LightDB Distributed Guide

发布 *22.4*

LightDB

2023 年 01 月 18 日

目录:

1 前言

本文提供了手工搭建分布式的步骤及方法，以便自动化程序开发者及客户使用参考。

2 环境基本信息

本次测试环境为如下 8 台机器:

```
172.16.0.11
172.16.0.12
172.16.0.13
172.16.0.14
172.16.0.15
172.16.0.16
172.16.0.17
172.16.0.18
```

为了描述方便，后面指定具体哪台机器时，仅用 ip 地址末段 11,12..18 来指代。

3 安装分布式节点

3.1 安装单机版数据库

首先获取 `LightDB 22.4` 版本的安装包，解压后，参考 LightDB 安装手册 ，在每台机器上安装一个单机版实例。

为了便于操作，安装程序会在 `~/.bashrc` 中写入 LightDB 相关的环境变量，可以重新登录一下 shell，或者执行 `source ~/.bashrc` 使得环境变量生效。

我们演示环境把数据库实例安装在 `/data/lightdb` 目录下，您在具体操作时可以安装到其他目录下，只要 lightdb 用户有对应目录的权限即可。

安装完成后，可以查看 `LTHOME` 和 `LTDATA` 环境变量确定实际安装目录和实例目录:

```
[lightdb@0b3770c2d30a ~]$ echo $LTHOME
/data/lightdb/lightdb-x/13.8-22.4
[lightdb@0b3770c2d30a ~]$ echo $LTDATA
/data/lightdb/lightdb-x/13.8-22.4/data/defaultCluster/
```

3.2 启用分布式插件

LightDB 分布式功能是在 canopy 插件中实现的，在单机版环境中是没有启用 canopy 插件，需要在每台机器上手工启用一下。

首先编辑 `$LTDATA/lightdb.conf` 文件，修改 GUC 参数 `shared_preload_libraries` ，在参数最开头加上 `canopy` 。

例如:

```
# 原始值
shared_preload_libraries='lt_stat_statements,lt_stat_activity,lt_prewarm,lt_cron,lt_
↪hint_plan,lt_show_plans'

# 修改后(注意要在开头加canopy，不能加在其他位置)
shared_preload_libraries='canopy,lt_stat_statements,lt_stat_activity,lt_prewarm,lt_
↪cron,lt_hint_plan,lt_show_plans'
```

添加完成后，重启一下数据库让参数生效: `lt_ctl restart` 。

使用 `ltsql` 工具登录数据库，创建测试数据库 `test1` 和 `canopy` 插件。

```
[lightdb@0b3770c2d30a ~]$ ltsql
ltsql (13.8-22.4)
Type "help" for help.

# 注：创建test1库用于测试
lightdb@postgres=# CREATE DATABASE  test1;
NOTICE:  Canopy partially supports CREATE DATABASE for distributed databases
DETAIL:  Canopy does not propagate CREATE DATABASE command to workers
HINT:  You can manually create a database and its extensions on workers.
CREATE DATABASE

# 注：切换到刚创建的test1库中，(后续可以使用ltsql -d test1直接登录到test1库中)
lightdb@postgres=# \c test1
You are now connected to database "test1" as user "lightdb".

# 注：创建canopy插件
lightdb@test1=# CREATE EXTENSION canopy;
CREATE EXTENSION
```

此时插件就创建成功，需要注意的是插件是和数据库关联的，我们演示环境是在 `test1` 库中，您也可以在其他库中做。

3.3 LightDB 免密配置

因为需要在多个节点中相互调用，以执行分布式执行计划。所以我们指定所有节点机器之间相互登录数据库都是免密的，配置方法如下:

编辑文件: `$LTDATA/lt_hba.conf` 添加如下配置项:

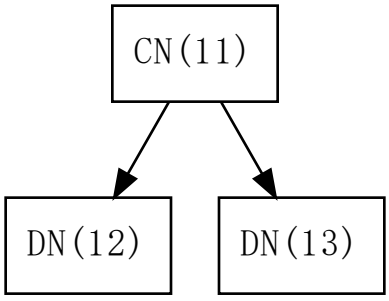
```
host    all             all             172.16.0.1/24           trust
host    replication     all             172.16.0.1/24           trust
```

上面配置项的含义就是所有 `172.16.0` 网段的客户端都可以免密登录数据库。

编辑完成后，调用 `lt_ctl reload` 重新加载一下配置文件让配置生效 (无需重启数据库)。

4 1CN 和 2DN 的分布式部署

基于 LightDB 单机版在 3 台机器上搭建一个由 1CN 和 2DN 组成的分布式集群。LightDB 的安装程序是支持这种部署方式的 (多机单实例部署方式)，这里为了演示部署细节，采用基于单机版 LightDB 的基础上部署。



4.1 部署分布式节点

参考章节: 安装分布式节点 , 在 11,12,13 三台机器上各部署分布式节点。

4.2 添加分布式节点

我们在 11(CN) 机器上登录数据库，添加两个 DN 节点:

```
[lightdb@0b3770c2d30a defaultCluster]$ ltsql -d test1
ltsql (13.8-22.4)
Type "help" for help.

lightdb@test1=# SELECT canopy_add_node('172.16.0.12', 5432);
canopy_add_node
-----------------
               2
(1 row)

lightdb@test1=# SELECT canopy_add_node('172.16.0.13', 5432);
canopy_add_node
-----------------
               3
(1 row)

lightdb@test1=# SELECT canopy_set_coordinator_host('172.16.0.11', 5432);
canopy_set_coordinator_host
----------------------------

(1 row)
```

这样就形成了以 11 为 CN 节点，12 和 13 为 DN 节点的分布式架构。

我们可以查询 pg_dist_node 表获取节点信息:

```
lightdb@test1=# select * from pg_dist_node;
nodeid | groupid |  nodename   | nodeport | noderack | hasmetadata | isactive |␣
↪noderole | nodecluster | metadatasynced | shouldhaveshards
--------+---------+-------------+----------+----------+-------------+----------+------
↪----+-------------+----------------+------------------
      2 |       2 | 172.16.0.12 |     5432 | default  | t           | t        |␣
↪primary  | default     | t              | t
      3 |       3 | 172.16.0.13 |     5432 | default  | t           | t        |␣
↪primary  | default     | t              | t
      5 |       0 | 172.16.0.11 |     5432 | default  | t           | t        |␣
↪primary  | default     | t              | f
(3 rows)
```

4.3 分布式测试

我们创建一个分布式表简单测试一下

```
# 创建一个普通本地表
lightdb@test1=# create table test_table(id int primary key, name text);
CREATE TABLE

# 插入10w条测试数据
lightdb@test1=# insert into test_table select v, v || 'name' from generate_series(1,
↪100000) as v;
INSERT 0 100000

# 把普通表改为分布式表
lightdb@test1=# select create_distributed_table('test_table', 'id');
NOTICE:  Copying data from local table...
NOTICE:  copying the data has completed
DETAIL:  The local data in the table is no longer visible, but is still on disk.
HINT:  To remove the local data, run: SELECT truncate_local_data_after_distributing_
↪table($$public.test_table$$)
create_distributed_table
--------------------------

(1 row)
```

此时我们成功创建了一个分布式表，我们在做查询操作时，会走分布式执行计划:

```
lightdb@test1=# explain select count(*) from test_table;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
↪-------------------
Aggregate  (cost=250.00..250.02 rows=1 width=8)
   ->  Custom Scan (Canopy Adaptive)  (cost=0.00..0.00 rows=100000 width=8)
         Task Count: 32
         Tasks Shown: One of 32
         ->  Task
               Node: host=172.16.0.12 port=5432 dbname=test1
               ->  Aggregate  (cost=43.99..44.00 rows=1 width=8)
                     ->  Seq Scan on test_table_102040 test_table  (cost=0.00..38.59␣
↪rows=2159 width=0)
(8 rows)
```

我们可以通过 canopy_tables 查看分布式表信息:

```
lightdb@test1=# select * from canopy_tables;
-[ RECORD 1 ]-------+------------
table_name          | test_table
canopy_table_type   | distributed
distribution_column | id
colocation_id       | 2
table_size          | 7488 kB
shard_count         | 32
table_owner         | lightdb
access_method       | heap
```

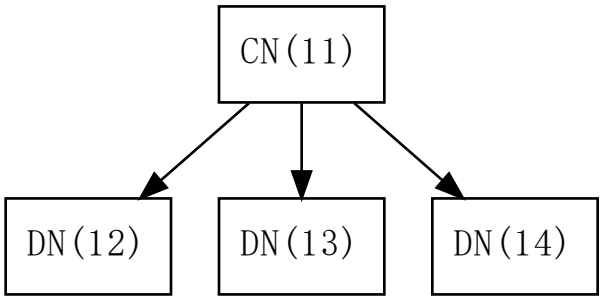
我们通过 canopy_shards 表查看表数据的分布情况:

```
lightdb@test1=# select table_name,shardid,shard_name,nodename,nodeport from canopy_
↪shards;
table_name | shardid |    shard_name     |  nodename   | nodeport
-----------+---------+-------------------+-------------+----------
test_table |  102040 | test_table_102040 | 172.16.0.12 |     5432
test_table |  102041 | test_table_102041 | 172.16.0.13 |     5432
test_table |  102042 | test_table_102042 | 172.16.0.12 |     5432
test_table |  102043 | test_table_102043 | 172.16.0.13 |     5432
...
test_table |  102068 | test_table_102068 | 172.16.0.12 |     5432
test_table |  102069 | test_table_102069 | 172.16.0.13 |     5432
test_table |  102070 | test_table_102070 | 172.16.0.12 |     5432
test_table |  102071 | test_table_102071 | 172.16.0.13 |     5432
(32 rows)
```

我们可以看到数据分片分布在 12,13 两台机器上。

5 DN 节点扩容

我们把前面部署的 1CN/2DN 架构添加 1 个 DN 节点扩展为 1CN/3DN 架构。



5.1 部署分布式节点

在 14 机器上部署分布式节点（参考章节: 安装分布式节点 ）。

5.2 添加分布式节点

在 11(CN) 上执行 SQL 添加 DN 节点:

```
lightdb@test1=# SELECT canopy_add_node('172.16.0.14', 5432);
canopy_add_node
-----------------
               6
(1 row)
```

5.3 重新分布数据

添加新的 DN 节点后，已有的分布式表是不会自动扩展到新节点的。新创建的分布式表可以自动分布式到 14 节点。如果已有的分布式表确实需要调整，则需要执行 rebalance_table_shards 操作，本操作需要依赖 GUC 参数 wal_level 调整。

```
lightdb@test1=# SELECT rebalance_table_shards('test_table');
NOTICE:  Moving shard 102041 from 172.16.0.13:5432 to 172.16.0.14:5432 ...
ERROR:  ERROR:  logical decoding requires wal_level >= logical
CONTEXT:  while executing command on 172.16.0.13:5432
while executing command on localhost:5432
```

wal_level 参数调整方法可以参考章节启用分布式插件 中的 shared_preload_libraries 参数修改方法，把 wal_level 修改为 wal_level=logical 。

修改后，可以重新执行分片操作:

```
lightdb@test1=# SELECT rebalance_table_shards('test_table');
NOTICE:  Moving shard 102041 from 172.16.0.13:5432 to 172.16.0.14:5432 ...
NOTICE:  Moving shard 102040 from 172.16.0.12:5432 to 172.16.0.14:5432 ...
NOTICE:  Moving shard 102043 from 172.16.0.13:5432 to 172.16.0.14:5432 ...
NOTICE:  Moving shard 102042 from 172.16.0.12:5432 to 172.16.0.14:5432 ...
NOTICE:  Moving shard 102045 from 172.16.0.13:5432 to 172.16.0.14:5432 ...
NOTICE:  Moving shard 102044 from 172.16.0.12:5432 to 172.16.0.14:5432 ...
NOTICE:  Moving shard 102047 from 172.16.0.13:5432 to 172.16.0.14:5432 ...
NOTICE:  Moving shard 102046 from 172.16.0.12:5432 to 172.16.0.14:5432 ...
NOTICE:  Moving shard 102049 from 172.16.0.13:5432 to 172.16.0.14:5432 ...
NOTICE:  Moving shard 102048 from 172.16.0.12:5432 to 172.16.0.14:5432 ...
rebalance_table_shards
------------------------

(1 row)
```

重新分片后，可以再次查看 canopy_shards 表查看分布情况，可以看到部分分片已经分布到 14 节点上:

```
lightdb@test1=# select table_name,shardid,shard_name,nodename,nodeport from canopy_
↪shards;
table_name | shardid |    shard_name     |  nodename   | nodeport
------------+---------+-------------------+-------------+----------
test_table |  102040 | test_table_102040 | 172.16.0.14 |     5432
```

(下页继续)

(续上页)

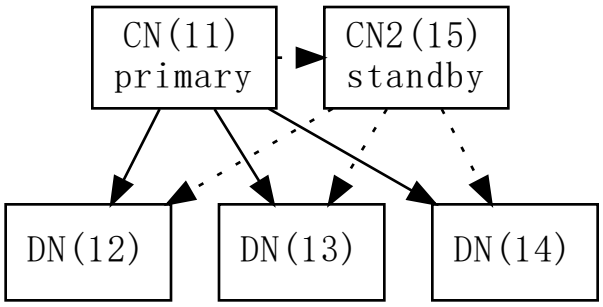
```
test_table |  102041 | test_table_102041 | 172.16.0.14 |      5432
...
test_table |  102070 | test_table_102070 | 172.16.0.12 |      5432
test_table |  102071 | test_table_102071 | 172.16.0.13 |      5432
(32 rows)
```

6 CN 节点扩容和高可用部署

LightDB 分布式的 CN 节点和 DN 节点都支持高可用部署形式，以获得更好的可靠性和性能。

在这种架构下，每个节点都是一个小的高可用集群。

在本节中，我们以 11 节点添加一个备机 15 为例讲解如何搭建一主一备高可用集群 (基于 ltcluster)，并且让我们分布式环境形成如下两个 CN 的结构:



如果您使用 Patroni 可以参考章节 (*Patroni 高可用部署方法*) 的方法来部署高可用。

6.1 部署主节点

我们先给 11 节点注册为 ltcluster 的主节点，纳入 ltcluster 的管理。

在 11 上创建配置文件 `${LTHOME}/etc/ltcluster/ltcluster.conf`:

```
cat>${LTHOME}/etc/ltcluster/ltcluster.conf<<EOF
node_id=11
node_name='cn-11'
conninfo='host=172.16.0.11 port=5432 user=ltcluster dbname=ltcluster connect_timeout=2
↪'
data_directory='${LTDATA}'
pg_bindir='${LTHOME}/bin'
failover='automatic'
log_level=INFO
log_facility=STDERR
log_file='${LTHOME}/etc/ltcluster/ltcluster.log'
shutdown_check_timeout=1800
```

(下页继续)

(续上页)

```
use_replication_slots=true
promote_command='${LTHOME}/bin/ltcluster standby promote -f ${LTHOME}/etc/ltcluster/
↪ltcluster.conf'
follow_command='${LTHOME}/bin/ltcluster standby follow -f ${LTHOME}/etc/ltcluster/
↪ltcluster.conf  --upstream-node-id=%n'
EOF
```

在 11 数据库上创建 ltcluster 的专用数据库和用户:

```
ltsql -c "CREATE ROLE ltcluster SUPERUSER PASSWORD 'ltcluster' login;"
ltsql -c "CREATE DATABASE ltcluster OWNER ltcluster;"
```

修改 shared_preload_libraries 参数，添加 ltcluster``配置项(参考章节:ref:`enable_distributed` )，注意 canopy 有顺序要求，要在最前面，ltcluster 添加在 canopy 后面即可。添加后使用 ``lt_ctl restart 重启数据库让配置生效。

把 11 数据库注册为 ltcluster 主节点:

```
ltcluster primary register -f ${LTHOME}/etc/ltcluster/ltcluster.conf -F
```

启动 ltclusterd 守护进程:

```
ltclusterd -d \
   -f ${LTHOME}/etc/ltcluster/ltcluster.conf \
   -p ${LTHOME}/etc/ltcluster/ltcluster.pid
```

通过如下命令可以查看 ltcluster 集群状态，此时只有一个节点

```
[lightdb@0b3770c2d30a ~]$ ltcluster -f ${LTHOME}/etc/ltcluster/ltcluster.conf service␣
↪status
ID | Name  | Role    | Status    | Upstream | ltclusterd | PID   | Paused? | Upstream␣
↪last seen
----+-------+---------+-----------+----------+------------+-------+---------+---------
↪-----------
11 | cn-11 | primary | * running |          | running    | 21335 | no      | n/a
```

6.2 部署备节点

在 15 机器上部署单机版，准备作为 11 节点的备机（参考章节: *安装单机版数据库*)。

在 15 上创建配置文件 ${LTHOME}/etc/ltcluster/ltcluster.conf :

```
cat>${LTHOME}/etc/ltcluster/ltcluster.conf<<EOF
node_id=15
node_name='cn-15'
conninfo='host=172.16.0.15 port=5432 user=ltcluster dbname=ltcluster connect_timeout=2
↪'
data_directory='${LTDATA}'
pg_bindir='${LTHOME}/bin'
failover='automatic'
log_level=INFO
log_facility=STDERR
log_file='${LTHOME}/etc/ltcluster/ltcluster.log'
shutdown_check_timeout=1800
use_replication_slots=true
```

(下页继续)

(续上页)

```
promote_command='${LTHOME}/bin/ltcluster standby promote -f ${LTHOME}/etc/ltcluster/
↪ltcluster.conf'
follow_command='${LTHOME}/bin/ltcluster standby follow -f ${LTHOME}/etc/ltcluster/
↪ltcluster.conf  --upstream-node-id=%n'
EOF
```

执行 lt_ctl stop 停止数据库，因为 15 节点是作为 11 节点的备机，所以需要停机从 11 节点拷贝实例数据库。

```
# 注意，这一步需要从11上拷贝完整的实例数据到15，
# 如果数据库已经有较多数据库，拷贝耗时会较久。
ltcluster -h 172.16.0.11 -p 5432 \
   -U ltcluster -d ltcluster \
   --log-level=DEBUG --verbose \
   -f ${LTHOME}/etc/ltcluster/ltcluster.conf standby clone -F
```

执行 lt_ctl start 启动数据库。

执行下面命令把 ltcluster 注册为 standby

```
ltcluster -f ${LTHOME}/etc/ltcluster/ltcluster.conf standby register
```

启动 ltclusterd 守护进程:

```
ltclusterd -d \
   -f ${LTHOME}/etc/ltcluster/ltcluster.conf \
   -p ${LTHOME}/etc/ltcluster/ltcluster.pid
```

此时查看集群状态，CN 节点的一主一备已经部署完成。

```
[lightdb@fc9eb9ccfa94 ~]$ ltcluster -f ${LTHOME}/etc/ltcluster/ltcluster.conf service␣
↪status
ID | Name  | Role    | Status    | Upstream | ltclusterd | PID   | Paused? | Upstream␣
↪last seen
----+-------+---------+-----------+----------+------------+-------+---------+---------
↪-----------
11 | cn-11 | primary | * running |          | running    | 21335 | no      | n/a
15 | cn-15 | standby |   running | cn-11    | running    | 1302  | no      | 1␣
↪second(s) ago
```

6.3 CN 备节点介绍

默认情况下，在 15 上可以执行只读操作。

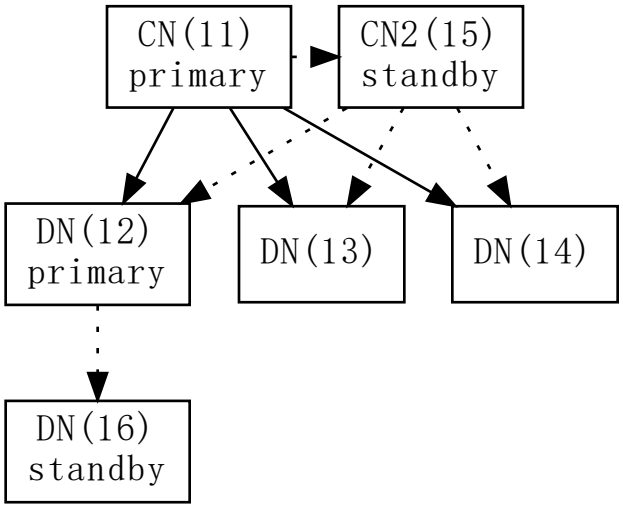
```
lightdb@test1=# select count(*) from test_table;
count
--------
100000
(1 row)

lightdb@test1=# update test_table set name='abc' where id = 1;
ERROR:  writing to worker nodes is not currently allowed
DETAIL:  the database is read-only
```

在 LightDB 高可用的备机中，因为所有数据前部来源于流复制，所以在备机是不能直接支持写操作的。但在分布式环境中，DML 操作是分发到 DN 节点执行的，所以 CN 备节点在启用 `canopy.writable_standby_coordinator` 选项后，可以在 15 上执行分布式表的 DML 操作。

```
lightdb@test1=# set canopy.writable_standby_coordinator=on;
SET
lightdb@test1=# update test_table set name='abc' where id = 1;
UPDATE 1

# DDL操作依然不支持
lightdb@test1=# create table test_table2(id int primary key);
RROR:  cannot execute CREATE TABLE in a read-only transaction
```

7 DN 节点高可用介绍

参考章节 (*CN 节点扩容和高可用部署*), 同样可以为 DN 节点添加备机，本节不再讲述部署细节。假设已经为 DN 节点 12 添加一个备节点 16，形成如下架构:



8 DN 节点高可用异常及处理

下面我们主动停止 12 的数据库，等会儿后，在 16 上可以查看 12 和 16 高可用集群状态:

```
[lightdb@7531b42d601c ~]$ ltcluster -f $LTHOME/etc/ltcluster/ltcluster.conf service␣
↪status
ID | Name   | Role    | Status    | Upstream | ltclusterd | PID   | Paused? | Upstream␣
↪last seen
----+-------+---------+-----------+----------+------------+-------+---------+----------
↪----------
```

(下页继续)

(续上页)

```
12 | cn-12 | primary | - failed  | ?         | n/a         | n/a  | n/a      | n/a
16 | cn-16 | primary | * running |           | running     | 1161 | no       | n/a

WARNING: following issues were detected
- unable to  connect to node "cn-12" (ID: 12)

HINT: execute with --verbose option to see connection error messages
```

此时 16 已经提升为 primary。

此时虽然备机已经通过 ltclusterd 守护进程自动提升为主，但是分布式集群中的节点数据仍然指向 12 节点。所以执行 SQL 还是会失败。

```
lightdb@test1=# select count(*) from test_table;
ERROR:  connection to the remote node 172.16.0.12:5432 failed with the following␣
↪error: could not connect to server: Connection refused
   Is the server running on host "172.16.0.12" and accepting
   TCP/IP connections on port 5432?
```

8.1 手工恢复

我们在 11(CN) 上把 12 节点的元数据改成指向 16 节点 (pg_dist_node 表)。

```
# 查询pg_dist_node得到12节点的nodeid为2
lightdb@test1=# select * from pg_dist_node;
nodeid | groupid |  nodename   | nodeport | noderack | hasmetadata | isactive |␣
↪noderole | nodecluster | metadatasynced | shouldhaveshards
--------+---------+-------------+----------+----------+-------------+----------+------
↪----+-------------+----------------+------------------
      2 |       2 | 172.16.0.12 |     5432 | default  | t           | t        |␣
↪primary  | default     | t              | t
      3 |       3 | 172.16.0.13 |     5432 | default  | t           | t        |␣
↪primary  | default     | t              | t
      5 |       0 | 172.16.0.11 |     5432 | default  | t           | t        |␣
↪primary  | default     | t              | f
      6 |       5 | 172.16.0.14 |     5432 | default  | t           | t        |␣
↪primary  | default     | t              | t
(4 rows)

# 修改nodeid=2的节点数据库地址为：172.16.0.16:5432
lightdb@test1=# select canopy_update_node(2,'172.16.0.16',5432);
canopy_update_node
--------------------

(1 row)

# 此时再次查询分布式表，已经可以查询成功。
lightdb@test1=# select count(*) from test_table;
count
--------
100000
(1 row)
```

8.2 自动恢复

根据前面分析，高可用 failover 后，只需要调用 canopy_update_node 更新一下元数据就可以了。我们可以通过手工或者高可用管理工具完成这个操作。

LightDB 提供了 canopy_ha_monitor.sh 用于监控 DN 节点的 failover 事件。这个脚本部署在 CN 上，如果有多个 CN，在每个 CN 上面部署一个。

在启动此脚本前，需要修改脚本配置，打开脚本 ${LTHOME}/bin/canopy_ha_monitor.sh，修改如下参数:

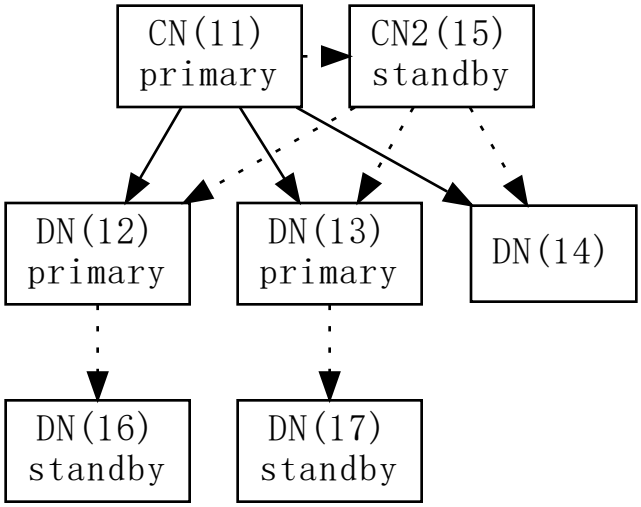
```
#!/bin/bash

# ---------------- config  start -------------------
# 分布式数据库名称(在此数据库中必须创建)
readonly DATABASE_NAME=postgres

# 数据库用户名
readonly DATABASE_USER=lightdb

# CN节点的数据库ip地址和端口号，如果有多个CN，则配置本机CN就可以了
readonly CN_CONNECT_INFO="172.16.0.19:61001"

declare -A DN_HA_MAP

# DN:  primary => standby
# 所有DN的高可用关联信息，前面是主，后面是备
DN_HA_MAP=(
   ["172.16.0.18:61001"]="172.16.0.18:61002"
   ["172.16.0.17:61001"]="172.16.0.17:61002"
)

# ---------------- config  end -------------------
```

运行脚本后，会连接 CN 节点数据库，获取所有 DN 节点信息，并检测数据库工作是否正常。

如果有异常，则检测对应 DN 节点的备库是否工作正常，是否已经切换为主模式，如果备库已经切换为主模式，则调用 canopy_update_node 函数在 CN 节点上修改对应节点的地址为备库的地址。这样分布式数据库就可以正常使用。

9 Patroni 高可用部署方法

因为每个节点的高可用集群是独立的，所以我们前面部署的基础上，增加一个 17 节点作为 13 节点的备机，让 13 和 17 两个节点组成 Patroni 高可用集群，以演示高可用的部署方法。最终架构如下所示:



9.1 部署 etcd

Patroni 需要依赖 ETCD，我们在 18 机器上部署一个单机版的 ETCD，在正式环境需要部署 ETCD 集群。

获取 etcd 的 release 包后，解压就可以使用。本文测试版本为: `etcd-v3.4.23-linux-amd64`

```
etcd --name 'etcd18' \
   --data-dir '/data/etcd' \
   --listen-client-urls 'http://0.0.0.0:2379' \
   --advertise-client-urls 'http://172.16.0.18:2379' \
   --listen-peer-urls 'http://0.0.0.0:2380' \
   --initial-advertise-peer-urls 'http://0.0.0.0:2380'  \
   --enable-v2=true
```

9.2 部署 Patroni 主节点

获取 LightDB 的 Patroni 安装包 (`patroni-2.1.3-lightdb`), 在 13 机器上解压。

因为 Patroni 是基于 Python 开发，所以需要先有 Python3 环境，然后安装如下依赖:

```
# 先进入 patroni-2.1.3-lightdb 目录
pip3 install --user -U pip setuptools
pip3 install --user -r requirements.txt
pip3 install --user psycopg
```

进入 `patroni-2.1.3-lightdb` 目录, 修改配置文件 `lightdb0.yml` :

```
# 集群名称
scope: cluster13

```

(下页继续)

(续上页)

```
#节点名称
name: lightdb13

# etcd配置段中添加hosts
hosts:
- 172.16.0.18:2379

# postgresql配置段
# 修改listen,设置为机器A上实例的监听端口和IP
listen: 127.0.0.1,172.16.0.13:5432
connect_addr: 172.16.0.13:5432
# 修改data_dir, 同前文安装时一致
data_dir: /data/lightdb/lightdb-x/13.8-22.4/data/defaultCluster
# 修改superuser用户名密码, 同前文安装时保持一致,例如:
superuser:
   username: lightdb
   password: lightdb123
```

登录数据库，创建 Patroni 需要的用户，如果您用其他的用户名和密码，需要对应修改 `lightdb0.yml` 配置文件。

```
CREATE USER replicator WITH replication encrypted password 'rep-pass';
CREATE USER rewind_user WITH encrypted password 'rewind_password';
```

启动 patroni

```
./patroni.py lightdb0.yml
```

此时可以通过 `patronictl.py` 工具查看集群状态，此时显示仅有一个 Role 为 Leader 的节点:

```
[lightdb@1e631f45d1f0 patroni-2.1.3-lightdb]$ ./patronictl.py  -c ./lightdb0.yml  list
+-----------+-------------+--------+---------+----+-----------+-----------------+
| Member    | Host        | Role   | State   | TL | Lag in MB | Pending restart |
+ Cluster: cluster13 (7188356765573697849) --+----+-----------+-----------------+
| lightdb13 | 172.16.0.13 | Leader | running |  2 |           | *               |
+-----------+-------------+--------+---------+----+-----------+-----------------+
```

9.3 部署 Patroni 备节点

在备节点 17 上，同样参考章节 (*部署 Patroni 主节点*), 修改 `lightdb0.yml`

进入 `patroni-2.1.3-lightdb` 目录, 修改配置文件 `lightdb0.yml` :

```
# 集群名称
scope: cluster13

#节点名称
name: lightdb17

# etcd配置段中添加hosts
hosts:
- 172.16.0.18:2379

# postgresql配置段
# 修改listen,设置为机器A上实例的监听端口和IP
```

(下页继续)

(续上页)

```
listen: 127.0.0.1,172.16.0.17:5432
connect_addr: 172.16.0.17:5432
# 修改data_dir, 同前文安装时一致
data_dir: /data/lightdb/lightdb-x/13.8-22.4/data/defaultCluster
# 修改superuser用户名密码, 同前文安装时保持一致,例如:
superuser:
   username: lightdb
   password: lightdb123
```

启动 patroni

```
./patroni.py lightdb0.yml
```

因为 cluster13 集群已经有主节点，所以 17 节点自动工作在备机模式。

此时通过 patronictl.py 工具查看集群状态，可以看到 patroni 集群有两个节点组成。

```
[lightdb@1e631f45d1f0 patroni-2.1.3-lightdb]$ ./patronictl.py  -c ./lightdb0.yml  list
+-----------+-------------+---------+---------+----+-----------+-----------------+
| Member    | Host        | Role    | State   | TL | Lag in MB | Pending restart |
+ Cluster: cluster13 (7188356765573697849) ---+----+-----------+-----------------+
| lightdb13 | 172.16.0.13 | Leader  | running |  4 |           | *               |
| lightdb17 | 172.16.0.17 | Replica | running |  4 |         0 | *               |
+-----------+-------------+---------+---------+----+-----------+-----------------+
```