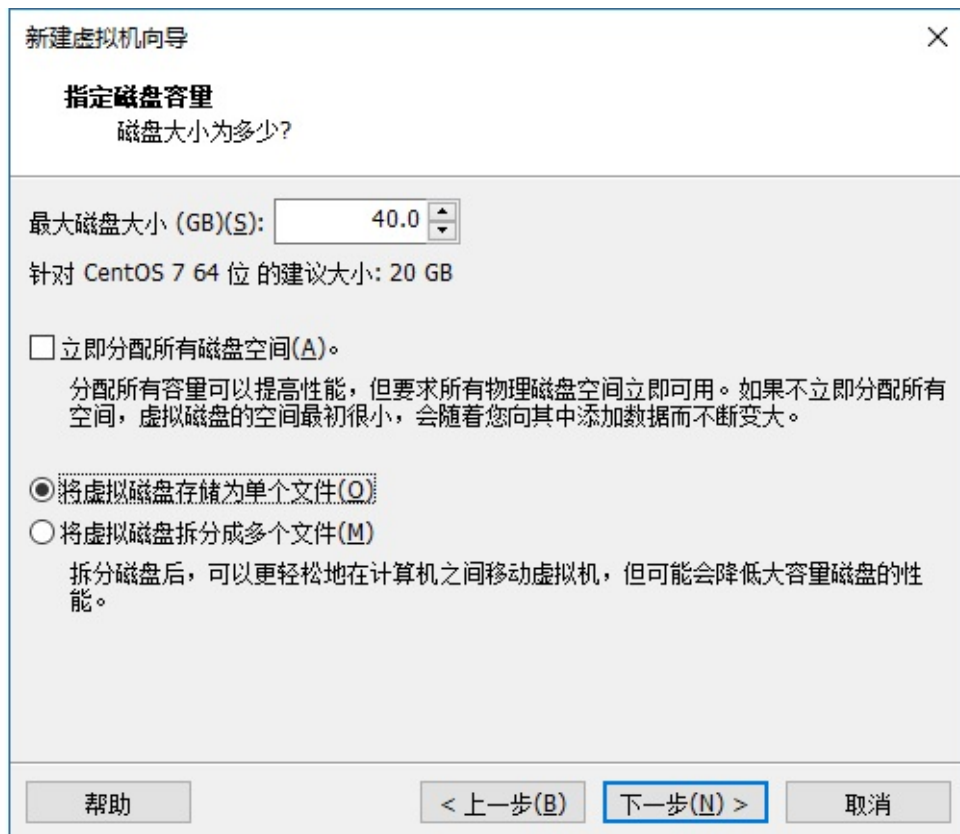




10. 在 选择磁盘 对话框，默认选择 创建新虚拟磁盘：

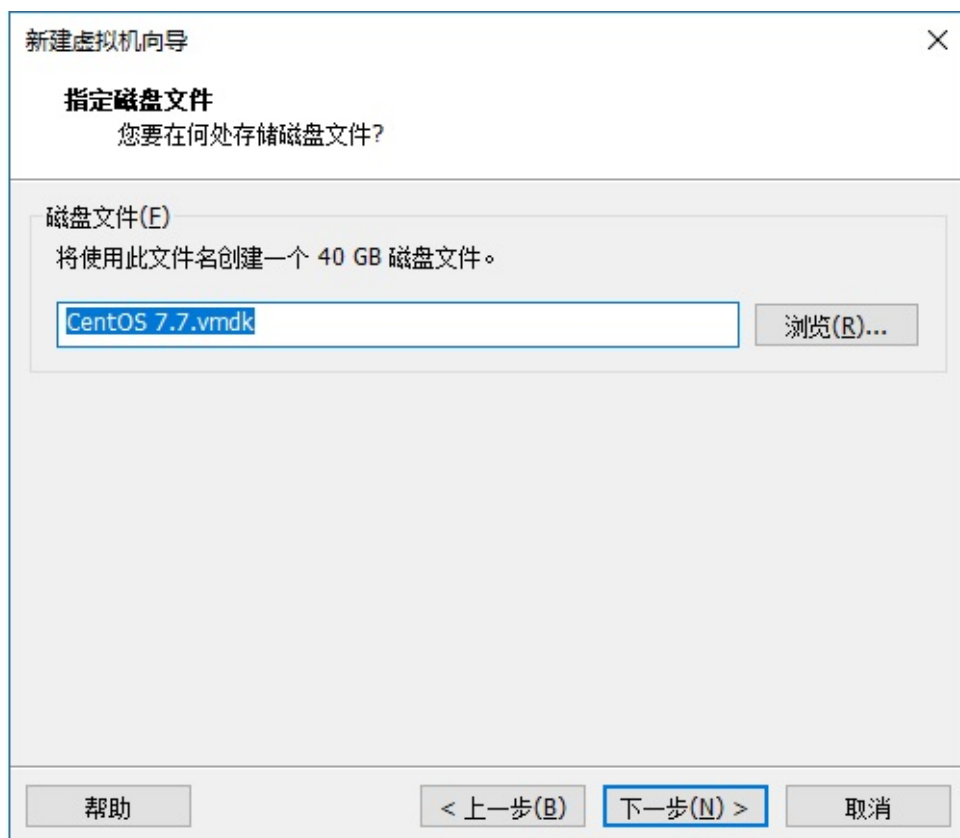


11. 在 指定磁盘容量 对话框，设置 最大磁盘大小 为 40 G，选择 将虚拟磁盘存储为单个文件：

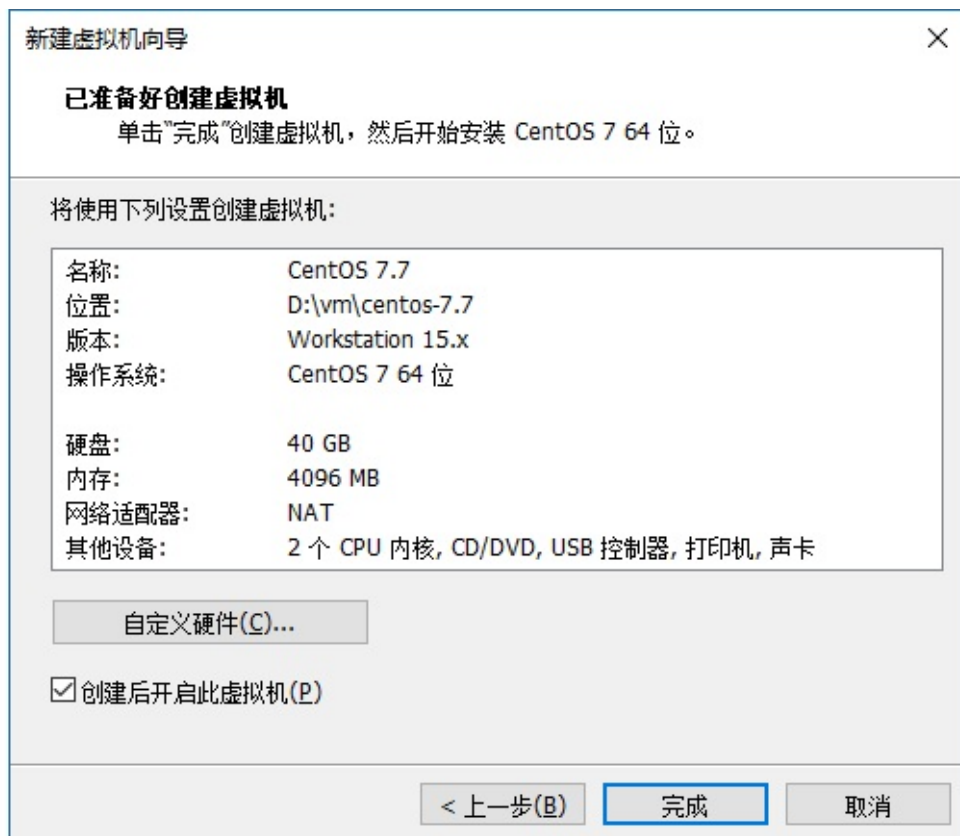


选择默认的 将虚拟磁盘拆分成多个文件 后，可能会降低磁盘的性能，所以在这里选择 将虚拟磁盘存储为单个文件。

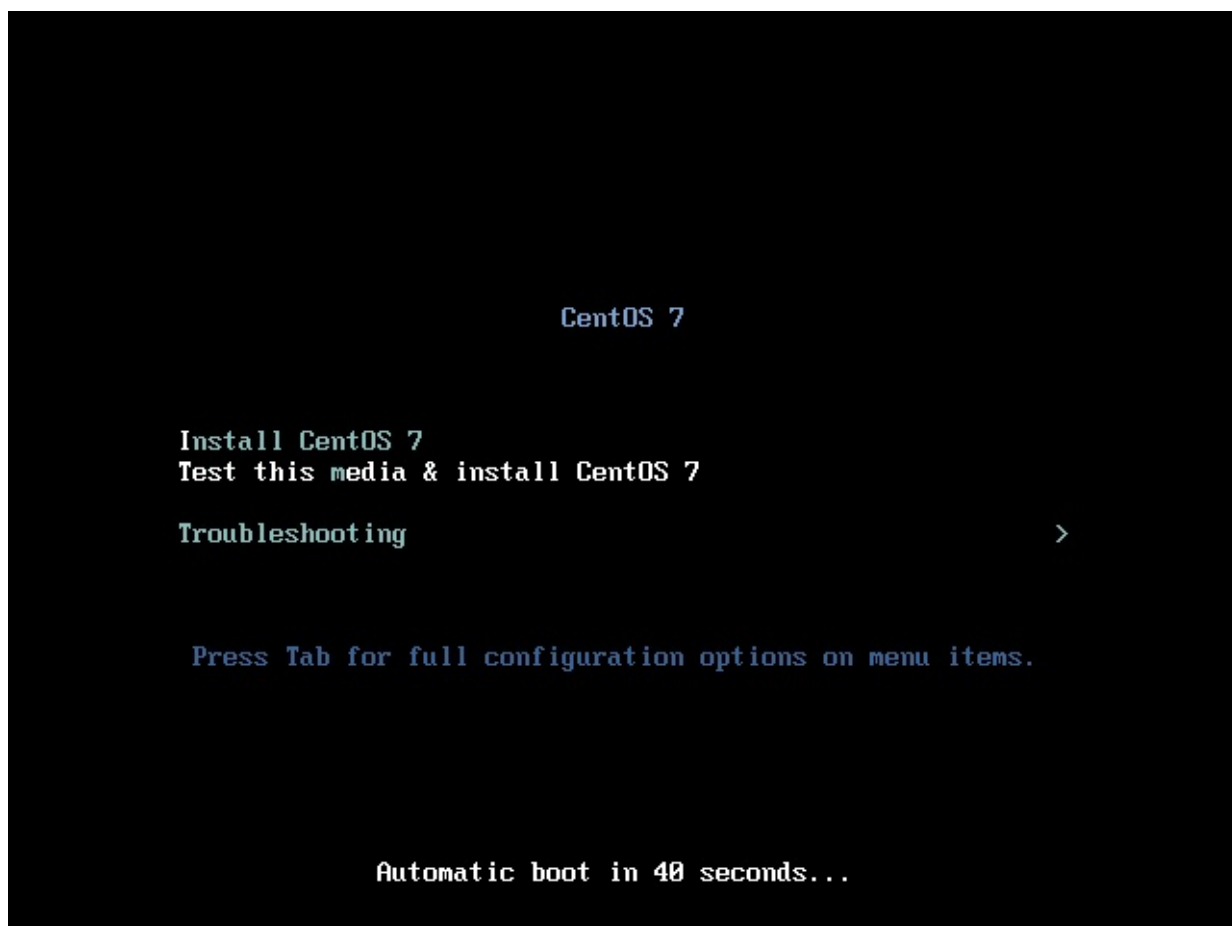
12. 在 指定磁盘文件 对话框，选择默认 磁盘文件：



13. 在 已准备号创建虚拟机 对话框，点击 完成：



14. 虚拟机启动完毕，系统自动进入 CentOS 7 安装界面，虚拟机已创建成功：



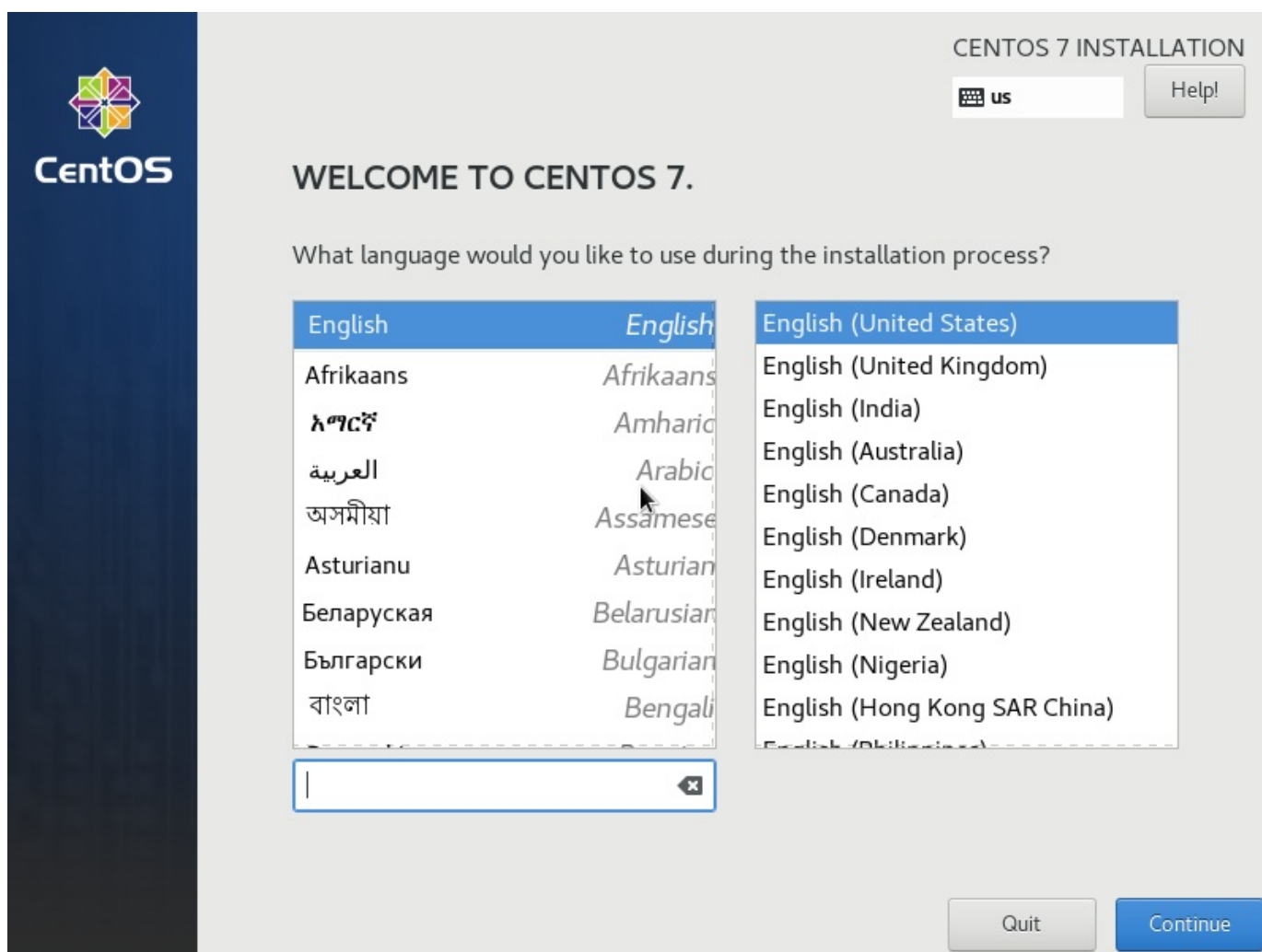
# 安装 CentOS 7.7

- [安装 CentOS 7.7 64 位版](#)
  - [安装 CentOS](#)
  - [配置 CentOS](#)
    - [关闭防火墙](#)
    - [配置国内 yum 源](#)
  - [安装附件软件](#)
  - [创建用户和组](#)

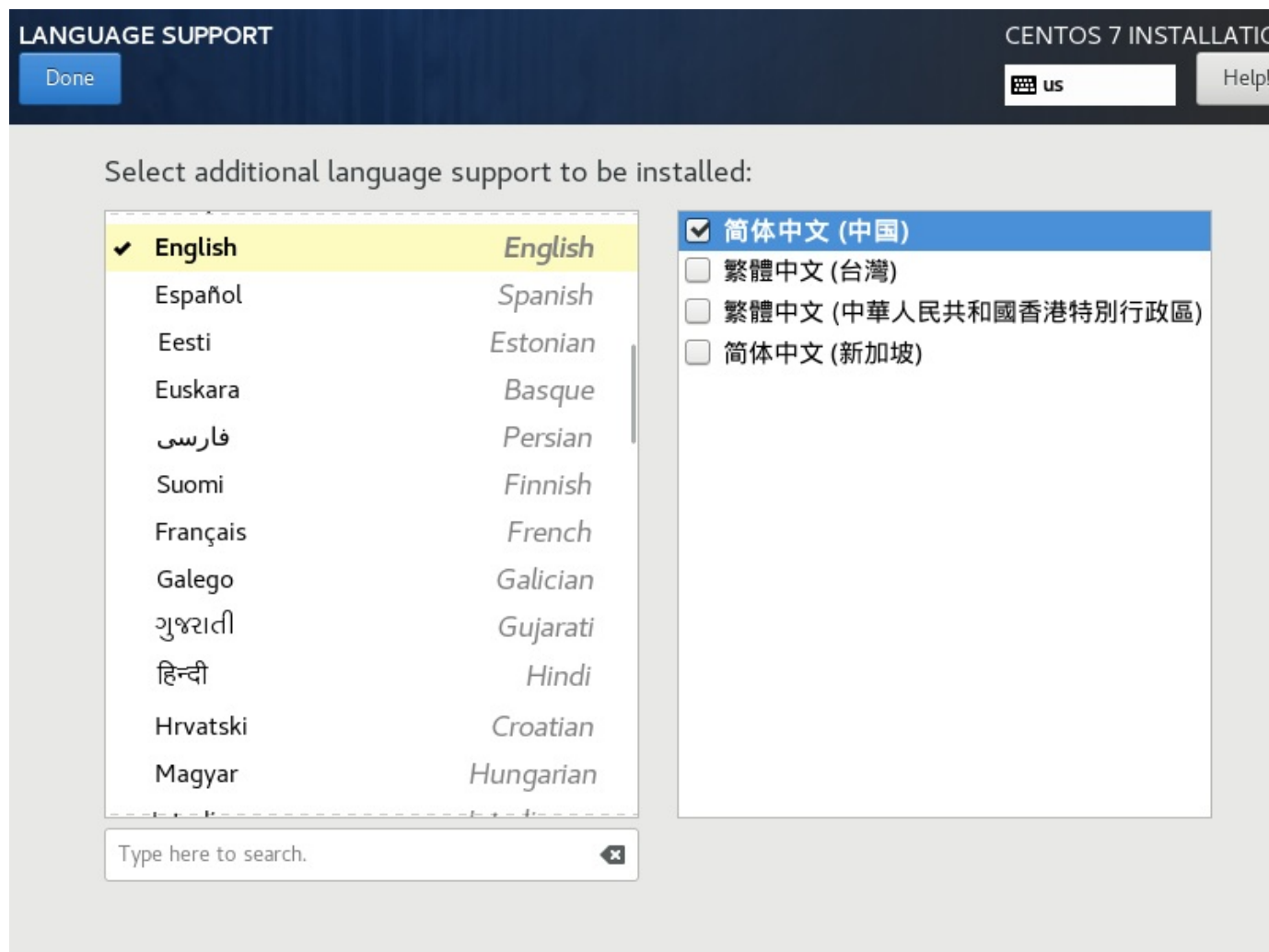
## 安装 CentOS 7.7 64 位版

### 安装 CentOS

1. 进入 CentOS 安装程序。在 WELCOME TO CENTOS 7 界面，默认选择 English 作为安装过程使用的语言。



2. 在 LANGUAGE SUPPORT 界面，选择 English 和 简体中文（中国）作为支持的语言。



3. 在 DATE & TIME 界面，设置 Region、City 以及日期和时间。

DATE & TIME


Done

CENTOS 7 INSTALLATION

us

Help!

Region: AsiaCity: ShanghaiNetwork TimeOFF




18:36 PM

24-hour

AM/PM

01 / 11 / 2020

 You need to set up networking first if you want to use NTP

4. 在 INSTALLATION DESTINATION 界面，默认选择 I will configure partitioning 自动分区配置。

INSTALLATION DESTINATION

CENTOS 7 INSTALLATION

Done

us


Help!

### Device Selection

Select the device(s) you'd like to install to. They will be left untouched until you click on the main menu's "Begin Installation" button.

#### Local Standard Disks


40 GiB



VMware, VMware Virtual S  
sda / 992.5 KiB free

Disks left unselected here will not be touched

#### Specialized & Network Disks



Add a disk...

Disks left unselected here will not be touched

### Other Storage Options

#### Partitioning

☐ Automatically configure partitioning. ☒ I will configure partitioning.

☐ I would like to make additional space available.

---

[Full disk summary and boot loader...](#)

1 disk selected; 40 GiB capacity; 992.5 KiB free [Refresh](#)

在 MANUAL PARTITIONING 界面，完成自定义分区：

The screenshot shows the 'MANUAL PARTITIONING' screen in the CentOS 7 installer. On the left, a list of partitions is shown under 'New CentOS 7 Installation':

Partition	Mount Point	Size
DATA	/data	20 GiB
centos-data		
SYSTEM	/boot	512 MiB
sda1		
<b>/</b>	<b>centos-root</b>	<b>15.5 GiB</b>
swap	centos-swap	4096 MiB

At the bottom left, it shows 'AVAILABLE SPACE 992.5 KiB' and 'TOTAL SPACE 40 GiB'. A link '1 storage device selected' is present.

On the right, the configuration for the selected partition 'centos-root' is shown:

- Mount Point:** /
- Device(s):** VMware, VMware Virtual S (sda)
- Desired Capacity:** 15.5 GiB
- Device Type:** LVM
- File System:** xfs
- Volume Group:** centos (0 B free)
- Label:** (empty)
- Name:** root

Buttons for '+', '-', and a refresh icon are at the bottom left. A 'Reset All' button is at the bottom right.

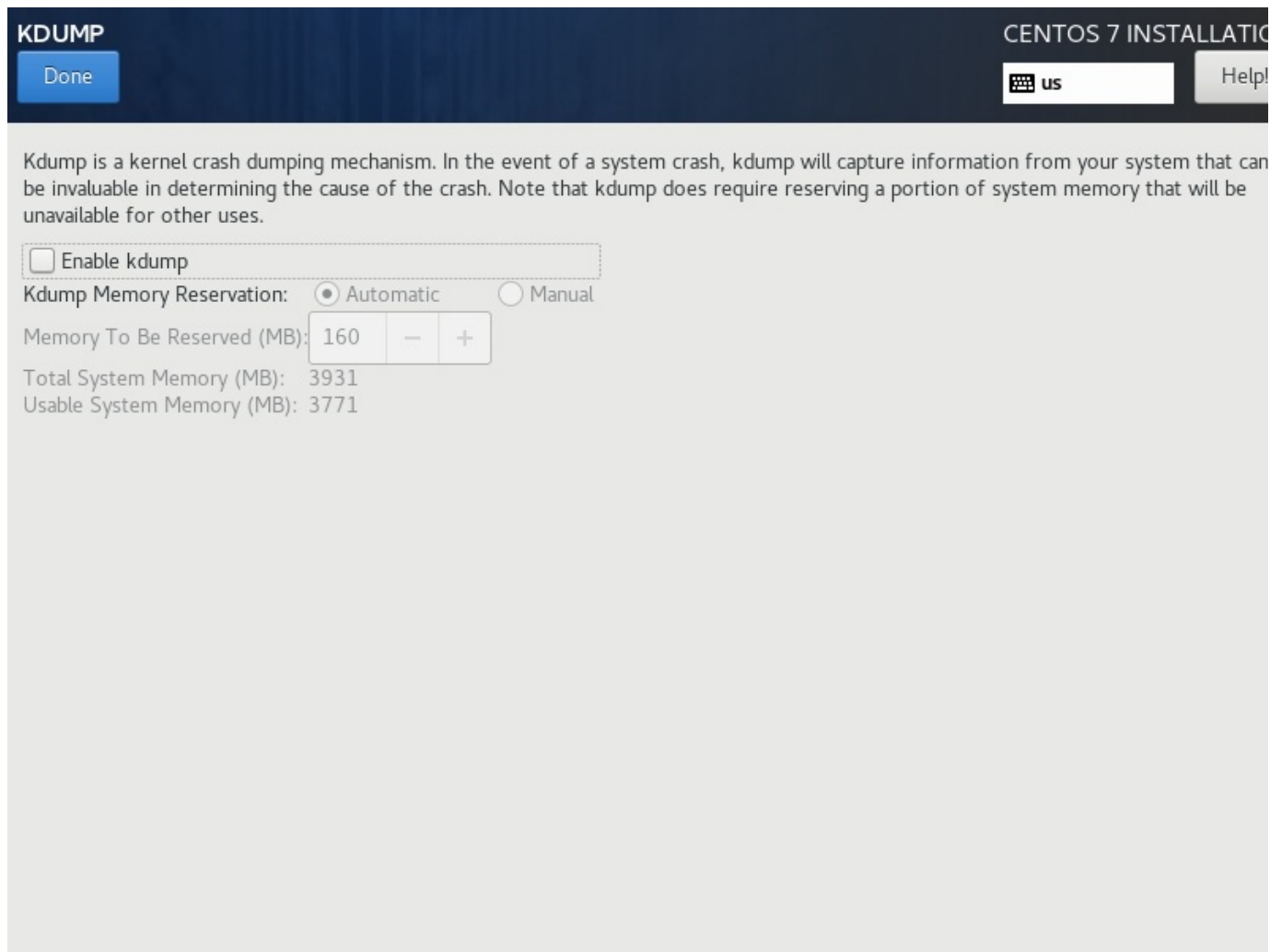
swap 分区一般建议设置成内存的 1-2 倍，这里设置成内存的 1 倍。随后的调优部分会关闭 swap，这是很多使用磁盘的应用程序的常规调优手段。

新建一个独立的数据分区，以便后续可以对特定分区进行调优。

File System 使用默认的 XFS，XFS 通常有着更好的性能。这种性能的提升主要影响是 Kafka 的写入性能。根据官网的测试报告，使用 XFS 的写入时间大约是 160 毫秒，而使用 Ext4 大约是 250 毫秒。因此生产环境中最好使用 XFS 文件系统。

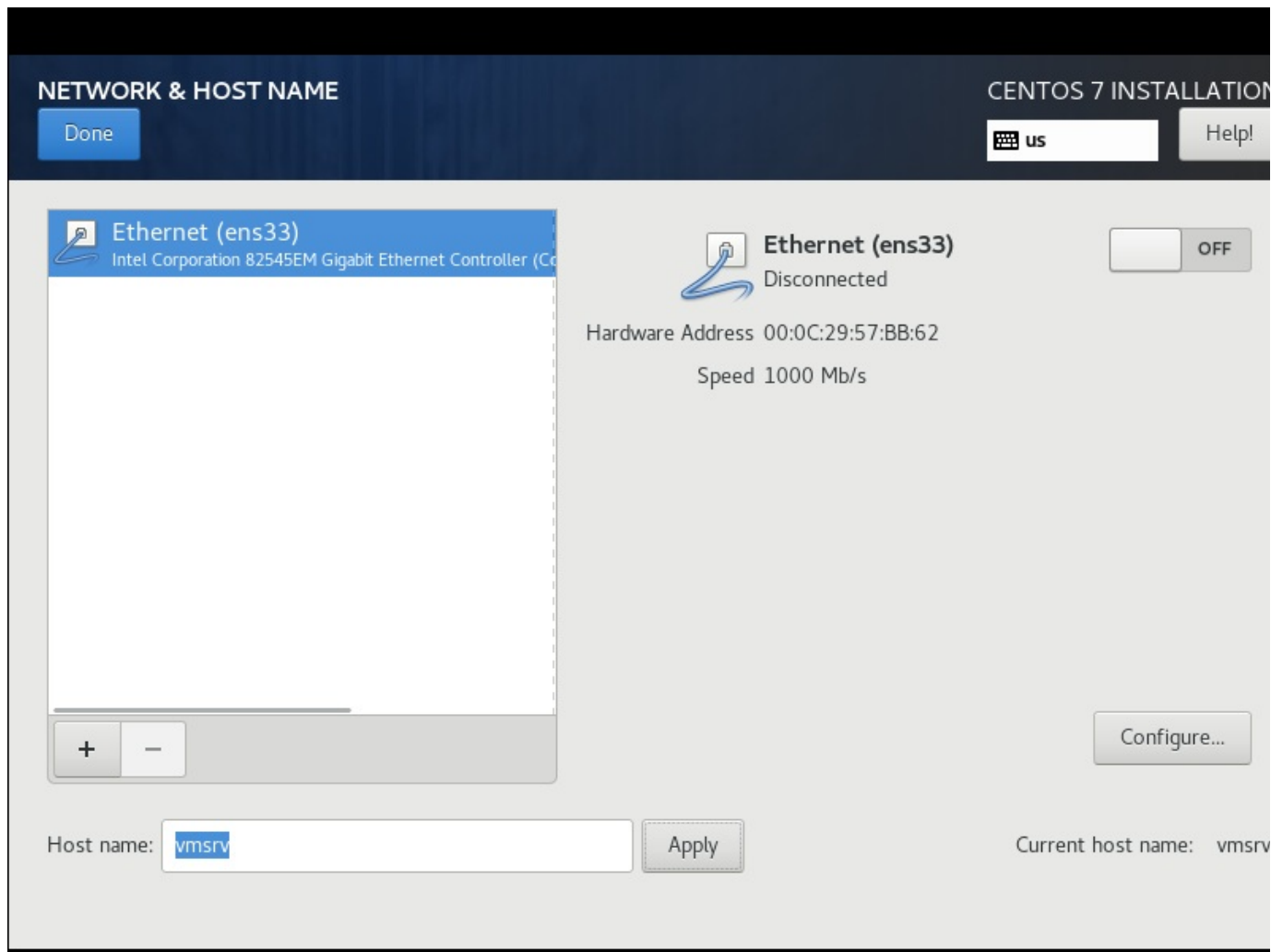
5. 在 KDUMP 界面，取消选中 Enable kdump。kdump 是在系统崩溃、死锁或者死机的时候用来转储内存运行参数的一个工具和服务，不需要分析内核崩溃原因的话，不用开启。



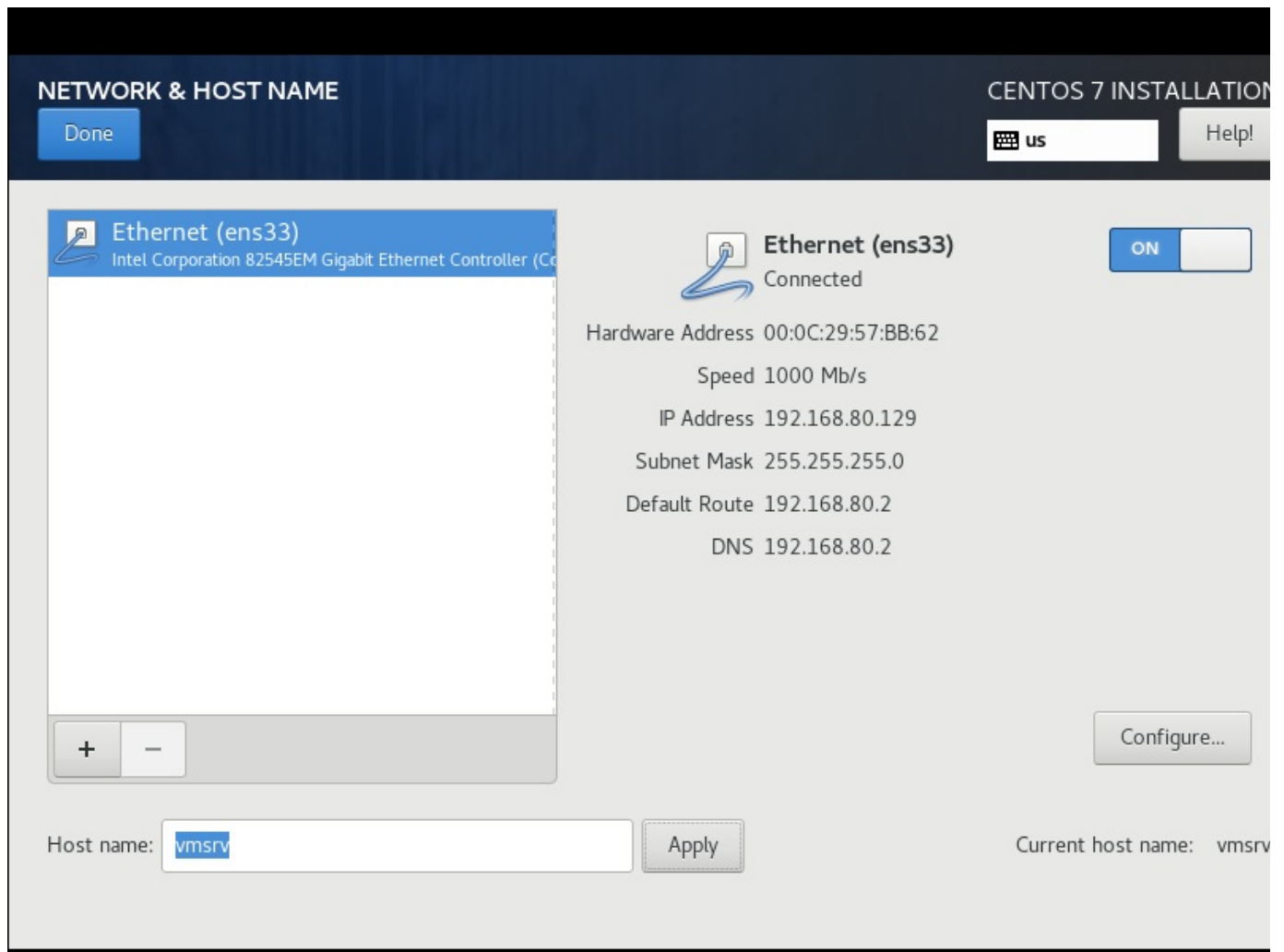


6. 在 NETWORK & HOST NAME 界面，配置网络和主机名。

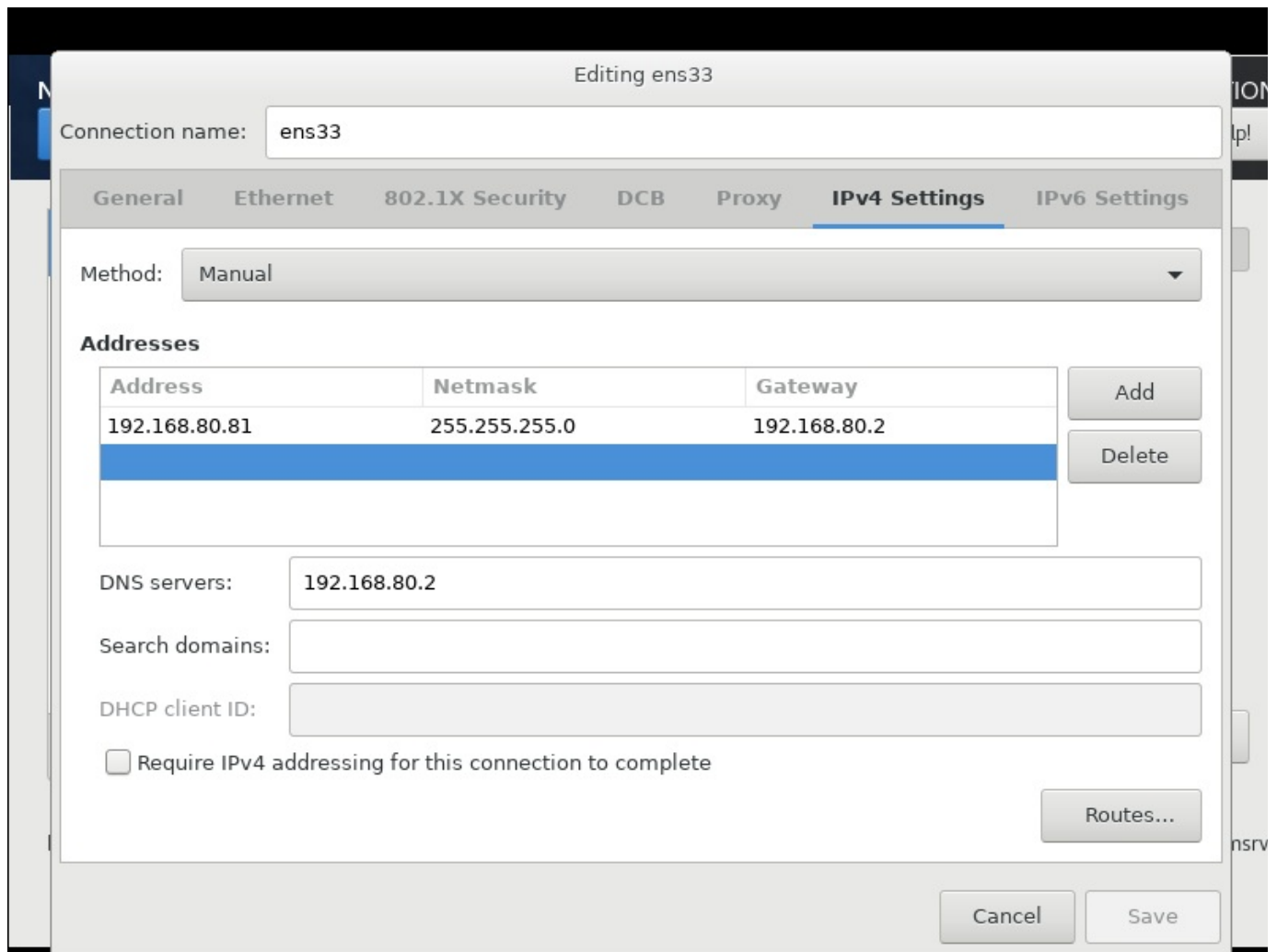
将 Host name 设置成 vmsrv，点击 Apply 生效：



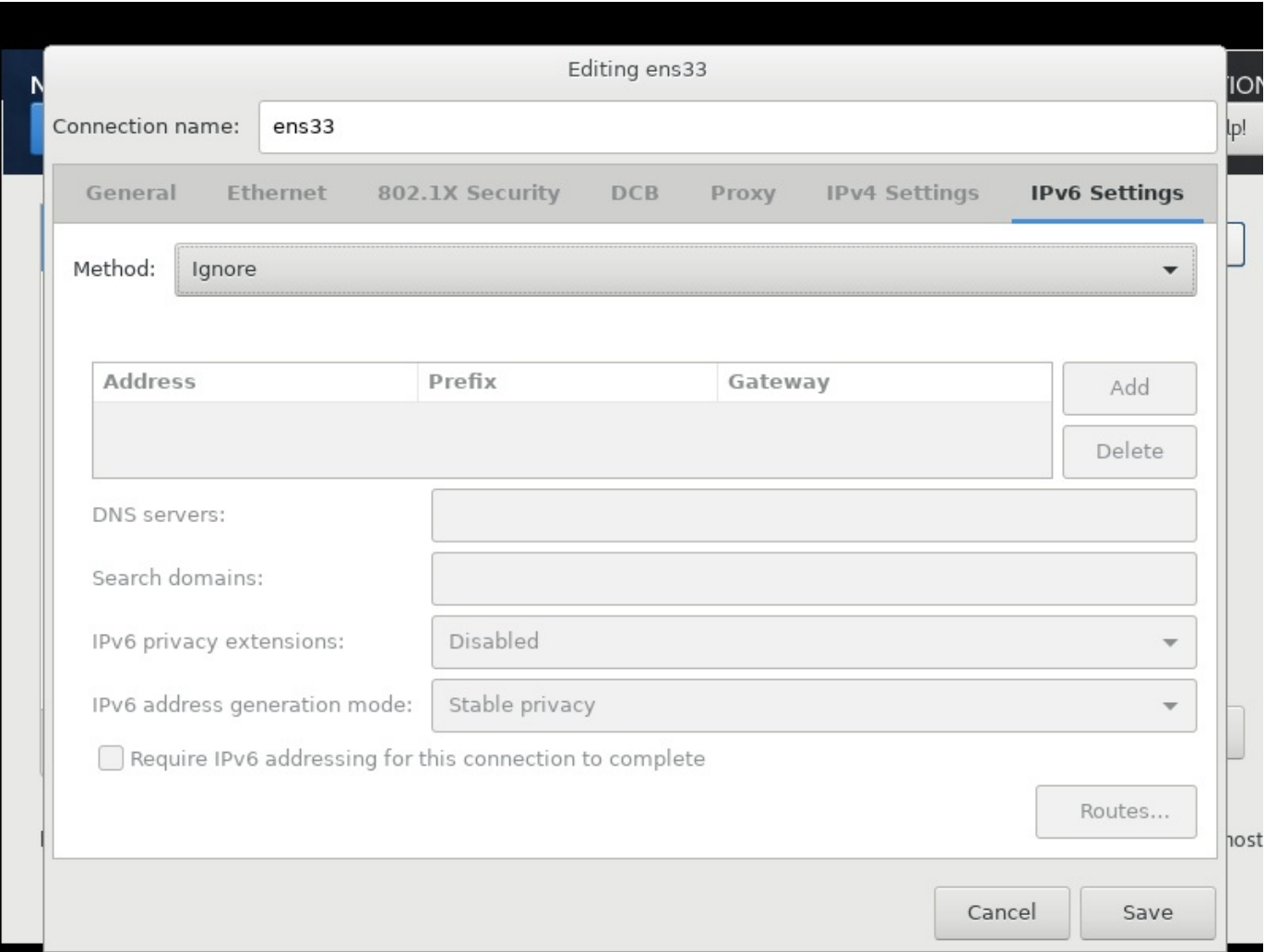
启用 Ethernet(ens33) :



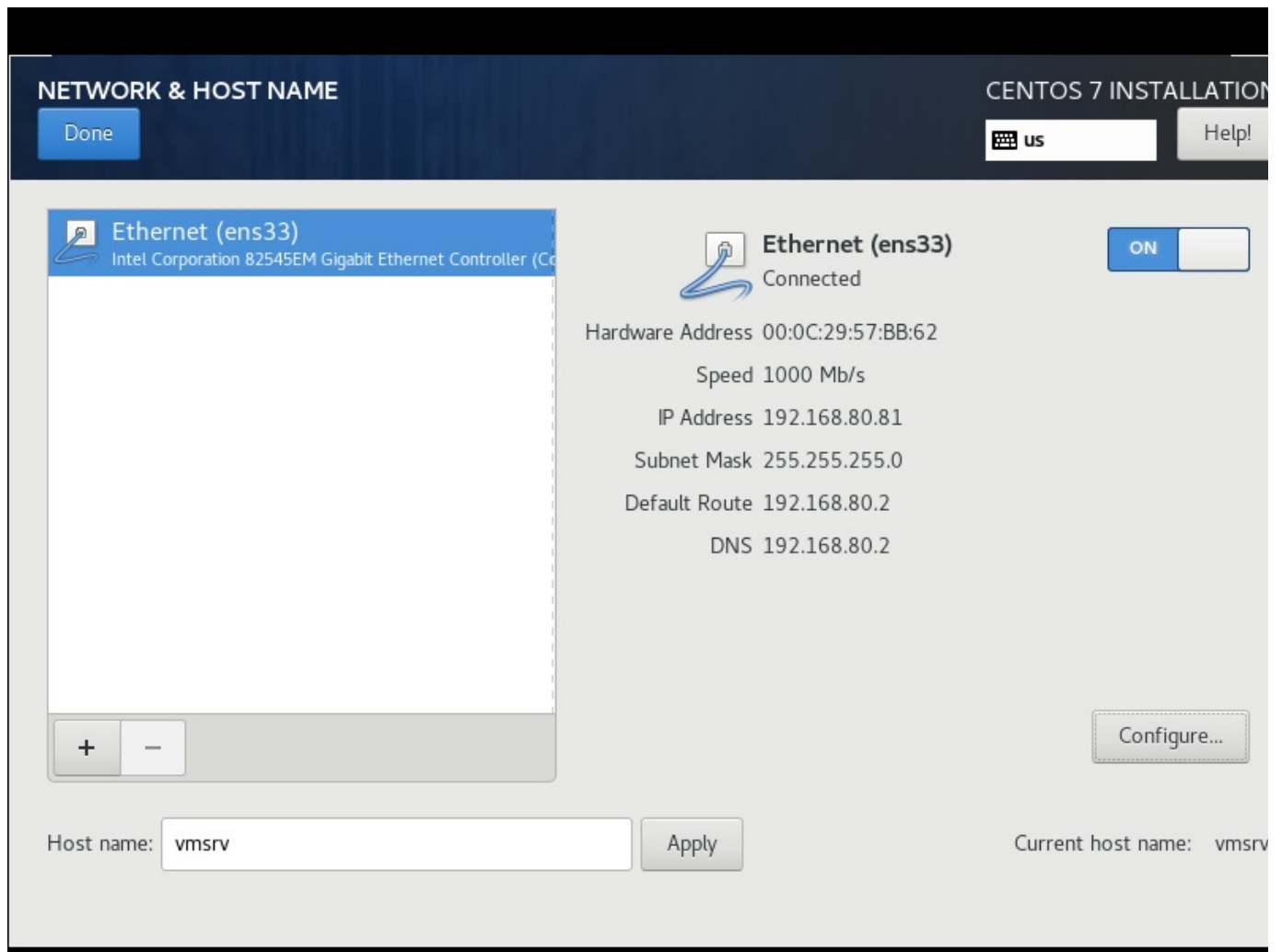
点击 Configure 手动配置网络参数。在 IPv4 Settings 对话框，手动设置 IP 地址（为方便学员交流，开发环境统一设置为：192.168.80.81）：



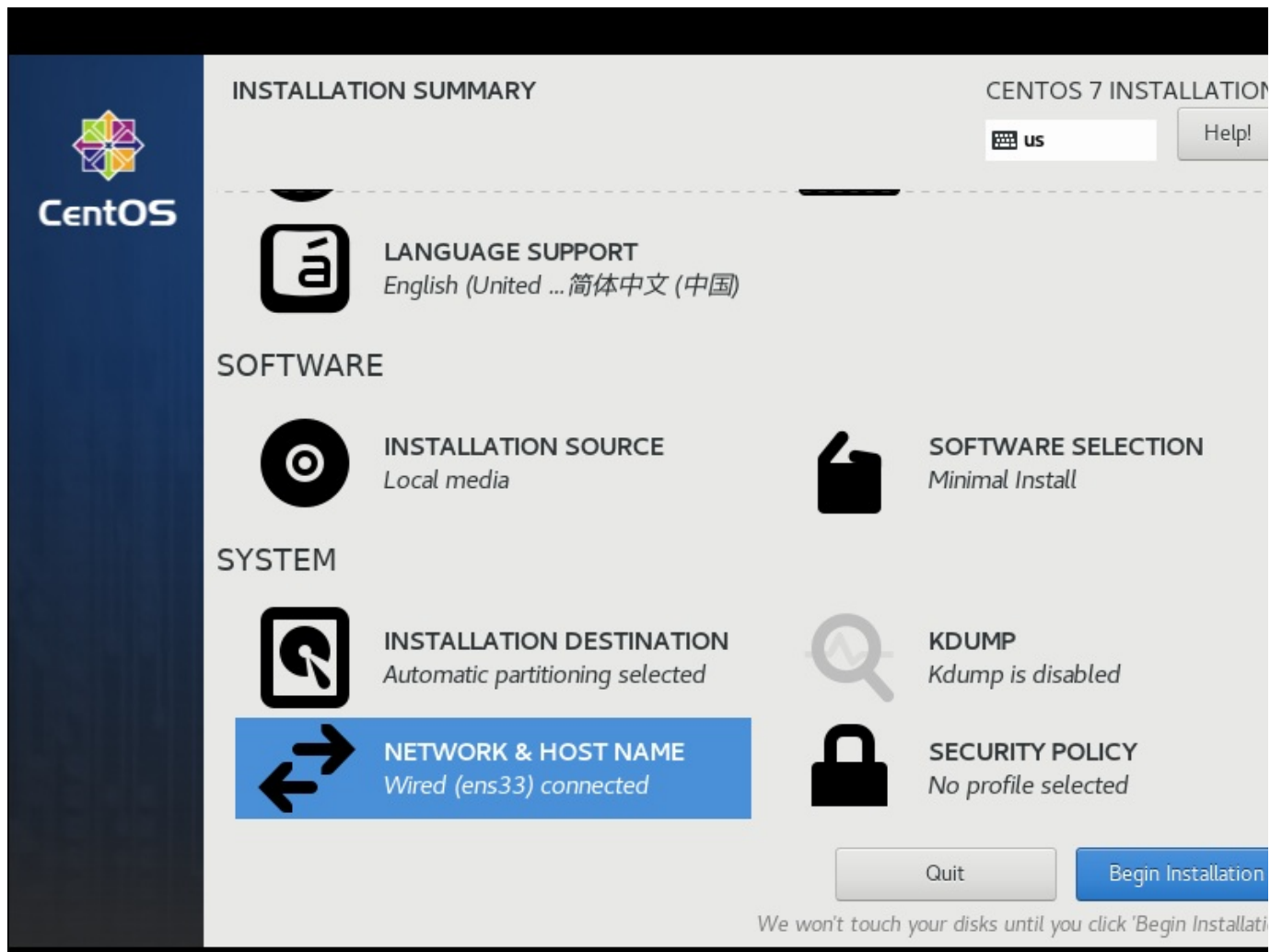
在 IPv6 Settings 对话框，将 Method 设置为 Ignore，禁用 IPv6：



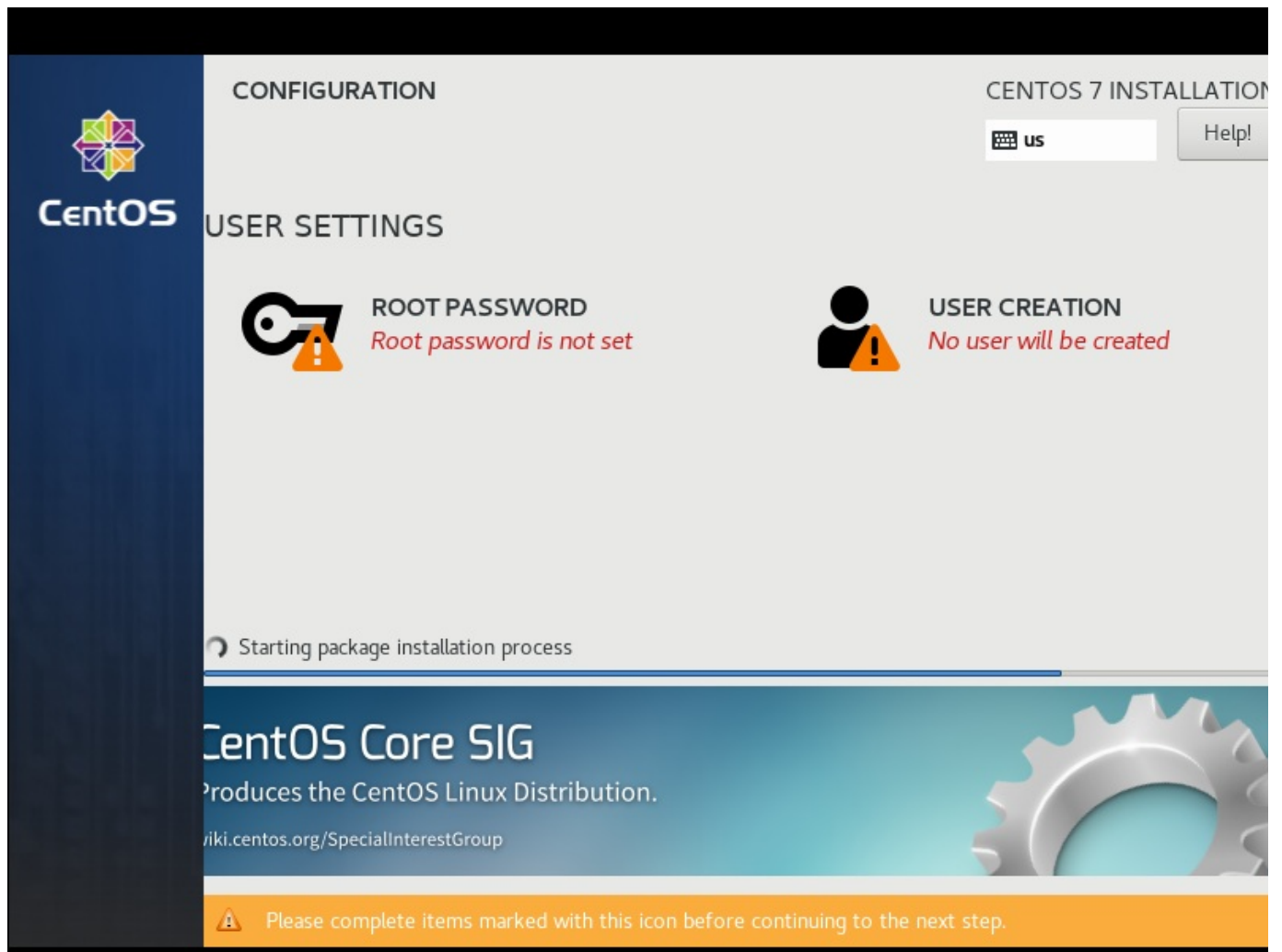
回到 NETWORK & HOST NAME 界面：



7. 在 INSTALLATION SUMMARY 界面，点击右下角的 Begin Installation 开始安装。



8. 进入 CONFIGURATION 界面，系统提示正在安装。



9. 在 ROOT PASSWORD 界面，输入 root 账号的密码：admin123（该文档为 Java Developer 系列，仅供开发参考。千万千万千万不要将你的生产环境也配置成此密码，否则你就等着被删库跑路吧），点击 Done 确定（一定要连续点击两次）



ROOT PASSWORD

CENTOS 7 INSTALLATION

Done

us

Help

The root account is used for administering the system. Enter a password for the root user.

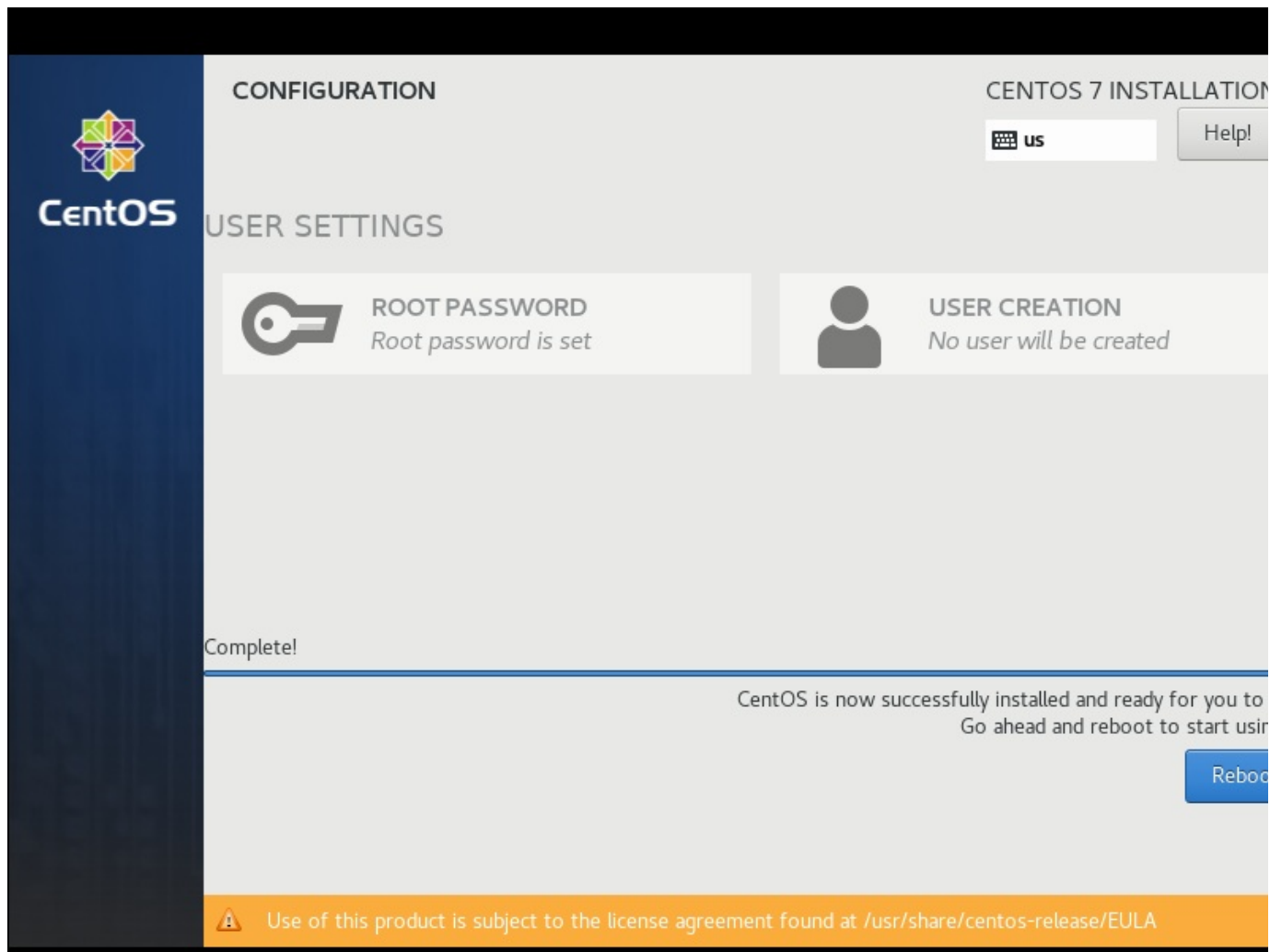
Root Password:

Confirm:

Weak

The password you have provided is weak: The password fails the dictionary check - it is based on a dictionary word You will have to press Done twice to confirm it..

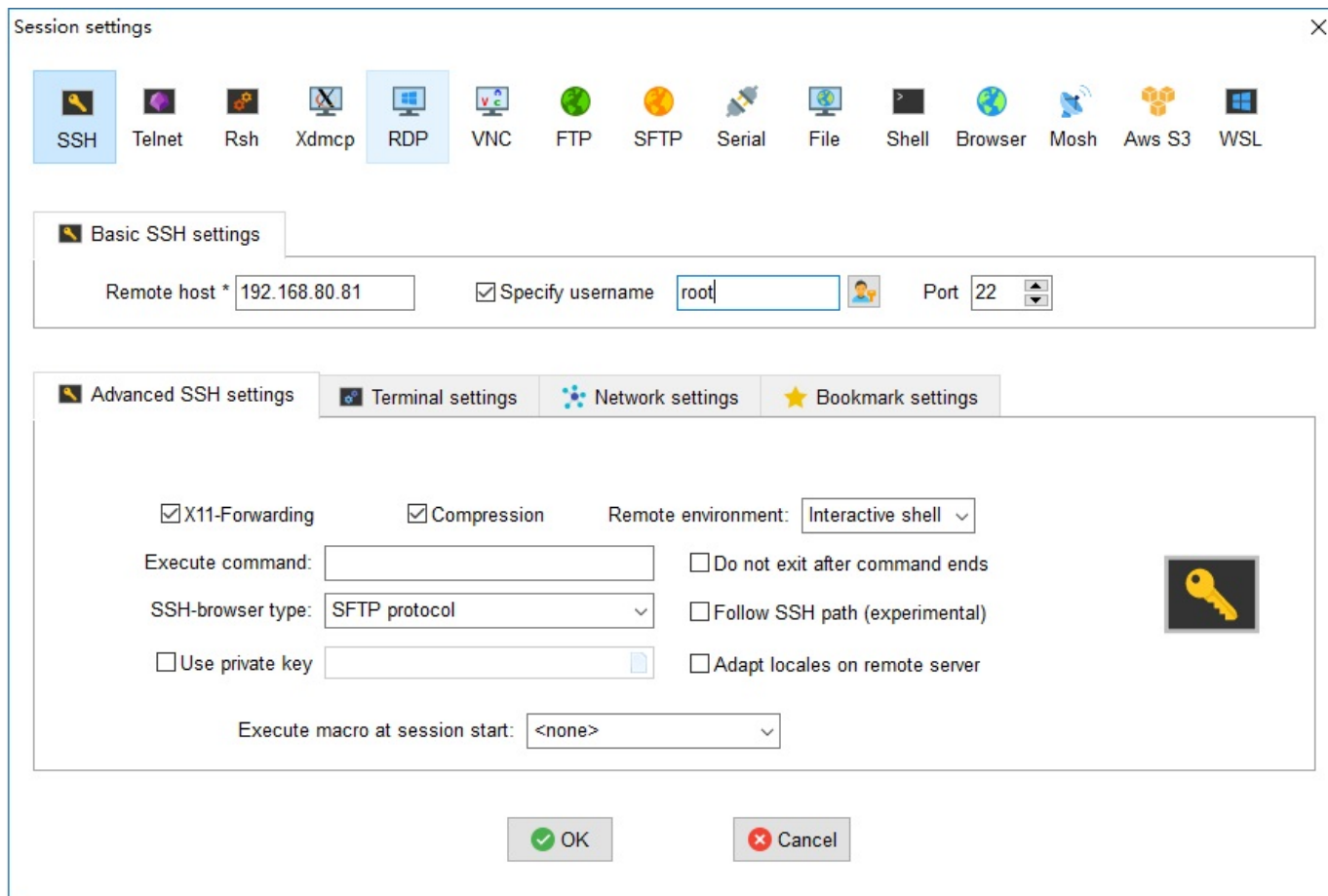
10. 回到 CONFIGURATION 界面，耐心等待（由于是最小化安装，一般不超过 10 分钟），直到安装进度显示 Complete，点击 Reboot 重启系统。



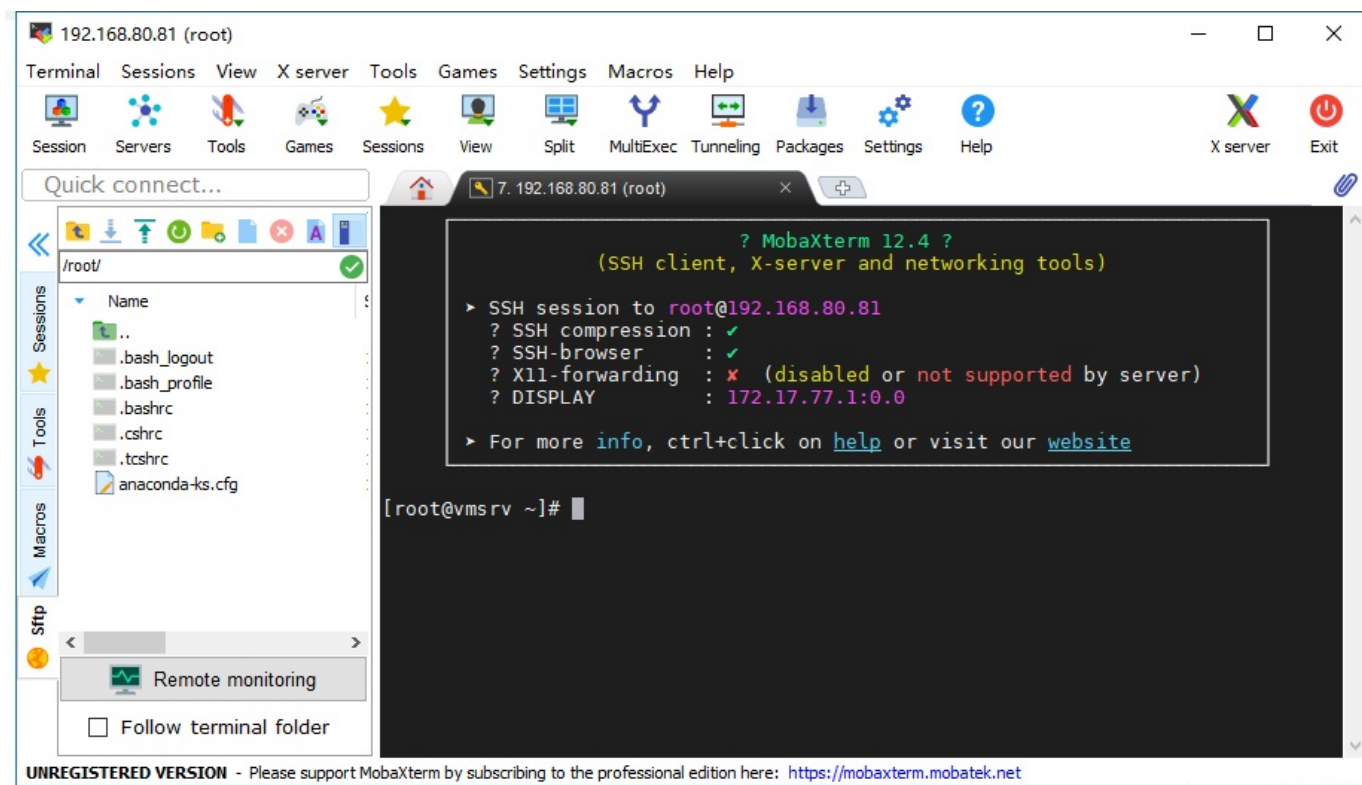
11. 耐心等待（一般不超过 10 秒钟），重启成功，系统显示登录对话框。

```
CentOS Linux 7 (Core)  
Kernel 3.10.0-1062.el7.x86_64 on an x86_64  
  
vmsrv login:
```

12. 运行 MobaXterm , 新建一个 SSH 会话 , 使用 root 账号登录。



13. 打开会话，显示登录成功。在此，恭喜您，安装完成。



## 配置 CentOS

## 关闭防火墙

重要的事情说三遍，本系列是 Developer 系列，仅供开发人员参考，关闭防火墙只是为了方便，生产环境千万别这么干。

```
# systemctl stop firewalld
# systemctl disable firewalld
Removed symlink /etc/systemd/system/multi-user.target.wants/firewalld.service.
Removed symlink /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service.
```

## 配置国内 yum 源

这里我们使用 [清华大学开源软件镜像站](#)，感觉上华为的镜像不太稳定，经常出现超时的问题。

1. 打开 [CentOS 镜像使用帮助](#)。
2. 根据建议，首先备份 CentOS-Base.repo。

```
# mv /etc/yum.repos.d/CentOS-Base.repo /etc/yum.repos.d/CentOS-Base.repo.bak
```

3. 编辑CentOS-Base.repo， `vi /etc/yum.repos.d/CentOS-Base.repo`，写入以下内容：

```
# CentOS-Base.repo
#
# The mirror system uses the connecting IP address of the client and the
# update status of each mirror to pick mirrors that are updated to and
# geographically close to the client. You should use this for CentOS updates
# unless you are manually picking other mirrors.
#
# If the mirrorlist= does not work for you, as a fall back you can try the
# remarked out baseurl= line instead.
#
#

[base]
name=CentOS-$releasever - Base
baseurl=https://mirrors.tuna.tsinghua.edu.cn/centos/$releasever/os/$basearch/
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch
#&repo=os
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

#released updates
[updates]
```

```

name=CentOS-$releasever - Updates
baseurl=https://mirrors.tuna.tsinghua.edu.cn/centos/$releasever/updates/$basearch/
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch
&repo=updates
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

#additional packages that may be useful
[extras]
name=CentOS-$releasever - Extras
baseurl=https://mirrors.tuna.tsinghua.edu.cn/centos/$releasever/extras/$basearch/
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch
&repo=extras
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

#additional packages that extend functionality of existing packages
[centosplus]
name=CentOS-$releasever - Plus
baseurl=https://mirrors.tuna.tsinghua.edu.cn/centos/$releasever/centosplus/$basearch/
#mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch
&repo=centosplus
gpgcheck=1
enabled=0
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

```

#### 4. 安装 EPEL 源：

```
# yum install epel-release
```

#### 5. 更新软件包缓存：

```

# yum clean all
Loaded plugins: fastestmirror
Cleaning repos: base extras updates
# yum makecache
Loaded plugins: fastestmirror
Determining fastest mirrors
base
| 3.6 kB 00:00:00
extras
| 2.9 kB 00:00:00
updates

```

```

(1/10): base/7/x86_64/group_gz                | 2.9 kB  00:00:00
(2/10): base/7/x86_64/filelists_db            | 165 kB  00:00:03
(3/10): extras/7/x86_64/filelists_db          | 7.3 MB  00:00:05
(4/10): extras/7/x86_64/other_db              | 207 kB  00:00:01
(5/10): base/7/x86_64/other_db                | 100 kB  00:00:00
(6/10): extras/7/x86_64/primary_db            | 2.6 MB  00:00:01
(7/10): base/7/x86_64/primary_db              | 153 kB  00:00:03
(8/10): updates/7/x86_64/filelists_db        | 6.0 MB  00:00:06
(9/10): updates/7/x86_64/other_db            | 3.3 MB  00:00:04
(10/10): updates/7/x86_64/primary_db         | 368 kB  00:00:00
Metadata Cache Created                       | 5.9 MB  00:00:06

```

6. 更新软件包，这个根据你的网络情况，需要耐心等待一会：

```
# yum update -y
~~shu
```

## 安装附件软件

### 1. 安装常用软件包

```
# yum install -y wget vim net-tools.x86_64 telnet zip unzip deltarpm jq
```

## 创建用户和组

1. 创建 `oper` 组：

```
# groupadd oper
```

2. 创建用户 `lemon`，将默认密码设置为 `lemon123`：

```
# useradd -g oper lemon  
# passwd lemon
```



# 安装 Docker

- 安装 Docker
  - 安装 Docker
  - 配置 Docker
  - Docker 实例
    - Oracle Database 11g XE
    - Oracle Database 12c

# 安装 Docker

## 安装 Docker

1. 在 Linux 机器上打开一个新的 Shell。
2. 使用 wget 从 <https://get.docker.com> 获取并运行 Docker 安装脚本，然后采用 Shell 中管道（pipe）的方式来执行这个脚本。

```
# wget -qO- https://get.docker.com/ | sh
```

3. 启动 Docker 并将其设置为开机启动。

```
# systemctl start docker
# systemctl enable docker
```

4. 最好通过非 root 用户来使用 Docker。这时需要添加非 root 用户到本地 Docker Unix 组当中。下面的命令展示了如何把名为 lemon 的用户添加到 Docker 组中，以及如何确认操作是否执行成功。请读者自行使用系统中的有效用户。

```
# sudo usermod -aG docker lemon
# cat /etc/group | grep docker
```

如果读者当前登录用户就是要添加到 Docker 组中的用户的话，则需要重新登录，组权限设置才会生效。

5. 恭喜！Docker 已经在读者的 Linux 机器上安装成功。运行下面命令来确认安装结果。

```
$ docker --version
Docker version 19.03.5, build 633a0ea
$ docker system info
```

```

Client:
  Debug Mode: false
Server:
  Containers: 0
    Running: 0
    Paused: 0
    Stopped: 0
  Images: 0
  Server Version: 19.03.5
  Storage Driver: overlay2
    Backing Filesystem: xfs
    Supports d_type: true
    Native Overlay Diff: true
  Logging Driver: json-file
  Cgroup Driver: cgroupfs
  Plugins:
    Volume: local
    Network: bridge host ipvlan macvlan null overlay
    Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
  Swarm: inactive
  Runtimes: runc
  Default Runtime: runc
  Init Binary: docker-init
  containerd version: b34a5c8af56e510852c35414db4c1f4fa6172339
  runc version: 3e425f80a8c931f88e6d94a8c831b9d5aa481657
  init version: fec3683
  Security Options:
    seccomp
    Profile: default
  Kernel Version: 3.10.0-1062.9.1.el7.x86_64
  Operating System: CentOS Linux 7 (Core)
  OSType: linux
  Architecture: x86_64
  CPUs: 2
  Total Memory: 3.84GiB
  Name: vmsrv
  ID: EORB:X7ZF:HFAH:76I4:MHT5:NCLY:4GCL:67GB:VBKI:ERPB:7NTA:5AIS
  Docker Root Dir: /var/lib/docker
  Debug Mode: false
  Registry: https://index.docker.io/v1/
  Labels:
  Experimental: false
  Insecure Registries:
    127.0.0.0/8
  Live Restore Enabled: false

WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled

```

## 配置 Docker

### 1. 配置国内镜像：

```
# cat <<EOF > /etc/docker/daemon.json
{
  "registry-mirrors": [
    "https://registry.docker-cn.com",
    "http://hub-mirror.c.163.com",
    "https://mirror.ccs.tencentyun.com",
    "https://tzddjd6l.mirror.aliyuncs.com"
  ],
  "insecure-registries": [],
  "debug": true,
  "experimental": false
}
EOF
```

### 2. 重启 Docker：

```
# systemctl restart docker
```

## Docker 实例

### Oracle Database 11g XE

#### 1. 拉取镜像。

```
$ docker pull oracleinanutshell/oracle-xe-11g
```

#### 2. 创建并运行容器。

```
$ docker run --name oracle11g -d -p 1521:1521 -p 8080:8080 -e ORACLE_ALLOW_REMOTE=true oracleinanutshell/oracle-xe-11g
```

- -d：已守护进程的方式在后台运行容器。

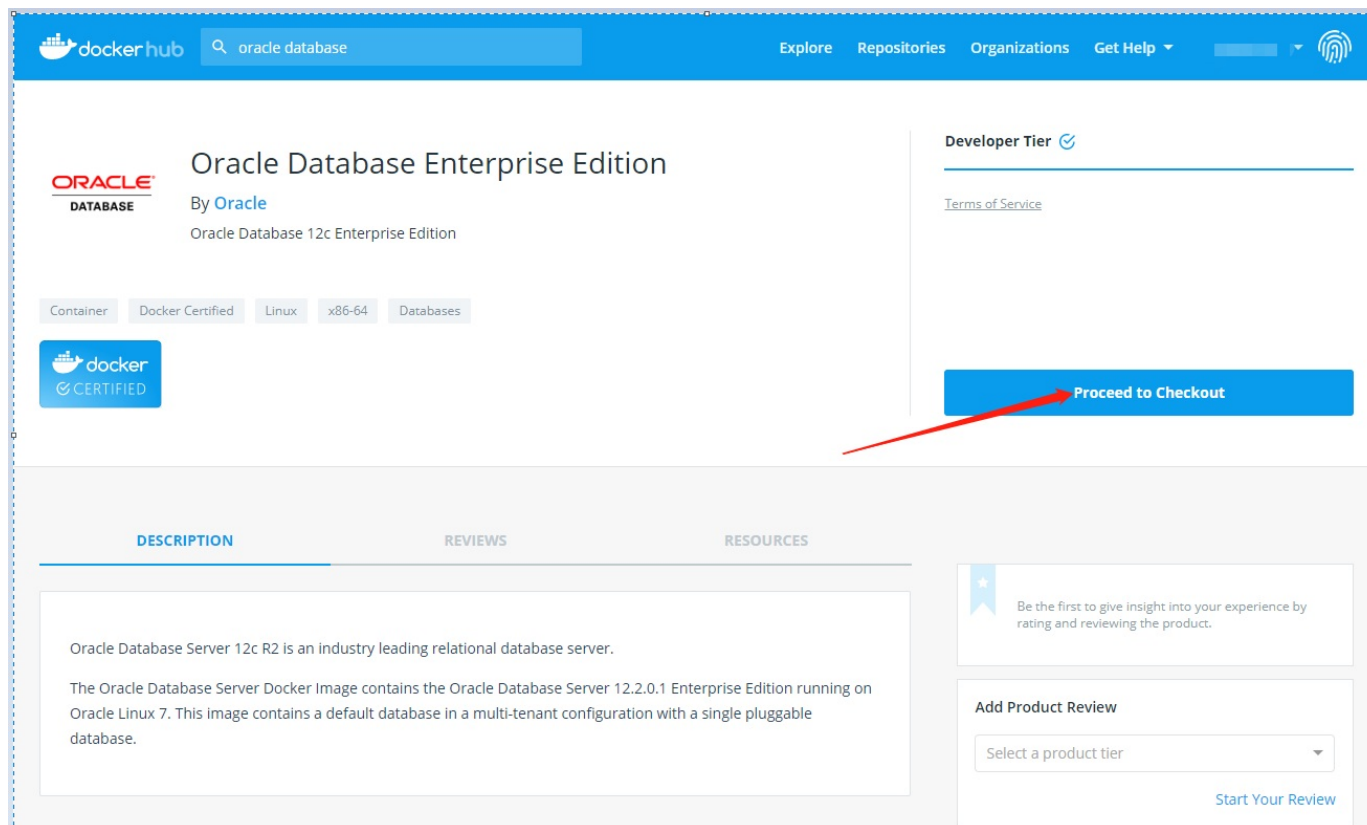
#### 3. 修改 SYSTEM 密码，初始密码：oracle。

```
$ docker exec -it oracle11g bash
$ sqlplus
Enter user-name: system
Enter password: oracle
SQL> alter user system identified by admin123;
```

```
SQL> exit;  
$ exit
```

## Oracle Database 12c

1. 在 Docker Hub 中查看 Oracle Database 的安装说明。
2. 登录 Docker Hub，搜索到 Oracle Database 镜像，并点击 Proceed to Checkout。



3. 填写联系信息，并同意协议，点击 Get Content。

4. 系统返回 Setup Instructions。

5. 拉取镜像。

```
$ docker login
$ docker pull store/oracle/database-enterprise:12.2.0.1
```

6. 创建并运行容器。

```
$ docker run -d -it --name oracle12c -p 1521:1521 -p 5500:5500 -e DB_SID=ORCLCDB -e DB_MEMORY=2G store/oracle/database-enterprise:12.2.0.1
```

7. 注意：启动较慢，请耐心等待。使用 `docker ps` 命令查看容器状态，状态变成 healthy 即代表启动完成。

```
$ docker ps
```

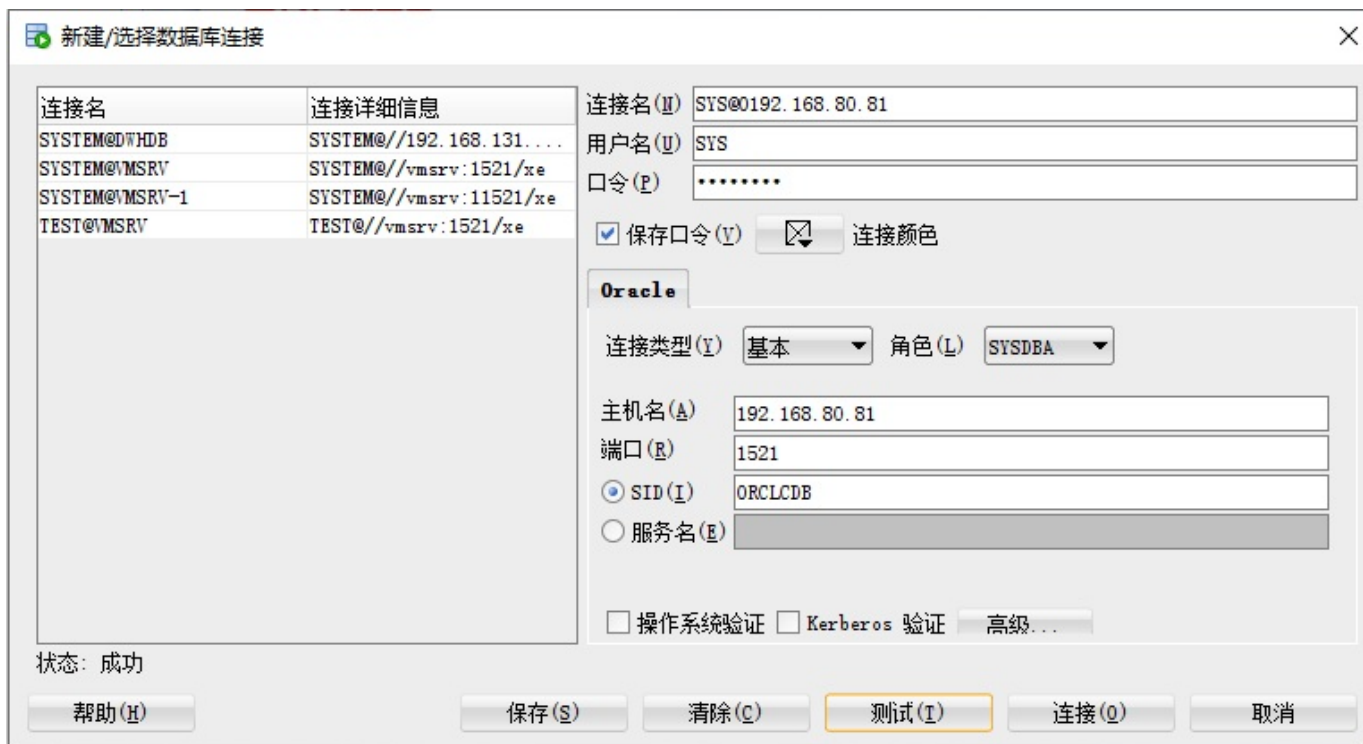
CONTAINER ID	IMAGE	STATUS	COMMAND
CREATED		NAMES	PORTS
11a27f8db0e9	store/oracle/database-enterprise:12.2.0.1	Up 9 minutes (healthy)	"/bin/sh -c '"/bin/ba... 21/tcp, 0.0.0.0:5500->5500/tcp

8. 修改 SYS 密码。

```
$ docker exec -it oracle12c bash
[oracle@11a27f8db0e9 /]$ sqlplus /nolog
SQL> conn / as sysdba
SQL> alter user sys identified by admin123;
```

```
SQL> exit;
$ exit
```

## 9. 使用 Oracle SQL Developer 创建连接，验证安装是否成功。



## 10. 启动数据库的归档模式。

```
$ sqlplus / as sysdba
SQL> shutdown immediate;
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> startup mount;
ORACLE instance started.
Total System Global Area 671088640 bytes
Fixed Size 8796384 bytes
Variable Size 213911328 bytes
Database Buffers 440401920 bytes
Redo Buffers 7979008 bytes
Database mounted.
SQL> alter database archivelog;
Database altered.
SQL> alter database open;
Database altered.
SQL> exit;
```

## 11. 启用补充日志。

```
$ sqlplus / as sysdba
SQL> alter database add supplemental log data (all) columns;
Database altered.
SQL> exit;
```

## 12. 创建 Oracle 用户。

```
create role logmnr_role;
grant create session to logmnr_role;
grant execute_catalog_role,select any transaction ,select any dictionary t
o logmnr_role;
create user kminer identified by kminerpass;
grant logmnr_role to kminer;
alter user kminer quota unlimited on users;

create role c##logmnr_role;
grant create session to c##logmnr_role;
grant execute_catalog_role,select any transaction ,select any dictionary,l
ogmining to c##logmnr_role;
create user c##kminer identified by kminerpass;
grant c##logmnr_role to c##kminer;
alter user c##kminer quota unlimited on users set container_data = all cont
ainer = current;
```

# 安装 Oracle JDK 1.8

- [安装 Oracle JDK 1.8.0\\_221 64 位版](#)
  - [安装 JDK](#)

## 安装 Oracle JDK 1.8.0\_221 64 位版

### 安装 JDK

1. 从 Oracle 官网下载二进制包 [jdk-8u221-linux-x64.tar.gz](#) , 并上传到 /opt 目录。

注意：Oracle 已经修改了 8u211 及其之后版本的许可证。

2. 解压缩：

```
# cd /opt
# tar -zxvf jdk-8u221-linux-x64.tar.gz
```

修改目录权限：

```
# chown -R lemon:oper /opt/jdk1.8.0_221
```

3. 修改环境变量：

```
# vim /etc/profile
```

将以下内容添加到 /etc/profile

```
export JAVA_HOME=/opt/jdk1.8.0_221
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
export PATH=$PATH:$JAVA_HOME/bin:
```

使环境变量马上生效：

```
# source /etc/profile
```

4. 验证 Java：

```
# java -version
java version "1.8.0_221"
```



```
Java(TM) SE Runtime Environment (build 1.8.0_221-b11)  
Java HotSpot(TM) 64-Bit Server VM (build 25.221-b11, mixed mode)
```

# 附录 B

---

[Chrony](#)

[MobaXterm](#)

[Kafka Tool](#)

# Chrony

- [Chrony](#)
  - [安装 Chrony](#)
  - [常用命令](#)
    - [检查时钟是否同步](#)
    - [手动调整系统时钟](#)
- [参考资料](#)

# Chrony

## 安装 Chrony

1. 安装 chrony（用于自动同步系统时钟）：

```
# yum erase 'ntp*'
# yum install -y chrony
```

ntpd 和 chrony 可能会导致冲突，不能同时安装

2. 配置 ntp 服务器：

```
# cp /etc/chrony.conf /etc/chrony.conf.bak
# vim /etc/chrony.conf
```

删除默认的 ntp 服务器，改为阿里的 ntp 服务器：

```
server ntp1.aliyun.com iburst
server ntp2.aliyun.com iburst
server ntp3.aliyun.com iburst
server ntp4.aliyun.com iburst
server ntp5.aliyun.com iburst
server ntp6.aliyun.com iburst
server ntp7.aliyun.com iburst
```

## 3. 启动 chronyd :

```
# service chronyd start
Redirecting to /bin/systemctl start chronyd.service
# chkconfig chronyd on
Note: Forwarding request to 'systemctl enable chronyd.service'.
```

## 4. 验证 chronyd :

```
# chronyc sources -v
210 Number of sources = 2

.-- Source mode '^' = server, '=' = peer, '#' = local clock.
/ .- Source state '*' = current synced, '+' = combined , '-' = not combined
/
| /  '?' = unreachable, 'x' = time may be in error, '~' = time too variable
.
||                                     .- xxxx [ yyyy ] +/- zzz
Z
||      Reachability register (octal) -.      | xxxx = adjusted offset
/
||      Log2(Polling interval) --.      |      | yyyy = measured offset
/
||                                     \      |      | zzzz = estimated error
.
||                                     |      |      | \
MS Name/IP address             Stratum Poll Reach LastRx Last sample
=====
=====
^* 120.25.115.20                2    6    17     7    -27ms[ -137ms] +/-  6
17ms
^+ 203.107.6.88                 2    6    17     9    -32ms[ -32ms] +/-  6
13ms
```

在返回的输出中，^\* 表示首选的时间源。

```
# chronyc tracking
Reference ID      : 78197314 (120.25.115.20)
Stratum          : 3
Ref time (UTC)   : Sat Jan 11 12:39:01 2020
System time      : 0.000672388 seconds fast of NTP time
Last offset      : -0.109685801 seconds
RMS offset       : 0.109685801 seconds
Frequency        : 0.000 ppm slow
Residual freq    : -36916.840 ppm
Skew             : 1000000.000 ppm
```

```

Root delay      : 1.225557446 seconds
Root dispersion : 40.070934296 seconds
Update interval : 2.9 seconds
Leap status     : Normal

```

## 常用命令

### 检查时钟是否同步

chronyc tracking 用于显示系统时间信息。

```

$ chronyc tracking
Reference ID      : 78197314 (120.25.115.20)
Stratum          : 3
Ref time (UTC)   : Wed Jan 15 01:03:05 2020
System time      : 0.000222231 seconds fast of NTP time
Last offset      : +0.000310285 seconds
RMS offset       : 4079.027587891 seconds
Frequency        : 6.416 ppm slow
Residual freq    : +0.049 ppm
Skew             : 4.128 ppm
Root delay       : 0.036260299 seconds
Root dispersion  : 0.002073948 seconds
Update interval  : 129.9 seconds
Leap status      : Normal

```

Ref time (UTC) : 来自参考源的 UTC 时间。

System time : 在正常的操作中，chronyd 绝不会步进系统时钟，因为时钟的跳跃可能会对某些应用程序产生不利的影响。取而代之的是，通过稍微加快或降低系统时钟的速度来纠正系统时钟中的错误，直到错误被消除，然后恢复系统时钟的正常速度。在开发环境使用 VMWare 虚拟机的暂停和恢复功能时将很容易重现这个问题，如果你要立即步进同步时钟，请使用 chronyc makestep 命令。

### 手动调整系统时钟

要立即步进同步系统时钟，请使用以下命令：

```

# chronyc makestep
200 OK

```

注：如果步进同步系统时间没有立即生效，你可以尝试重启服务器。

---

## 参考资料

- [USING CHRONY](#)

# MobaXterm

- [MobaXterm 12.4](#)
  - [使用 root 用户登录系统](#)

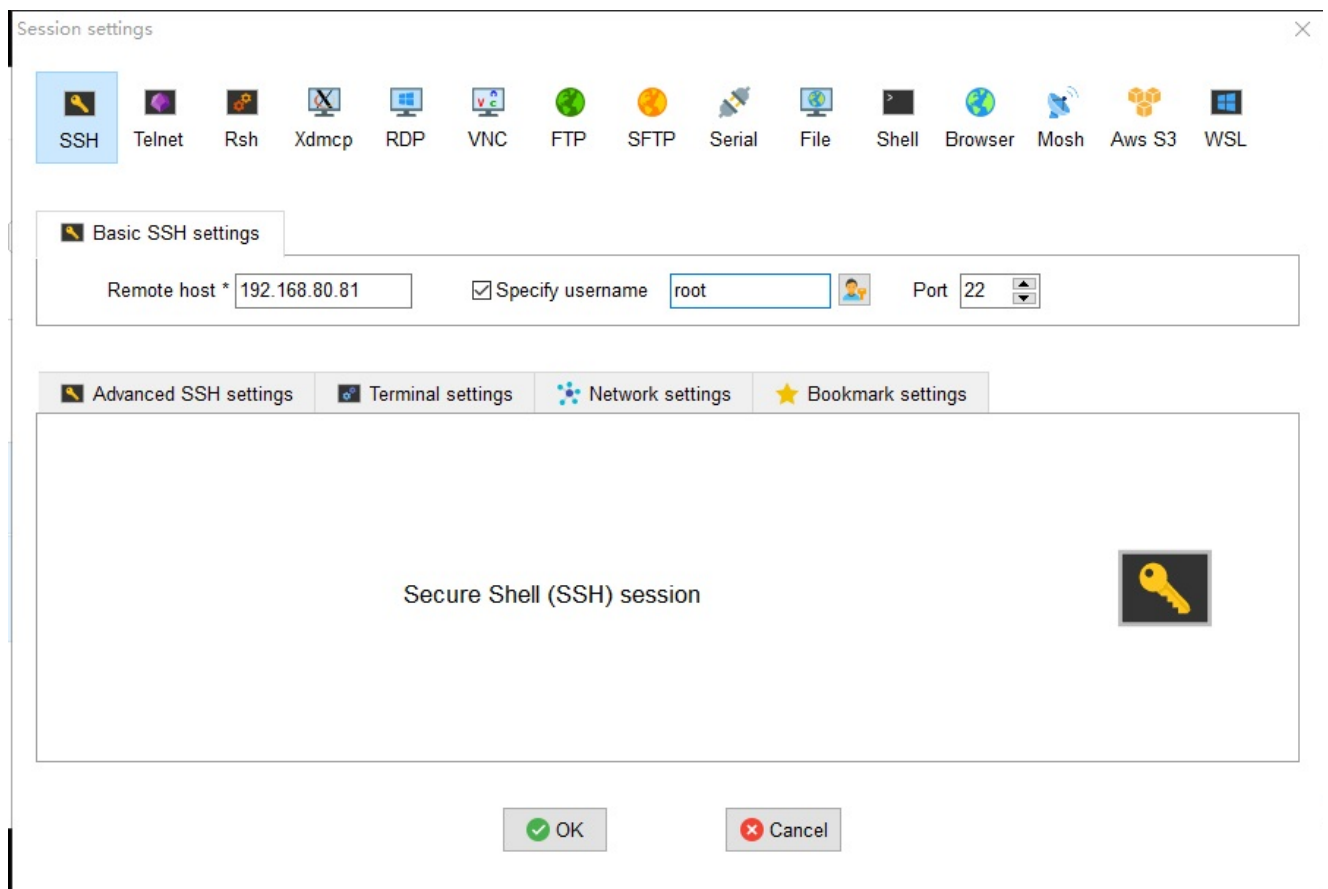
## MobaXterm 12.4

### 使用 root 用户登录系统

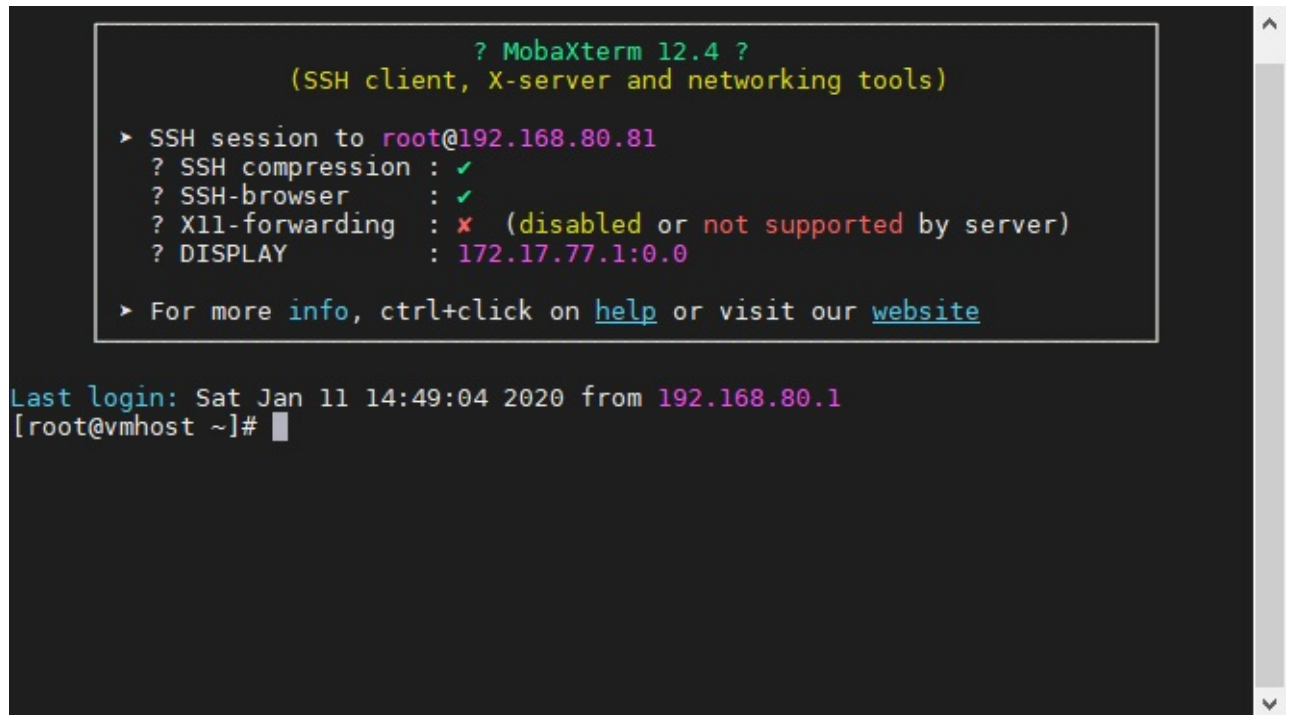
在安装 Kafka 软件之前，您必须以 root 用户身份完成一些任务。要以 root 用户身份登录，请完成以下过程：

- 使用 MobaXterm 登录到远程云主机/本地虚拟机。如果你还没有 MobaXterm，你可以从 [这里](#) 下载，它非常方便使用。

1. 启动 MobaXterm 新建一个 SSH 远程会话。



2. 登录成功，你应该能看到一个 # 提示符，在后面的内容中，我们会以 # 表示 root 用户操作。



A screenshot of the MobaXterm terminal window. The window has a dark background with a light gray border. Inside, a white box contains the following text:   
? MobaXterm 12.4 ?  
(SSH client, X-server and networking tools)  
➤ SSH session to root@192.168.80.81  
? SSH compression : ✓  
? SSH-browser : ✓  
? X11-forwarding : ✗ (disabled or not supported by server)  
? DISPLAY : 172.17.77.1:0.0  
➤ For more info, ctrl+click on help or visit our website  
Below the white box, the terminal shows: Last login: Sat Jan 11 14:49:04 2020 from 192.168.80.1  
[root@vmhost ~]#

```
                ? MobaXterm 12.4 ?  
            (SSH client, X-server and networking tools)  
  
➤ SSH session to root@192.168.80.81  
? SSH compression : ✓  
? SSH-browser      : ✓  
? X11-forwarding   : ✗ (disabled or not supported by server)  
? DISPLAY          : 172.17.77.1:0.0  
  
➤ For more info, ctrl+click on help or visit our website  
  
Last login: Sat Jan 11 14:49:04 2020 from 192.168.80.1  
[root@vmhost ~]#
```



# Kafka Tool

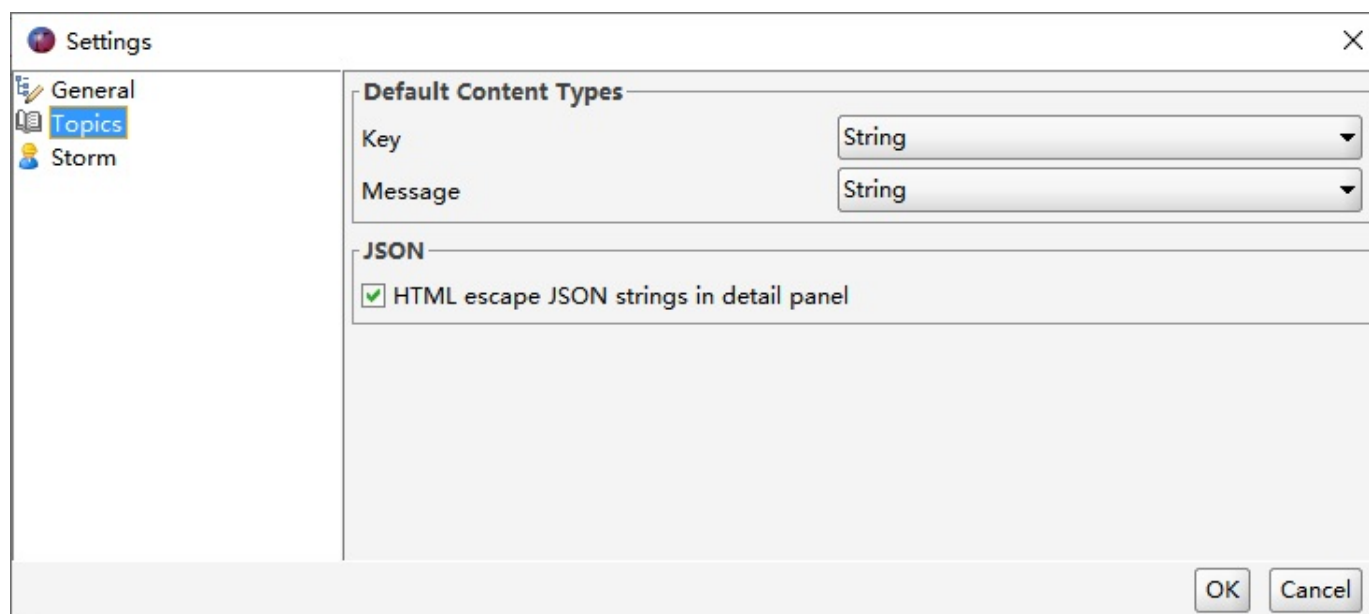
- [Kafka Tool 2.0.6](#)

## Kafka Tool 2.0.6


1. 修改 hosts 文件，请将以下内容添加到 hosts 文件：

```
192.168.80.81    vmsrv
```

2. 启动 Kafka Tool，在 Settings 窗口，将 Topics 的 Default Content Types 改为 String：



3. 新建连接：

 Add Cluster

General

Cluster name

kafka-2.3.1@192.168.80.81

Kafka Cluster Version

2.3

Zookeeper Host

192.168.80.81

Ping

Zookeeper Port

2181

chroot path

/

Broker Security

Type

Plaintext

Advanced

Bootstrap servers

192.168.80.81:9092

You can optionally specify broker endpoint(s) to use instead of brokers discovered in Zookeeper. For example: broker1:90...

SASL Mechanism

Enter a value to override the default sasl.mechanism value [GSSAPI]

Offset Topic

☐ Use background thread to read from \_\_consumer\_offsets topic

If disabled, viewing the consumer offsets will be considerably slower

Test

Add

Cancel