Samba-Cluster with CTDB and GlusterFS

Author:
Stefan Kania

Ort:
SambaXP 2020 Göttingen

31. Mai 2020

Content

1 Introduction

In this year's tutorial I will show you how easy it is to set up a CTDB-Cluster with GlusterFS to manage a failover load-balanced fileserver-cluster. We will create a GlusterFS replicated volume of two nodes. On the same hosts we will configure a Samba-4.11 CTDB-cluster. You will have one Windows- and one Linux-client to access the shares. At the end of the day you will have all the knowledge to set up a CTDB-cluster on your own.

I created a special Vagrant box and a Vagrantfile with the basic configuration of all needed Linux-hosts. In table 1.1 you will find the names and IP-addresses of all Linux-hosts:

Hostname	IP-1	IP-2
addc01	192.168.56.41	n/a
cluster01	192.168.56.42	192.168.57.42
cluster02	192.168.56.43	192.168.57.43
linux	192.168.56.51	n/a

Table 1.1: Names and IPs of all Linux-hosts

IP-1 is the IP-address from the production network and *IP-2* is the IP-address for the heartbeat network. The heartbeat network is only for the communication between the cluster-hosts. The clients will connect to the cluster only via the production network. Later you will need two more IP-addresses as dynamic IP for the CTDB-nodes.

To access all the Linux-hosts the systems have a user named *vagrant* with the password *vagrant*. The password for *root* is also *vagrant*.

1.1 Already prepared

Because we only have one day I already prepared some basic setups, so we don't have to deal with the basics.

The domain controller

On host *addc01* is a Samba 4 domain controller configured with the settings in table 1.2.

Hostname	addc-01.example.net
Realm	EXAMPLE.NET
Admin-password	Passw0rd
DNS-backend	internal DNS-server
Forwarder	1.1.1.1
Samba-Version	4.11 (Louis van Belle)

Table 1.2: Domain controller settings

1.2 The cluster-nodes

As you have seen in Table 1.1 you have two host for setting up the cluster. In table 1.3 you will find the important information of both hosts.

host 01 public-name	cluster-01.example.net
host 02 public-name	cluster-02.example.net
host 01 heartbeat-name	c-01.heartbeat.net
host 02 heartbeat-name	c-02.heartbeat.net
Samba-Version	4.11 (Louis van Belle)
Gluster-Version	7.x
Brick-Device	sdb1
Brick mountpoint	/gluster
Volume mountpoint	/glusterfs
Brick size	2GB LVM2 thinprovisioned

Table 1.3: Domain controller settings

2 Setting up GlusterFS

The first step will be, setting up the replicated Gluster-Volume with two nodes. The file-system is already prepared, so we can start directly with configuring the Gluster-cluster. There are only a few steps we need to get a working Gluster-cluster. But before you start you have to take care of a few things:

- You need a *xfs* formatted partition on each of your Gluster-nodes. If you want to setup *Gluster-snapshot* you have to set up a LVM2 thinprovisioned volume, as it is on the systems you are using here today.
- One *mountpoint* for the brick. Here we user /gluster. A mountpoint for the Gluster-volume it self. Here we will use /glusterfs
- Two network cards, one for connecting the cluster to the production network, here we will user *enp0s8* with IP-address *192.168.56.42* for the hosts *cluster-01* and *192.168.56.43* for host *cluster-02*. The second network card will be used for the heartbeat network between the cluster hosts. The second network card – *enp0s9* – is already configured with IP-address *192.168.57.42* for the first cluster host and *192.168.57.43* for the second cluster host.
- For each of your CTDB-nodes you need another IP-address from your production network, this IP-address will be given to the dynamically via CTDB when the cluster starts. Later in this tutorial CTDB will manage the takeover of this IP if one of the hosts shutdown or failed. Here we will take *192.168.56.61* for the first CTDB-host and *192.168.56.62* for the second host.

For today all the steps are already done.

Add the peers

The first step will be to set up the *trusted pool*. All hosts must be added to the trusted pool. In listing 2.1 you will see the commands to add the second gluster-node to the pool. You have to do this on the first of the gluster-host:

```
root@cluster-01:~# gluster peer probe c-02
peer probe: success.
```

Listing 2.1: Add the host to the trusted pool

If you try to add the peer and you get one of the error messages from listing 2.2:

```
root@cluster-01:~# gluster peer probe c-02
Connection failed. Please check if gluster daemon is operational.

root@cluster-01:~# gluster peer probe c-02
peer probe: failed: Probe returned with Transport endpoint is not connected
```

Listing 2.2: Errors during adding a peer

The first error message will point you to a not running *glusterd* on the host you are trying to add the peer. Restart the the daemon with `systemctl restart glusterd`. The second error message will point to a not running daemon on the peer you are trying to add to the pool. Restart the *glusterd* on the other node.

If you could add the node *cluster-02* on the node *cluster-01*, add the host *cluster-01* to the trusted pool on node *cluster-02*, as you can see in listing 2.3:

```
root@cluster-02:~# gluster peer probe c-01
peer probe: success.
```

Listing 2.3: Add the other node to the pool

Now you can check the status of each node and take a look at the list of all nodes with the `gluster`-command as you can see in listing 2.4:

```
root@cluster-01:~# gluster peer status
Number of Peers: 1

Hostname: c-02
Uuid: aca7d361-51df-4d1f-9b0f-4cf494029f21
State: Peer in Cluster (Connected)

root@cluster-02:~# gluster peer status
Number of Peers: 1

Hostname: c-01.heartbeat.net
Uuid: adafbf93-e716-4d99-bf89-e8044d57e3aa
State: Peer in Cluster (Connected)
Other names:
c-01

root@cluster-02:~# gluster pool list
UUID                                    Hostname                State
adafbf93-e716-4d99-bf89-e8044d57e3aa    c-01.heartbeat.net Connected
aca7d361-51df-4d1f-9b0f-4cf494029f21    localhost          Connected
```

Listing 2.4: Listing the nodes and the pool

On each host you will find the information of the peer in /var/lib/glusterd/peers/$< UUID >$.

Now you have all the peers added to the pool , you will need for the gluster-volume.

Creating the volume

The next step is creating the volume. But before we create the volume of two bricks let me explain some things. If you start creating the volume and give just two bricks as parameter you will see a warning, that it‘s not a good idea to create a replicated volume with only two bricks, because you will not be able to to set up a *quorum*. In a production environment you should always create a replicate volume of an odd number of nodes, because of the quorum. With an even number of nodes you can still have a quorum, but for example: You build a cluster of 4 nodes, one node get's separeted the other 3 nodes still have a valide quorum, but what will happed if the cluster is separeted in half, so you have 2 times 2 nodes, none of them will meat the quorum. So you see, an odd number of nodes is better for the quorum to work.

What is the *quorum* and why is it so important?

If you set up three nodes and you are lose the connection between node-1 and the other nodes (node-2 and node-3) but still the clients from the production network can reach all three nodes and all three nodes still running the glusterd. So one client can connect to node-1 and do some changes on a file. Another client can connect to the rest of the Gluster-cluster (node-2 and node-3) and change the same file, because node-2 and node-3 can‘t communicate to node-1 about open files anymore.

You will get a *split brain*-status of your cluster as soon as the connection is reestablished. If you configure a quorum of 51% the two nodes still communicate (node-2 and node-3) will meet the quorum, but the other node (node-1) not. So the Gluster-daemon will stop taking any changes from a client on node-1 the node will either stop the service or will go to a read-only status.

With two nodes you can‘t set up a quorum, because each node is 50% of the cluster. Only with an odd number of nodes you can configure a good working quorum. That‘s why you will get the warning when creating the volume with two nodes. But it‘s just a warning.

This problem will apply to CTDB too. In the future the developer plan to introduce an optional quorum where nodes will have to be connected to >50% of configured nodes before they can join the cluster.

With 2 nodes it is very easy to get a stupid form of split brain. Node A is shut down and node B is active, updating information in persistent databases (perhaps id-mapping info?). Node B is shut down and node A is restarted. Now node A's old database is in use for a while. When node B is restarted then some databases from A might be used and some from B - it depends on the sequence numbers.

Now choose one of the nodes to creating the volume. In listing 2.5 you will see the command:

```
root@cluster-01:~# gluster volume create gv0 replica 2 c-01:/gluster/brick \
     c-02:/gluster/brick
Replica 2 volumes are prone to split-brain. Use Arbiter or Replica 3 to avoid \
    this. See: http://docs.gluster.org/en/latest/Administrator%20Guide/Split%20\
    brain%20and%20ways%20to%20deal%20with%20it/.
Do you still want to continue?
 (y/n) y
volume create: gv0: success: please start the volume to access data
```

Listing 2.5: Creating the volume

This is the warning I mentioned before. But by typing a «y» you can create the volume anyway.

Now let's take a look at the setup and the status of the volume. You can see the result in listing 2.6:

```
root@cluster-01:~# gluster v info

Volume Name: gv0
Type: Replicate
Volume ID: 5d1e1031-5474-48e9-9451-1dbeb5ebb79e
Status: Created
Snapshot Count: 0
Number of Bricks: 1 x 2 = 2
Transport-type: tcp
Bricks:
Brick1: c-01:/gluster/brick
Brick2: c-02:/gluster/brick
Options Reconfigured:
transport.address-family: inet
storage.fips-mode-rchecksum: on
nfs.disable: on
performance.client-io-threads: off

root@cluster-01:~# gluster v status
Volume gv0 is not started
```

Listing 2.6: Status after creating the volume

With `gluster v info` or `gluster volume info` you will see the setup of the volume and a list of set parameters ad the end of the output. The command `gluster v status` is telling you, that the cluster is not running. You have to start the volume before you can use it. Listing 2.7 you see the command and the new status:

```
root@cluster-01:~# gluster v start gv0
volume start: gv0: success

root@cluster-01:~# gluster v status gv0
Status of volume: gv0
Gluster process                                 TCP Port  RDMA Port  Online  Pid
------------------------------------------------------------------------------
Brick c-01:/gluster/brick                       49152     0          Y       1583
Brick c-02:/gluster/brick                       49152     0          Y       9830
Self-heal Daemon on localhost                   N/A       N/A        Y       1604
Self-heal Daemon on c-02                        N/A       N/A        Y       9851

Task Status of Volume gv0
------------------------------------------------------------------------------
There are no active volume tasks
```

Listing 2.7: Starting the volume

Now the volume is running and ready to use.

To bring up the volume every time you restart your system you must enable the *glusterd* in systemd, as you can see in listing 2.8:

```
root@cluster-02:~# systemctl enable glusterd.service
Created symlink /etc/systemd/system/multi-user.target.wants/glusterd.service -> \
```

```
/lib/systemd/system/glusterd.service.
```

Listing 2.8: Activate glusterd

2.1 Setting some Samba-Options

To get a better performance from Gluster when connecting via SMB it's possible to set some options to your Gluster-volume. Starting from Gluster version 6 most of the options are put together in a group of options.

Hint!

Starting with Gluster7 the group-option for Samba is not a part of the debian-packages anymore. So I created a new group-options file

Listing 2.1.1 is showing the list of my group-options file:

```
cluster.self-heal-daemon=enable
performance.cache-invalidation=on
server.event-threads=4
client.event-threads=4
performance.parallel-readdir=on
performance.readdir-ahead=on
performance.nl-cache-timeout=600
performance.nl-cache=on
network.inode-lru-limit=200000
performance.md-cache-timeout=600
performance.stat-prefetch=on
performance.cache-samba-metadata=on
features.cache-invalidation-timeout=600
features.cache-invalidation=on
nfs.disable=on
cluster.data-self-heal=on
cluster.metadata-self-heal=on
cluster.entry-self-heal=on
cluster.force-migration=disable
```

Listing 2.1.1: Samba-group options

I used this options several times and the performance of the Gluster-volume is much better as without these options. You will find the file in /var/lib/glusterd/groups/ the name of the file is stka-samba. If you find a file named samba in this directory then you have the original file from the glusterfs-server package.

You only have to set this options on one of your nodes. In listing 2.1.2 you see the command to set and list the new options:

```
root@cluster-02:~# gluster v set gv0 group stka-samba
volume set: success

root@cluster-02:~# gluster v info

Volume Name: gv0
Type: Replicate
Volume ID: 5d1e1031-5474-48e9-9451-1dbeb5ebb79e
Status: Started
Snapshot Count: 0
Number of Bricks: 1 x 2 = 2
```

```
Transport-type: tcp
Bricks:
Brick1: c-01:/gluster/brick
Brick2: c-02:/gluster/brick
Options Reconfigured:
cluster.force-migration: disable
cluster.entry-self-heal: on
cluster.metadata-self-heal: on
cluster.data-self-heal: on
features.cache-invalidation: on
features.cache-invalidation-timeout: 600
performance.cache-samba-metadata: on
performance.stat-prefetch: on
performance.md-cache-timeout: 600
network.inode-lru-limit: 200000
performance.nl-cache: on
performance.nl-cache-timeout: 600
performance.readdir-ahead: on
performance.parallel-readdir: on
client.event-threads: 4
server.event-threads: 4
performance.cache-invalidation: on
cluster.self-heal-daemon: enable
transport.address-family: inet
storage.fips-mode-rchecksum: on
nfs.disable: on
performance.client-io-threads: off
```

Listing 2.1.2: Setting Samba-Options

As you can see, all the options are set. If you would like to set a single option you can do it with `gluster v set` $< volume-name > < option >=< value >$. To reset an option to it's original value use `gluster v reset`$< volume-name > < option >$. To see all options use `gluster v get` $< volume-name >$ `all`.

2.2 Mounting the gluster-volume

To use the volume you have to mount the volume either to a local mountpoint or over the network to a host which has the glusterfs-client installed. To mount the volume to a local mountpoint on your hosts the system will use *fuse*. First try to mount the volume manually. In listing 2.2.1 you can see how to mount the volume on both nodes:

```
root@cluster-02:~# mount -t glusterfs cluster-02:/gv0 /glusterfs

root@cluster-02:~# mount | grep glusterfs
cluster-02:/gv0 on /glusterfs type fuse.glusterfs (rw,relatime,\
        user_id=0,group_id=0,default_permissions,allow_other,\
        max_read=131072)


root@cluster-01:~# mount -t glusterfs cluster-01:/gv0 /glusterfs

root@cluster-01:~# mount | grep glusterfs
cluster-01:/gv0 on /glusterfs type fuse.glusterfs (rw,relatime,\
        user_id=0,group_id=0,default_permissions,allow_other,\
        max_read=131072)
```

Listing 2.2.1: Mount volume on both nodes

As you can see I used the host's own name as source on each host, but what will happened if you mount the volume via the network on a pure glusterfs-client if the host you use for mounting the volume crashes? The client will go to the next node and reconnect, because during the mount-process the client receives a list of all nodes holding the volume, so the client is knowing all nodes of the volume.

As I told you before and as you can see, the system is using fuse.mount to mount the volume. With most of the Debian based distributions there is a problem using *fuse.mount* with network-filesystems, because Debian first try to mount the fuse-volume and then starts the network and that will not work. So you need a different way to mount the volume during the start of your glusterfs-client. You will find a good solution in creating a systemd-script for mounting the volume. In listing 2.2.2 you will see a systemd-script to mount the volume:

```
[Unit]
Description = Data dir
After=network.target glusterfs-server.service
Required=network-online.target

[Mount]
RemainAfterExit=true
ExecStartPre=/usr/sbin/gluster volume list
ExecStart=/bin/mount -a -t glusterfs
Restart=on-failure
RestartSec=3
What=cluster-01:/gv0
Where=/glusterfs
Type=glusterfs
Options=defaults,acl

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

Listing 2.2.2: Systemd-script to mount the volume

Tip!

You will find the script on each host in /daten

It's very important that the name of the script is the same as the name of the mountpoint. If your mountpoint is not in filesystem-root but for example in /vol/glusterfs you must use the filename vol.glusterfs.mount for your script.

To try the script unmount the glustervolume with `umount /glusterfs` and run `systemctl start glusterfs.mount` afterwards check if the volume is mounted with `mount`. Now you have to enable the script with `systemctl enable glusterfs.mount`, so that the volume will be mounted every time you start your system.

Do all the steps on both nodes, so the volume will be mounted on both nodes everytime you reboot the system.

Now you can use the volume

2.3 What would be next

Normally you would test the volume by writing files, deactivating a node and seeing what happens if the host comes back after you have written some file to the remaining node.

And you would do the first performance- and recovery-tests, but we don't have so much time in a one day tutorial to do so. The only thing you should do is, on one node change to the directory /glusterfs, create some files and directories and see if the same entries appear on the other node in /glusterfs, this will show you, that the cluster is working.

Important!

Never ever write or delete any files or directories directly on the brick it self /gluster/brick. This will crash your volume!

3 Setting up CTDB

After setting up ther GlusterFS-cluster we now come to the main topic of the day – setting up a CTDB-cluster. The first step will be a look at the configuration-files for CTDB. All the needed packages are already installed. If you want to set up a CTDB-cluster on your own system, you need one more package next to the samba-packages. It's the package ctdb, if you use the SerNet-Packages install the sernet-samba-ctdb-package.

Let's take a look at the files needed for configuring CTDB. In table 3.1 you will find all files you need for configuring CTDB:

/etc/ctdb/ctdb.conf	basic configuration
/etc/ctdb/script.options	setting options for event-scripts
/etc/ctdb/nodes	All IP-addresses of all nodes
/etc/ctdb/public_addresses	dynamic IP-addresses for all nodes

Table 3.1: Configuration-files for CTDB

The ctdb.conf file

The ctdb.conf-file has changed a lot from the old configuration style. This file will no longer be used to configure the different services managed by CTDB. At the moment the only setting you have to do inside the file is setting up the *recovery lock*-file . This file is used by all nodes to check if it's possible to lock files inside the cluster for exclusive use. If you don't use a recovery lock file your cluster can run into a split brain situation. By default the *recovery lock* is NOT set. You should not use CTDB without a *recovery lock* unless you know what you are doing. The variable must point to a file inside your mounted gluster-volume. To use the recovery lock enter the line from listing 3.1 into /etc/ctdb/ctdb.conf on both nodes:

```
recovery lock = /glusterfs/ctdb.lock
```

Listing 3.1: Defining the recovery lock

The *recovery lock* setting needs to be in the *[cluster]* section.

Hint!

You don't have to create the recovery lock file, it will be created by CTDB on the first start of the CTDB-daemon

The file script.options

All the service CTDB will provide will be started via special scripts. In this file you can set options to the script. An example is shown in the script. There is as section for the service-script 50.samba.options named *CTDB_SAMBA_SKIP_SHARE_CHECK* this option by default is set to *yes*. This means, every time you create a new share CTDB will check if the path exists, if not CTDB will stop. But if you use the *vfs-module glusterfs* you will have no local path in the share-configuration. The share points to a directory on your gluster-volume, so CTDB can‘t check the path. So if you going to use *glusterfs* you must set this option for Samba to *no*. Because you can set all options to all service-scripts in this file, you don‘t have to change any of the service-scripts. You will find more information on all options in the manpage `man ctdb-script.options`.

The file nodes

CTDB must know all hosts belonging to it‘s cluster, in this file you have to put all IPs from the heartbeat network of all nodes. This file must have the same content on all nodes. Here we just put the two IPs from our two nodes into the file. In listing 3.2 you see the content of the file. In most distributions the file does‘n exists, you have to create it:

```
192.168.57.42
192.168.57.43
```

Listing 3.2: Content of the file nodes

The file public_addresses

Every time CTDB starts it will provide an IP-address to all nodes in the CTDB-Cluster, this must be an IP-address from the production network.

After starting the cluster, CTDB will take care of those IP-addresses and will give an IP-address of this list to every CTDB-node. If a CTDB-node crashes CTDB will assign the IP-address, from the crashed node, to another CTDB-node. So every IP-address from this file is always assigned to on of the nodes.

CTDB is doing the failover for the services. If one node fails the IP-address will switch to one of the remaining nodes. All clients will then reconnect to this node. That‘s possible because all nodes have all session-information of all clients.

For each node you need a public_addresses-file. The files can be different on the nodes, depending to which subnet you would like to assign the node. Here we just have one subnet so both nodes have identical public_addresses-files. Listing 3.3 is shown the content of the file:

```
192.168.56.101/24 enp0s8
192.168.56.102/24 enp0s8
```

Listing 3.3: The file public_addresses

Hint!

Responsible for assigning the dynamic IP-address from the file public_address is the program `ip addr`, but you also need the program `ethtool` to check the state of a link. So always check that the program is installed.

3.1 Starting CTDB the first time

Now you have configured the CTDB-service on both nodes, then you will be ready for the first start. To see what will happened during the start you can open another terminal and start `tail -f /var/log/ctdb/ctdb.log` to see the messages. First start one node with `systemctl restart ctdb`, look at the log-messages and then start the second node and still keep an eye on the log.

Starting the first node will show the messages from listing 3.1.1

```
2020/02/11 17:32:53.778637 ctdbd[1926]: monitor event OK - node re-enabled
2020/02/11 17:32:53.778831 ctdbd[1926]: Node became HEALTHY. Ask recovery \
                           master to reallocate IPs
2020/02/11 17:32:53.779152 ctdb-recoverd[1966]: Node 0 has changed flags \
                           - now 0x0  was 0x2
2020/02/11 17:32:54.575970 ctdb-recoverd[1966]: Unassigned IP 192.168.56.102 \
                           can be served by this node
2020/02/11 17:32:54.576047 ctdb-recoverd[1966]: Unassigned IP 192.168.56.101 \
                           can be served by this node
2020/02/11 17:32:54.576254 ctdb-recoverd[1966]: Trigger takeoverrun
2020/02/11 17:32:54.576780 ctdb-recoverd[1966]: Takeover run starting
2020/02/11 17:32:54.594527 ctdbd[1926]: Takeover of IP 192.168.56.102/24 on \
                           interface enp0s8
2020/02/11 17:32:54.595551 ctdbd[1926]: Takeover of IP 192.168.56.101/24 on \
                           interface enp0s8
2020/02/11 17:32:54.843175 ctdb-recoverd[1966]: Takeover run completed \
                           successfully
```

Listing 3.1.1: Log on node 0 starting the node

Here you can see, that the node has taken both dynamic IP-addresses, you can check this with `ip a l enp0s8`.

Important!

Don‘t use `ifconfig` to list the IP-addresses, `ifconfig` will not list the dynamically assigned IP-addresses.

Before you start the second node, take a look at the CTDB-status with `ctdb status`. You will see – as in listing 3.1.2 – that the first node you have just started has the status *OK*, the other node has the status *DISCONNECTED|UNHEALTHY|INACTIVE*.

```
root@cluster-01:~# ctdb status
Number of nodes:2
pnn:0 192.168.57.42    OK (THIS NODE)
pnn:1 192.168.57.43    DISCONNECTED|UNHEALTHY|INACTIVE
Generation:1636031659
Size:1
hash:0 lmaster:0
Recovery mode:NORMAL (0)
Recovery master:0
```

Listing 3.1.2: Status after starting first node

Now you can start CTDB on the second node with `systemctl restart ctdb`. Inside the log on the first node you will see the message that the takeover was successfully. In listing 3.1.3 you will see the last lines from the log:

```
2020/02/11 17:51:49.964668 ctdb-recoverd[6598]: Takeover run starting
2020/02/11 17:51:50.004374 ctdb-recoverd[6598]: Takeover run completed \
                           successfully
2020/02/11 17:51:59.061780 ctdb-recoverd[6598]: Reenabling recoveries \
                           after timeout
2020/02/11 17:52:04.632267 ctdb-recoverd[6598]: Node 1 has changed flags \
                           - now 0x0  was 0x2
2020/02/11 17:52:04.989395 ctdb-recoverd[6598]: Takeover run starting
2020/02/11 17:52:05.008763 ctdbd[6554]: Release of IP 192.168.56.102/24 \
                           on interface enp0s8  node:1
2020/02/11 17:52:05.154588 ctdb-recoverd[6598]: Takeover run completed \
                           successfully
```

Listing 3.1.3: Log-entries starting the second node

If you see a lot of messages as in listing 3.1.4, check if the gluster-volume is mounted correctly and if the *recovery lock*-option in /etc/ctdb/ctdb.conf is set correctly:

```
2020/02/11 17:51:00.883523 ctdbd[6554]: CTDB_WAIT_UNTIL_RECOVERED
2020/02/11 17:51:00.883630 ctdbd[6554]: ../../ctdb/server/ctdb_monitor.c:324 \
                           wait for pending recoveries to end. Wait \
                           one more second.
```

Listing 3.1.4: Recovery error

A look at the status will show both nodes *OK*, as you can see in listing 3.1.5:

```
root@cluster-01:~# ctdb status
Number of nodes:2
pnn:0 192.168.57.42    OK (THIS NODE)
pnn:1 192.168.57.43    OK
Generation:101877096
Size:2
hash:0 lmaster:0
hash:1 lmaster:1
Recovery mode:NORMAL (0)
Recovery master:0
```

Listing 3.1.5: Status after both nodes started

Whenever you stop a node, the other node will takeover the IP-address assigned by CTDB. Now CTDB is running, but you still have no service configured. You should only continue configuring the services if both nodes are healthy.

As a test you can stop CTDB on one node and you will see, that one of the other nodes will get the IP from the stopped node. As soon as you restart the node, the host which took the former IP-address, will give any of the IP-address back to the node and all nodes will have the status *OK* again.

The next step will be setting up samba.

4 Configure Samba to work with CTDB

In this part we will set up a Samba-cluster to provide fileservices to Windows- and Linux-clients. We will join the cluster into a Samba Active Directory-domain and create shares. We will use three different techniques to serve the shares to the clients:

- Using the local mounted gluster-volume to create the share.
- Using the vfs-module *glusterfs* to directly point to the volume on the gluster-cluster without mounting the volume on the local system.
- Using the new vfs-module *glusterfs_fuse*.

4.1 Joining the domain

To join the cluster to the domain you have to do the following steps:

- Creating the DNS-records
- Configure Samba via registry
- Join the cluster

4.1.1 Creating the DNS-records

To join the domain, the first step should be creating the DNS-entries for the cluster. In this tutorial I already set up a Samba-domain on host *addc01*. I used the internal DNS-Server. In your production environment you should use *bind9* as DNS-server.

The reason is, that bind9 provides DNS-round-robin. With DNS-round-robin you have a loadbalancing for your CTDB-cluster. Creating the DNS-records for the cluster will be the same no mater which DNS-server you use. In listing 4.1.1.1 you will see all commands needed to create the reversezone and all DNS-records:

```
root@addc-01:~# kinit administrator
administrator@EXAMPLE.NET's Password:********

root@addc-01:~# samba-tool dns zonecreate addc-01 56.168.192.in-addr.arpa -k yes
Zone 56.168.192.in-addr.arpa created successfully

root@addc-01:~# systemctl restart samba-ad-dc

root@addc-01:~# samba-tool dns add addc-01 example.net cluster A 192.168.56.101
Record added successfully

root@addc-01:~# samba-tool dns add addc-01 example.net cluster A 192.168.56.102
Record added successfully

root@addc-01:~# samba-tool dns add addc-01 56.168.192.in-addr.arpa 101 PTR \
                cluster.example.net
Record added successfully

root@addc-01:~# samba-tool dns add addc-01 56.168.192.in-addr.arpa 102 PTR \
                cluster.example.net
Record added successfully

root@addc-01:~# host cluster
cluster.example.net has address 192.168.56.101
cluster.example.net has address 192.168.56.102

root@addc-01:~# host 192.168.56.101
```

```
101.56.168.192.in-addr.arpa domain name pointer cluster.example.net.

root@addc-01:~# host 192.168.56.102
102.56.168.192.in-addr.arpa domain name pointer cluster.example.net.
```

Listing 4.1.1.1: Creating DNS-entries

In the listing you see that both dynamic IP-addresses getting the same DNS-name. Resolving the hostname will give you both IP-addresses. So a client can connect to either of the addresses and will always reach one of the cluster nodes.

Hint!

A client always connect to the cluster and not to a special host.

4.1.2 Configuring Samba

If you use Samba together with CTDB you have to use the *registry* to configure Samba. The reason is, that you configure the cluster and not every single host. All configurations of the cluster can be done on either one of the Samba-hosts. The CTDB-Samba-hosts will share all TDB-files. The files will be stored locally but will be committed between all nodes.

The easiest way to write the settings into the registry is to write a file in smb.conf-style and the import the file into the registry. If you start Samba, the first place where Samba looks for any configuration is the smb.conf file. This is the reason why you always have a minimum smb.conf. In listing 4.1.2.1 you will see the settings for the configuration. You don't have to type the file, you will find the file on *cluster01* in /daten the name of the file is smb.conf.first:

```
[global]
        workgroup = EXAMPLE
        realm = EXAMPLE.NET
        netbios name = CLUSTER
        security = ADS
        template shell = /bin/bash
        winbind use default domain = Yes
        winbind refresh tickets = Yes
        idmap config *:range = 10000-19999
        idmap config example:range = 1000000-1999999
        idmap config example:backend = rid
```

Listing 4.1.2.1: First configuration for Samba

The next step is to create the smb.conf to tell Samba that clustering should be used and all the configuration comes out of the registry. In listing 4.1.2.2 you find the content of the file:

```
[global]
        clustering = yes
        include = registry
```

Listing 4.1.2.2: Content of smb.conf

Important!

Before you join the cluster make sure that you are using the domain controller as DNS-Server. Check the /etc/resolv.conf for the IP-address of your domain controller, on both nodes

You will find the file on both nodes in the /daten. Copy the file on both nodes to /etc/samba. Now import the configuration file into the registry with the command `net conf import /daten/smb.conf.first`. You can test the import with `net conf list` on both nodes. Test all settings with `testparm` on both nodes. If you got no error message you can join the cluster into the domain as you can see in listing 4.1.2.3:

```
root@cluster-01:/etc/ctdb# net ads join -U administrator
Enter administrator's password: *******

Using short domain name -- EXAMPLE
Joined 'CLUSTER' to dns domain 'example.net'
Not doing automatic DNS update in a clustered setup.

root@cluster-01:~# net ads testjoin
Join is OK
```

Listing 4.1.2.3: Joining the cluster

A DNS-update for a cluster is not possible, that's why we created the DND-records before we joined the cluster, but as you can see with `net ads testjoin` the join is valid. Another test is to see if the account for the cluster was created in AD to do the test do a `samba-tool computer list` on your domain controller.

Now you have joined the cluster to the domain. Up to this point you haven't started any of the Samba-services. Starting and stopping the services *smbd*, *nmbd* and *winbind* should be done by CTDB and not via *systemd*. So before you you configure CTDB to take care of the services you have to stop and disable the services in systemd as you can see in listing 4.1.2.4:

```
root@cluster-01:~# systemctl stop smbd nmbd winbind
root@cluster-01:~# systemctl disable smbd nmbd winbind

root@cluster-02:~# systemctl stop smbd nmbd winbind
root@cluster-02:~# systemctl disable smbd nmbd winbind
```

Listing 4.1.2.4: Stopping and disable Samba-servcies

As you can see in the listing, you have to do it on both nodes.

4.2 Configure CTDB to take care of Samba-services

After you have done the configuration of Samba and you joined the cluster into your domain, you can start configuring of CTDB to take care of the Samba-services. Since Samba 4.9 the way to configure CTDB has changed a lot, up to this point you had only noticed this while you where configuring the *recovery lock*. The old way to configure the services was activating the services in the ctdb.conf-file. Starting with Samba 4.9 that has changed, now you have a command to activated the services.

Every service will be enabled via an *event script*. To enable the scripts you use the command `ctdb`. We need the Samba-script and the winbind-script. In listing 4.2.1 you see the commands to enable the scripts:

```
root@cluster-01:~# ctdb event script enable legacy 50.samba
root@cluster-01:~# ctdb event script enable legacy 49.winbind

root@cluster-02:~# ctdb event script enable legacy 50.samba
```

```
root@cluster-02:~# ctdb event script enable legacy 49.winbind
```

Listing 4.2.1: Enable the event scripts

After enabling both – *50.samba* and *49.winbind* – restart CTDB with `systemctl restart ctdb` on both nodes. After a while both CTDB-nodes become *Ok* again. Checking with `ps` and `ss -tlpn` you will see, that the Samba-services are running. Now you have configured CTDB to take care of starting and stopping all Samba-services.

What will happen when you execute the command? All event scripts are located in /usr/share/ctdb/events/legacy if you activate a script, the script will be linked to /etc/ctdb/events/legacy/. If you take a look at the script, you will see, it looks like a init-script of a service – that's all it is.

4.3 Checking the CTDB-cluster

After your CTDB-cluster is running and the Samba-services are active let‘s take a look at some CTDB-tests.

4.3.1 ctdb status

Shows the actual status of the whole cluster. All nodes are listed and you can see the status of all nodes.

Tip!

If you restart one or more nodes do a `watch ctdb status` so you will see all two seconds a refreshed list of the status.

In listing 4.3.1.1 you see the result of `ctdb status`:

```
root@cluster-02:~# ctdb status
Number of nodes:2
pnn:0 192.168.57.42    OK
pnn:1 192.168.57.43    OK (THIS NODE)
Generation:1823539022
Size:2
hash:0 lmaster:0
hash:1 lmaster:1
Recovery mode:NORMAL (0)
Recovery master:1
```

Listing 4.3.1.1: Output of ctdb status

What you will see here:

- The number of nodes in the cluster.
- A list of all nodes *pnn*-number the IP-address and the status.
- *Generation* is just a number which changes if cluster recovery is taking place. There is no special meaning.
- The number of active nodes in the cluster – here 2 nodes.

- The line *hash:*< *n* > *lmaster:*< *n* > is used to calculate the *lmaster*, it's calculated via a hash value.
- The *Recovery mode* shows if everything in the cluster is good *NORMAL*, or a recovery is taking place *RECOVERY*.
- The *Recovery master* is the node which is responsible for a recovery.

If you just want to see the status of one, or some hosts you can use `ctdb nodestatus`. With *nodestatus* you can see the status of the actual node or a list of all hosts as you can see in listing 4.3.1.2:

```
root@cluster-01:~# ctdb nodestatus
pnn:0 192.168.57.42    OK (THIS NODE)

root@cluster-01:~# ctdb nodestatus 1
pnn:1 192.168.57.43    OK

root@cluster-01:~# ctdb nodestatus 0,1
pnn:0 192.168.57.42    OK (THIS NODE)
pnn:1 192.168.57.43    OK

root@cluster-01:~# ctdb nodestatus all
Number of nodes:2
pnn:0 192.168.57.42    OK (THIS NODE)
pnn:1 192.168.57.43    OK
```

Listing 4.3.1.2: ctdb nodestatus

There are some more possibilities to check the cluster, take a look at listing 4.3.1.3:

```
root@cluster-01:~# ctdb uptime
Current time of node 0        :                Wed Feb 12 18:56:14 2020
Ctdbd start time              : (000 04:53:49) Wed Feb 12 14:02:25 2020
Time of last recovery/failover: (000 04:53:43) Wed Feb 12 14:02:31 2020
Duration of last recovery/failover: 0.573115 seconds

root@cluster-01:~# ctdb listnodes
192.168.57.42
192.168.57.43

root@cluster-01:~# ctdb ping
response from 0 time=0.000124 sec  (20 clients)

root@cluster-01:~# ctdb ip
Public IPs on node 0
192.168.56.101 1
192.168.56.102 0
```

Listing 4.3.1.3: Some more tests

For more information about testing a CTDB-cluster see the manpage of `ctdb`.

4.3.2 Check all services

The next important check is to see which services can CTDB provide and which services are actually configured. In many documentations you find the command `ctdb scriptstatus`

to list all the running services, but this command is deprecated. Since Samba 4.9 you should check the event scripts with `ctdb event status legacy monitor`. Then you will see all the running (monitored) services. To see the list of all services do a `ctdb event script list legacy`. In listing 4.3.2.1 you will see both commands:

```
root@cluster-01:~# ctdb event status legacy monitor
00.ctdb              OK         0.005 Thu Feb 13 18:28:35 2020
01.reclock           OK         0.023 Thu Feb 13 18:28:35 2020
05.system            OK         0.017 Thu Feb 13 18:28:35 2020
10.interface         OK         0.019 Thu Feb 13 18:28:35 2020
49.winbind           OK         0.011 Thu Feb 13 18:28:35 2020
50.samba             OK         0.101 Thu Feb 13 18:28:35 2020

root@cluster-01:~#  ctdb event script list legacy
* 00.ctdb
* 01.reclock
* 05.system
  06.nfs
* 10.interface
  11.natgw
  11.routing
  13.per_ip_routing
  20.multipathd
  31.clamd
  40.vsftpd
  41.httpd
* 49.winbind
* 50.samba
  60.nfs
  70.iscsi
  91.lvs
```

Listing 4.3.2.1: List the CTDB-services

The first command is showing all running services. The second command is showing all services CTDB can provide, all running services are marked with a *.

4.3.3 The command onnode

The command `onnode` can be used to execute a command on all, or some, nodes of the cluster. To get `onnode` running without being ask for a password, on each host create a ssh-key without passphrase with `ssh-keygen` for the user *root* and copy it to all nodes. To copy the key to all nodes, including the own host, user the command `ssh-copy-id`. You need the key on the localhost too, because all commands will be executed via ssh. To copy an ID from node *c-01* to *root@c-02* execute `root@cluster-01 ssh-copy-id -i .ssh/id_rsa.pub root@c-02`. After you copied all keys, try if you can login without password.

After you managed to login without password do the test from listing 4.3.3.1:

```
root@cluster-02:~# onnode all ctdb nodestatus

>> NODE: 192.168.57.42 <<
pnn:0 192.168.57.42    OK (THIS NODE)

>> NODE: 192.168.57.43 <<
```

```
pnn:1 192.168.57.43    OK (THIS NODE)
```

Listing 4.3.3.1: First test with onnode

You see here, that the command `ctdb nodestatus` will be executed on all nodes in the cluster. Instead of *all* you can give a node-number or a comma separated list of node-numbers, then the command will be executed only on this nodes.

You can also restart services on all nodes so the command *onnode all systemctl restart ctdb* would restart the CTDB on all nodes. You can execute any command via `onnode`.

One more useful thing you can do with onnode – you can copy files out of the storage into a directory of all nodes. A good example: You want to add a new node to your CTDB-cluster so you have to add a new line to the files nodes and public_addresses, so the easiest way to do so is creating the needed file in /glusterfs on one node where the gluster-volume is mounted. Then copy the file as you can see in listing 4.3.3.2:

```
root@cluster-01:~# vi /glusterfs/nodes
...
root@cluster-01:~# onnode all cp /glusterfs/nodes /etc/ctdb

>> NODE: 192.168.57.42 <<

>> NODE: 192.168.57.43 <<
```

Listing 4.3.3.2: Copy files with onnode

If you would do the same with public_addresses you would have added another node to your CTDB-cluster.

Important!

Always write the new IP-address at the end of the list in both files

5 Creating shares

To make a CTDB-cluster a fileserver you have to create shares, so that your users can store their files on the cluster. Together with Gluster you have three ways to create shares:

- The classic way (the gluster-volume is mounted via the *glusterfs-client*)
- Using the VFS-module *glusterfs*
- Using the VFS-module *glusterfs_fuse*

5.1 The classic way

You can mount the gluster-volume via the *glusterfs-client* to all CTDB-nodes. Then create the share via a local path. In this tutorial we have both, the glusterfs-server and the CTDB-server, on the same machine, but we mounted the gluster-volume via the glusterfs-client with the systemd mount-script. So it's easy to create the share.

But remember, because we use the clustermode on the Samba-server you must configure the share (like all other shares) over the registry.

A simple way to change the registry is exporting the registry with `net conf list > registry.conf`. With this command you export the registry in a smb.conf-style file. Then you define the share in the file and then import the file with `net conf import registry.conf`. Listing 5.1.1 is showing all steps:

```
root@cluster-01:~# net conf list > registry.conf

root@cluster-01:~# mkdir /glusterfs/localshare

root@cluster-01:~# vi registry.conf

--- start append to file ----
[localshare]
        inherit acls = Yes
        path = /glusterfs/localshare
        read only = No
        vfs objects = axl_xattr
--- end append to file   ----

root@cluster-01:~# net conf import registry.conf
```

Listing 5.1.1: Creating a share with local mounted gluster-volume

Important!

Remember to create the directory before you import the file.

You don‘t have to restart the cluster after you have created the new share. Use `smbclient -L cluster` to list all shares of the cluster and you will see the new share as in listing 5.1.2:

```
root@cluster-01:~# smbclient -L cluster
Enter EXAMPLE\root's password:
Anonymous login successful

        Sharename       Type      Comment
        ---------       ----      -------
        IPC$            IPC       IPC Service (Samba 4.11.6-Debian)
        localshare      Disk
SMB1 disabled -- no workgroup available
```

Listing 5.1.2: List the new local share

5.2 Using the glusterfs VFS-module

You can directly access the gluster-volume without mounting the gluster-volume to a local mountpoint on the CTDB-nodes. If you want to use the glusterfs-module you first have to deactivate the foldercheck in CTDB. Edit the file /etc/ctdb/script.options on all nodes as you can see in listing 5.2.1:

```
# 50.samba.options
CTDB_SAMBA_SKIP_SHARE_CHECK=yes
```

Listing 5.2.1: Disable the directory-check

After changing the settings on all nodes, restart CTDB.

Tip!

Use the command onnode to spread the new file to all nodes and to restart CTDB

Now create a new folder in your gluster-volume. (remember it‘s only here during the tutorial that the CTDB-server and the Gluster-Server on the same host). Then, again export the registry into a file and add the changes to the end of the file. You see all entries in listing 5.2.2:

```
root@cluster-01:~# mkdir /glusterfs/cfs-share
root@cluster-01:~# vi registry.conf

--- start append to file ----
[vfs-share]
        read only = no
        vfs objects = acl_xattr glusterfs
        glusterfs:volume = gv0
        glusterfs:logfile = /var/log/samba/glusterfs-gv1.log
        glusterfs:loglevel = 7
        glusterfs:volfile_server = cluster.example.net
        kernel share modes = no
        path = /vfs-share
--- end append to file   ----

root@cluster-01:~# net conf import registry.conf
```

Listing 5.2.2: Share with glusterfs_vfs

Here you can see, that there is no local *path*-variable set, just a path on the gluster-volume. You must giver the path relative to the volume-mountpoint on any of your gluster-nodes. After importing the changes you will see the new share with `smbclient -L cluster`.

Note!

Always put the VFS-module *glusterfs* ad the end of the *vfs objects*-parameter. If yo don‘t, the share will not work.

After you have written the changes to the registry you can look at the list of the share with `smbclient`. Listing 5.2.3 is showing the new list of shares:

```
root@cluster-01:~# smbclient -L cluster
Enter EXAMPLE\root's password:
Anonymous login successful

        Sharename       Type      Comment
        ---------       ----      -------
        IPC$            IPC       IPC Service (Samba 4.11.6-Debian)
        localshare      Disk
        vfs-share       Disk
SMB1 disabled -- no workgroup available
```

Listing 5.2.3: The new list of shares

5.3 Using the glusterfs fuse module

The third way to create a share together with gluster is using the new GLUSTERFS_FUSE-module. The *glusterfs_fuse* needs a mouted gluster-volume via glusterfs-client, so the share will point to a local path, but the module provides support for the get_real_filename-VFS call which enhances file access performance by avoiding multiple expensive case folding lookup calls to detect the appropriate case of an exisiting filename.

In listing 5.3.1 you see the configuration of a new share with glusterfs_fuse:

```
[fuse-share]
        path = /glusterfs/fuse-share
        read only = no
        vfs objects = acl_xattr glusterfs_fuse
        inherit acls = yes
```

Listing 5.3.1: Creating a share with glusterfs_fuse

Important!

Don‘t forget to create the directory for the share.

You can combine the *glusterfs_fuse*-module with any other vfs-module, but the glusterfs_fuse must be the last in the list.

After you created the directory and did an import of the changed registry you will see the share in the list of `smbclient` as in listing 5.3.2:

```
root@cluster-01:~# smbclient -L cluster
Enter EXAMPLE\root's password:
Anonymous login successful

        Sharename       Type      Comment
        ---------       ----      -------
        IPC$            IPC       IPC Service (Samba 4.11.6-Debian)
        localshare      Disk
        vfs-share       Disk
        fuse-share      Disk
SMB1 disabled -- no workgroup available
```

Listing 5.3.2: The new glusterfs_fuse share

Which way to use?

I know your next question: Which way should I use to create may shares when using GlusterFS as a storage? The answer is: It depends :-). Why? I have many customers using GlusterFS together with CTDB and Samba- or Windows-clients and I set up differet ways for the shares. On shares with big files (> 20MB) I got the best result using the glusterfs-module. On shares with many smale files (<100KB) and more then 20k of files in one directory using the new *glusterfs_fuse* gave the best result. So you should test for your self what is the best way to use.

Now you have a file-cluster with GlusterFS and CTDB as a loadbalanced, failover system.

Index