麒麟V11服务器系统安装ETCD集群

环境介绍

硬件环境：`3C6000`

操作系统版本：`Kylin-server-v11`

内核版本：`6.6.0-32.7.v2505.ky11.loongarch64`

安装Golang

1. Golang官网下载相关版本

```
[root@localhost ~]#wget  https://dl.google.com/go/go1.25.10.linux-
loong64.tar.gz
```

2. 解压并安装

```
[root@localhost ~]#tar -C /usr/local -xzf go1.22.0.linux-loong64.tar.gz
```

3. 添加环境变量

```
cat >> /etc/profile << 'EOF'
export GOROOT=/usr/local/go
export GOPATH=/root/go
export PATH=$PATH:$GOROOT/bin:$GOPATH/bin
export GO111MODULE=on
export GOPROXY=https://goproxy.cn,direct
EOF

source /etc/profile
```

4. 测试go版本

```
[root@localhost kubernetes]# go version
go version go1.25.10 linux/loong64
[root@localhost kubernetes]#
```

编译Etcd

1. 从github上下载etcd

```
[root@localhost kubernetes]#git clone -b v3.6.5
https://gitee.com/mirrors/etcd.git etcd-3.6.5
验证下载版本
[root@localhost kubernetes]#git describe --tags
```

2. 编译Etcd

```
编辑etcdmain.go源码，添加loong64架构信息
// 在函数开头找到官方架构支持列表，为其添加 loong64
func checkSupportArch() {
    // 例如，原有列表可能是 ["amd64", "arm64", ...]，现在我们手动将 loong64 加进去
    // 请注意，下述代码仅为逻辑示意，具体写法需匹配原文件的风格，可能是switch case或if语
句。
    supportedArches := map[string]bool{
        "amd64": true,
        "arm64": true,
        "loong64": true, // 将 loong64 添加到支持列表
    }
    if supportedArches[runtime.GOARCH] {
        return
    }
    // ... 其余逻辑保持不变 ...
}

执行编译
[root@localhost kubernetes]# GOTOOLCHAIN=local make build
```

etcd 项目为了保证所有开发者的编译环境都高度一致，通过几个文件"锁定"了 Go 版本。

1. `.go-version` **文件**：在 etcd 源码的根目录下，有一个名为 `.go-version` 的文件，里面就明确指定了项目所需的 Go 版本。
2. `go.mod` **中的** `toolchain` **指令**：在 `go.mod` 文件里，`toolchain` 指令通常会和 `.go-version` 文件保持一致。它的作用就是确保项目在编译时使用一个确切、固定的 Go 版本。

当你的 Go 版本高于或低于项目指定的要求时，Go 工具链会自动下载并临时切换到一个 "**最低工具链版本**" 来执行操作，以确保构建过程绝对可靠。通常情况下，系统会选择**满足** `go.mod` **文件中** `go` **版本要求的最低可用版本**。你的系统里有 1.25.10，但项目可能通过 `toolchain` 指令强制回退到了一个更旧的基准版本来进行构建。

```
# 优先使用本地安装的 Go 版本
GOTOOLCHAIN=local make build
# 或自动决定使用满足版本要求的最新本地工具链
GOTOOLCHAIN=auto make build
```

3. 验证etcd版本

```
编译完成的etcd在当前目录的bin目录下
[root@localhost etcd-3.6.5]# ls -l bin/
总计 82404
-rwxr-xr-x 1 root root 36194503  5月19日 13:25 etcd
-rwxr-xr-x 1 root root 24010507  5月19日 13:25 etcdctl
-rwxr-xr-x 1 root root 24171769  5月19日 13:25 etcdutl
[root@localhost etcd-3.6.5]#
```

```
[root@localhost etcd-3.6.5]# cp -r bin/* /usr/local/bin/
[root@localhost etcd-3.6.5]#

验证版本：
[root@localhost etcd-3.6.5]# etcd --version
etcd Version: 3.6.5
Git SHA: a0614505a
Go Version: go1.25.10
Go OS/Arch: linux/loong64
[root@localhost etcd-3.6.5]#
```

配置Etcd集群

1. 证书说明

```
                    +--------------------+
                    |     etcd 根 CA     |
                    |      (ca.crt)      |
                    +---------+----------+
                              | 签发
+--------------------------+       +-------------------------+
|    Client 证书           |       |    Server 证书          |
| (client.crt/client.key)  |       |(server.crt/server.key)  |
+--------------------------+       +-------------------------+
             | 验证                          | 验证
+--------------------------+       +-------------------------+
|     外部客户端           |       |    etcd节点间通信       |
|    (使用 client 证书)    |       | (使用 Server 证书)      |
+--------------------------+       +-------------------------+
```

- **Client证书**： 客户端（如apiserver、calico）访问etcd服务时使用的证书。
- **Server证书**： etcd服务端（供客户端访问）使用的证书。
- **Peer证书**： etcd集群内部节点之间通信使用的证书（同时具备client和server角色）。

```
实际上 只需要client证书和server证书即可，因为etcd即作为服务端也是客户端，证书太多容易搞混，只需要记住：
- clinet用于集群外部使用，比如k8s的apiserver。
- server证书用于集群之间使用，比如创建集群，etcdctl访问etcd的时候。
```

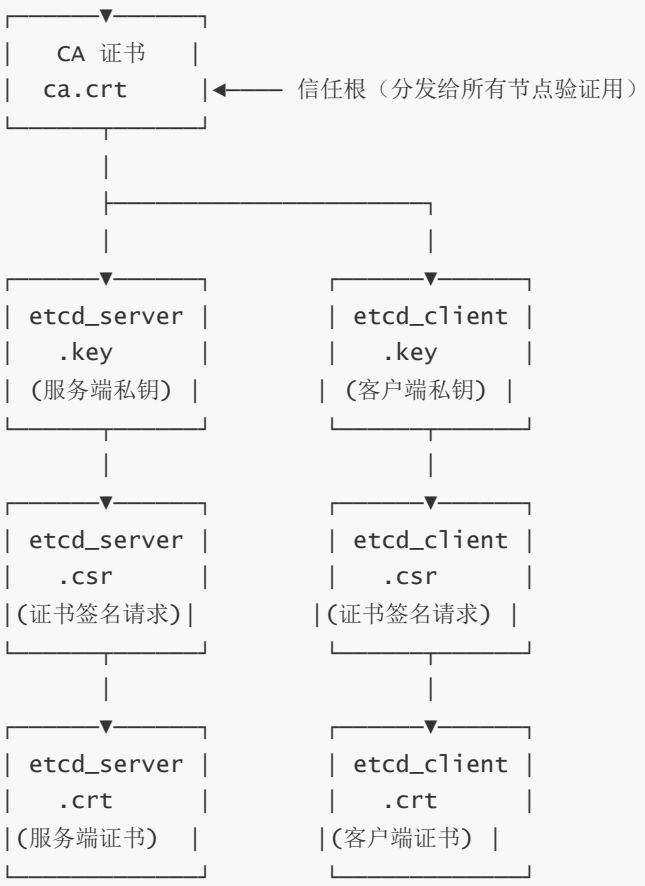
2. 证书工具

- openssl ：常用，使用简单方便。
- cfssl： 更高级工具，功能更多。

```
cfssl下载链接：https://github.com/cloudflare/cfssl/releases
```

3. 创建CA证书

```
┌──────────────┐
|   CA 私钥    |
|  ca.key      |◄─── 签发所有证书（保密，不外传）
└──────┬───────┘
       |
```




```
1. 生成一个 2048 位的 ca.key 文件
[root@localhost etcd-3.6.5]# openssl genrsa -out ca.key 2048

2.  在 ca.key 文件的基础上，生成 ca.crt 文件（用参数 -days 设置证书有效期）
[root@localhost etcd-3.6.5]# openssl req -x509 -new -nodes -key ca.key -subj
"/CN=etcd" -days 10000 -out ca.crt

注意：通过 -subj 参数指定了证书的主题。具体来说，这里仅设置了 CN（Common Name），并将其
设置为etcd,Common Name (CN) 或其他字段的内容对于验证证书的有效性和建立信任关系至关重要;
在某些情况下，可能还需要提供组织名称(O)，组织单位(OU)，国家(C)等信息。
```

4. 创建etcd_server证书

- 生成etcd_server.key私钥文件

```
1. 生成一个 2048 位的 etcd_server.key 文件：
[root@localhost etcd-3.6.5]# openssl genrsa -out etcd_server.key 2048
```

- 创建ssl.cnf配置文件

```
###   alt_names:  表示etcd集群的服务器ip
[ req ]
req_extensions = v3_req
distinguished_name = req_distinguished_name

[ req_distinguished_name ]

[ v3_req ]
basicConstraints = CA:FALSE
```

```
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names

[ alt_names ]
IP.1 = 19.190.3.130
IP.2 = 19.190.3.131
IP.3 = 19.190.3.132
```

- 通过etcd_ssl.cnf文件创建etcd_server.csr文件

```
[root@localhost etcd-3.6.5]# openssl req -new -key etcd_server.key -
config etcd_ssl.cnf -subj "/CN=etcd-server" -out etcd_server.csr
```

- 通过etcd_server.csr生成etcd_server.crt证书

```
[root@localhost etcd-3.6.5]# openssl x509 -req -in etcd_server.csr -CA
/root/etcd_crt/ca.crt -CAkey /root/etcd_crt/ca.key -CAcreateserial -days
36500 -extensions v3_req -extfile etcd_ssl.cnf -out etcd_server.crt
```

4. 创建etcd_client证书
   - 使用上述ca及etcd_ssl.cnf文件创建

```
[root@localhost etcd-3.6.5]# openssl genrsa -out etcd_client.key 2048

[root@localhost etcd-3.6.5]# openssl req -new -key etcd_client.key -config
etcd_ssl.cnf -subj "/CN=etcd-client" -out etcd_client.csr

[root@localhost etcd-3.6.5]# openssl x509 -req -in etcd_client.csr -CA
/root/etcd_crt/ca.crt -CAkey /root/etcd_crt/ca.key -CAcreateserial -days
36500 -extensions v3_req -extfile etcd_ssl.cnf -out etcd_client.crt
```

5. 创建etcd配置文件
   - 配置文件`/etc/etcd/etcd.conf`
     - 3个etcd节点都需要这样配置，注意修改集群中`name`,`ip`等关键参数!

```
[root@localhost etcd-3.6.5]#  mkdir   /etc/etcd
[root@localhost etcd-3.6.5]#  mkdir  /var/log/etcd
[root@localhost etcd-3.6.5]#  touch   /etc/etcd/etcd.yml
[root@etcd1 etcd_crt]# cat /etc/etcd/etcd.yml
# 节点名称，集群内必须唯一
name: etcd1

# 数据存储目录
data-dir: /var/lib/etcd

# WAL 日志目录（建议与 data-dir 分开存放，提升 IO 性能）
wal-dir: /var/lib/etcd/wal

# 监听客户端请求的地址（建议同时监听本地，方便 etcdctl 管理）
listen-client-urls: https://127.0.0.1:2379,https://19.190.3.130:2379
```

```
# 对外公布的客户端访问地址（其他组件通过此地址连接）
advertise-client-urls: https://19.190.3.130:2379

# 监听 peer 通信地址
listen-peer-urls: https://19.190.3.130:2380

# 对外公布的 peer 通信地址
initial-advertise-peer-urls: https://19.190.3.130:2380

# 静态集群成员列表（三节点必须完全相同）
initial-cluster:
'etcd1=https://19.190.3.130:2380,etcd2=https://19.190.3.131:2380,etcd3=https
://19.190.3.132:2380'

# 集群唯一标识令牌
initial-cluster-token: etcd-cluster

# 初始集群状态：new 表示新建集群，existing 表示加入已有集群
initial-cluster-state: new

# ==================== 客户端 TLS 配置 ====================
client-transport-security:
  cert-file: /root/etcd_crt/etcd_server.crt
  key-file: /root/etcd_crt/etcd_server.key
  trusted-ca-file: /root/etcd_crt/ca.crt
  client-cert-auth: true
  auto-tls: false

# ==================== Peer TLS 配置 ====================
peer-transport-security:
  cert-file: /root/etcd_crt/etcd_server.crt
  key-file: /root/etcd_crt/etcd_server.key
  trusted-ca-file: /root/etcd_crt/ca.crt
  client-cert-auth: true
  auto-tls: false

# ==================== 性能与压缩配置 ====================
# 后端存储配额（默认 2GB，建议 8GB）
quota-backend-bytes: 8589934592

# 自动压缩历史版本（每小时一次）
auto-compaction-retention: "1"
auto-compaction-mode: periodic

# 心跳间隔与选举超时（毫秒）
heartbeat-interval: 100
election-timeout: 1000

# ==================== 日志配置 ====================
log-level: info
log-outputs: [/var/log/etcd/etcd.log]
[root@etcd1 etcd_crt]
```

- 创建etcd的service启动文件

```
[root@etcd1 etcd]# cat  /usr/lib/systemd/system/etcd.service
[Unit]
Description=Etcd Server
After=network.target
Documentation=https://github.com/coreos

[Service]
Type=notify
ExecStart=/usr/local/bin/etcd --config-file /etc/etcd/etcd.yml
Restart=on-failure
RestartSec=5
LimitNOFILE=65536

[Install]
WantedBy=multi-user.target
```

- 启动etcd服务

```
[root@etcd1 etcd]# systemctl   daemon-reload

[root@etcd1 etcd]# systectl start etcd.service

[root@etcd1 etcd]# systemctl enable  etcd.service
```

- 验证集群健康状态

```
[root@etcd1 etcd_crt]# etcdctl --cacert=/root/etcd_crt/ca.crt   --
cert=/root/etcd_crt/etcd_server.crt --key=/root/etcd_crt/etcd_server.key  --
endpoints=https://19.190.3.130:2379,https://19.190.3.131:2379,https://19.190
.3.132:2379 endpoint health
https://19.190.3.131:2379 is healthy: successfully committed proposal: took
= 21.190668ms
https://19.190.3.130:2379 is healthy: successfully committed proposal: took
= 21.258188ms
https://19.190.3.132:2379 is healthy: successfully committed proposal: took
= 26.477773ms
[root@etcd1 etcd_crt]#
```

前面说了，我们在创建etcd证书的时候，创建了etcd_server和etcd_client两套证书，因为etcd_client用于集群外部使用的，比如kubernetes apiseever。因此我们也需要验证etcd_client证书是否正常，确保后续kubernetes可以正常使用，这里通用使用etcd_client验证集群健康性即可，如果无法验证，则用在k8s中也会异常

```
[root@etcd1 etcd_crt]# etcdctl --cacert=/root/etcd_crt/ca.crt   --
cert=/root/etcd_crt/etcd_client.crt --key=/root/etcd_crt/etcd_client.key  --
endpoints=https://19.190.3.130:2379,https://19.190.3.131:2379,https://19.190
.3.132:2379 endpoint health
https://19.190.3.131:2379 is healthy: successfully committed proposal: took
= 22.497462ms
https://19.190.3.132:2379 is healthy: successfully committed proposal: took
= 28.092627ms
https://19.190.3.130:2379 is healthy: successfully committed proposal: took
= 28.406738ms
[root@etcd1 etcd_crt]#
```