Configuring a High Availability Database with QVD

QVD DOCUMENTATION

<documentation@theqvd.com>

November 5, 2018

Contents

Revision Information

Objective

With this article we aim to create a Highly Available (HA) database for use with QVD, and to configure QVD itself to provide failover capability. Furthermore, we aim to impart some practical tips about maintaining and backing up such a system.

Part I

Implementation

Chapter 1

Prerequisites

You will need at least two nodes running SUSE Linux Enterprise Server (SLES) SP2 along with the SUSE Linux High Availability Extension (HAE) SP2. These may optionally be virtualised.

The nodes should be linked together by both a regular IPv4 network, and by crossover cable. If you are using a test environment, two IPv4 network links will work fine but it is **strongly** recommended to use a crossover link to avoid a single point of failure.

It should be possible to follow this guide using Ubuntu, however that is not covered here.

Chapter 2

Networking

Network Interfaces

Each node should have two interfaces with corresponding subnets. We will use the following:

```
qvddb1 eth0 172.20.64.91/24 and 192.168.0.91/24
qvddb2 eth0 172.20.64.92/24 and 192.168.0.92/24
```

Note
The latter two interfaces are assumed to be linked together by crossover cable.

Additionally, postgres will need its own cluster IP address, in this instance set to `172.20.64.90`.

Hostname Resolution

Add each link to the `hosts` file in all nodes, to ensure that they can be reached by hostname without DNS:

```
172.20.64.90    qvddb    # The postgres IP to which other hosts will connect
172.20.64.91    qvddb1   # IP Link
172.20.64.92    qvddb2   # IP Link
192.168.0.91    qvddb1cx # Crossover link
192.168.0.92    qvddb2cx # Crossover link
```

Chapter 3

Hardware

The nodes should have a minimum of 1GB of RAM and enough space to accommodate your database space requirements. This space **must** be on a separate partition (logical or physical) to the host OS, we assume a separate drive altogether.

Chapter 4

NTP

Configure each node to sync with an NTP server, to ensure the correct time on each resource. Use a local NTP server if you have one, or a publicly available one if not:

```
# echo "server clock.isc.org" >> /etc/ntp.conf
# chkconfig ntp on
# rcntp start
```

Now verify that ntp is using this new peer:

```
# ntpq -pn
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
 127.127.1.0     .LOCL.          10 l   47   64   77    0.000    0.000   0.001
*149.20.64.28    .GPS.            1 u   46   64   73  197.563   -2.471   1.292
```

Chapter 5

Software Installation

You will need to install the following software:

- Postgresql Server http://www.postgresql.org/
- DRBD http://www.drbd.org/
- Pacemaker http://www.linux-ha.org/wiki/Pacemaker
- mailx http://heirloom.sourceforge.net/mailx.html
- ntp http://www.ntp.org/

Install the above using zypper:

```
# zypper install sleha-bootstrap drbd drbd-pacemaker yast2-drbd \
postgresql-server postgresql postgresql-contrib mailx xfsprogs xfsdump ntp
```

Chapter 6

Cluster Configuration

Configure DRBD on both nodes in the file `/etc/drbd.conf`:

```
global {
    usage-count no;
}
common {
    syncer { rate 100M; }
    protocol      C;
}
resource postgres {
    startup {
       wfc-timeout 0;
       degr-wfc-timeout
       120;
    }
    disk { on-io-error detach; }
    on qvddb1 {
       device      /dev/drbd0;
       disk        /dev/sdb;
       address     192.168.0.91:7791;
       meta-disk   internal;
    }
    on qvddb2 {
       device      /dev/drbd0;
       disk        /dev/sdb;
       address     192.168.0.92:7791;
       meta-disk   internal;
    }
}
```

Here we refer to the resource that DRBD will manage as `postgres`, and identify the storage (partition or disk) as `disk` (in this instance `/dev/sdb`). `syncer` refers to the rate of transfer between the resources.

Now create the metadata on both resources as follows:

```
# drbdadm create-md postgres
```

And set the resources as up, again on both nodes:

```
# drbdadm up postgres
```

From **one** node only, initiate a sync between the devices:

```
# drbdadm -- --overwrite-data-of-peer primary postgres
```

This may take some time depending on the size of the partition. Progress can be monitored by concatenating `/proc/drbd`, or using the command `watch /etc/init.d/drbd status`. In due course, the nodes are eventually marked as `UpToDate`:

```
# cat /proc/drbd
version: 8.4.1 (api:1/proto:86-100)
GIT-hash: 91b4c048c1a0e06777b5f65d312b38d47abaea80 build by phil@fat-tyre, 2011-12-20  ↩
    12:43:15
 0: cs:Connected ro:Primary/Secondary ds:UpToDate/UpToDate C r-----
    ns:0 nr:0 dw:0 dr:664 al:0 bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:b oos:0
```

Chapter 7

Create the Postgres Partition

On both nodes, prepare the postgres data folder:

```
# mv /var/lib/pgsql/data /var/lib/pgsql/data.orig
# mkdir /var/lib/pgsql/data
# chown postgres:postgres /var/lib/pgsql/data
# chmod 700 /var/lib/pgsql/data
```

Prepare the postgres partition (here we will use XFS for its performance capabilities. Then mount the partition temporarily, and copy the original postgres data folder onto the partition. On the first node only:

```
# mkfs.xfs -f /dev/drbd0
# mount /dev/drbd0 /mnt
# cd /var/lib/pgsql/data.orig/
# tar cf - . | tar xf - -C /mnt
# umount /mnt
# mount /dev/drbd0 /var/lib/pgsql/data
```

Finally, start the postgres server:

```
# /etc/init.d/postgresql start
```

Chapter 8

Set Up and Configure Postgresql

Add the QVD database user

Please refer to the QVD Administration Manual for full details of a QVD database configuration. For now, `su` to postgres and add your qvd user and database:

```
# su - postgres
postgres@qvddb1:~> createuser -SDRP qvd
Enter password for new role:
Enter it again:
postgres@qvddb1:~> createdb -O qvd qvddb
```

Configure Postgresql

Now edit the file `/var/lib/pgsql/data/postgresql.conf` and set the transaction level as serializable and set the server to list on all addresses:

```
default_transaction_isolation = 'serializable'
listen_addresses = '*'
```

Now, in `/var/lib/pgsql/data/pg_hba.conf`, add md5 authentication for each node and the postgres and cluster ip addresses:

```
host  qvddb qvd 172.20.64.90/32  md5
host  qvddb qvd 172.20.64.91/32  md5
host  qvddb qvd 172.20.64.92/32  md5
```

Restart postgres:

```
# /etc/init.d/postgresql restart
```

Initialise the QVD Database

Create the node configuration in `/etc/qvd/node.conf` as follows:

```
database.host = qvddb
database.name = qvddb
database.user = qvd
database.password = changeme
```

Add the postgres ip address temporarily to eth0 on the first node, for the purposes of deployment:

```
# ip addr add 172.20.64.90/24 dev eth0
```

Ensure the password matches the one you used earlier when creating the qvddb user. Now, run the `/usr/lib/qvd/bin/qvd-deploy-db.pl` script which will create the QVD database:

```
# /usr/lib/qvd/bin/qvd-deploy-db.pl
```

Create a table for testing

Create a test table for verification / maintenance purposes. (Refer to the maintenance chapter later on). Use the same user and password as the one you have used for the HA setup, bearing in mind that this may have security implications:

```
# psql -h qvddb1 -U qvd qvddb
Password for user qvd:
qvddb=> create table pacemaker_monitor ( last_update timestamp );
CREATE TABLE
qvddb=> insert into pacemaker_monitor VALUES (current_timestamp);
INSERT 0 1
qvddb=> select * from pacemaker_monitor;
        last_update
 ----------------------------
 2013-02-24 00:22:17.902352
(1 row)
qvddb=>
```

Verify postgresql on the second node

Before you begin to manage the services with Pacemaker, check that postgres works on the second node. On the first node, stop postgresql and unmount the DRBD device:

```
# /etc/init.d/postgresql stop
# umount /dev/drbd0
```

Now, demote the first node:

```
# drbdadm secondary postgres
```

Promote the second node and mount the DRBD device:

```
# drbdadm primary postgres
# mount /dev/drbd0 /var/lib/pgsql/
```

Start postgresql on the second node:

```
# /etc/init.d/postgresql start
```

Ensure you can log in to the second node using the credentials that you set earlier. That done, it is time to stop all services to configure corosync:

On the second node:

```
# /etc/init.d/postgresql stop
# umount /dev/drbd0
# drbdadm secondary postgres
# /etc/init.d/drbd stop
```

And on the first node:

```
# drbdadm primary postgres
# /etc/init.d/drbd stop
```

Disable postgresql and DRBD from starting at initialization (on both nodes):

```
# chkconfig --level 35 drbd off
# chkconfig --level 35 postgresql off
```

Chapter 9

Configure Corosync

Corosync is a *Group Communication System* that provides a messaging layer for your cluster.

On the first node, edit `/etc/corosync/corosync.conf` and add the second interface and set rrp_mode to active:

```
totem {
...
  rrp_mode: active
  interface {
            ringnumber: 0
            bindnetaddr: 192.168.0.0
            mcastaddr: 239.44.32.17
            mcastport: 5405
            ttl: 1
            }
  interface {
            ringnumber: 1
            bindnetaddr: 172.20.64.0
            mcastaddr: 239.44.32.18
            mcastport: 5405
            ttl: 1
            }
}
...
```

Sync the config files and run the `csync2` command on the first node:

```
# scp /etc/corosync/* qvddb2:/etc/corosync/
# csync2 -xv
```

Verify the status on each node:

```
# corosync-cfgtool -s
Printing ring status.
Local node ID 1530926252
RING ID 0
     id = 172.20.64.91
     status = ring 0 active with no faults
RING ID 1
     id = 192.168.0.91
     status = ring 1 active with no faults
```

Corosync can be started on both nodes using the following command:

```
# /etc/init.d/openais start
```

Chapter 10

Cluster Resources

The following resources will be defined:

- Fence nodes (in our config we use libvirt, check the one that you need)
- Create a ping resource (to allow the cluster only to run where it has access to the router)
- Create cluster group:
  - DRBD (master / slave resource)
  - Filesystem resource
  - IP resource
  - Postgres resource
- Create a mail resource

Stop DRBD and Postgresql

On both nodes, stop the DRBD and postgresql services:

```
# /etc/init.d/postgresql stop
# umount /var/lib/pgsql/data
# /etc/init.d/drbd stop
```

Set the Cluster Defaults

Set the cluster recheck interval to 10 minutes:

```
# crm(live)configure# property cluster-recheck-interval="10min"
```

Set resource defaults *stickiness* so that it is not possible to move resources between nodes:

```
crm(live)configure#  rsc_defaults resource-stickiness="1"
```

Restart a resource locally once, and move to another node on the second failure:

```
crm(live)configure#  rsc_defaults migration-threshold="2"
```

Expire all failure counts after 3 minutes:

```
crm(live)configure# rsc_defaults failure-timeout="3min"
```

Create Fence Resources

Fencing is used to isolate problem hosts from the rest of the cluster. Particularly of concern is a situation whereby nodes lose contact with each other but continue to act at though they are the primary resource, leading to potentially irreparable data corruption. This is commonly known as a split-brain condition.

For the purposes of this setup, we will assume that your node are virtualised hosts and provide options for libvirt and XenAPI fencing. We will call the host system **aries** to show the examples, obviously you will need to replace this with your own host system.

libvirt Fencing

On both nodes, copy over the public ssh key of root and install `libvirt-client`:

```
# ssh-copy-id -i .ssh/id_rsa root@aries
# zypper install libvirt-client
```

In the first node, run `crm configure`:

```
# crm configure
crm(live)configure# primitive p_fence_qvddb1 stonith:external/libvirt params hostlist=" ↩
    qvddb1" \
hypervisor_uri="qemu+ssh://aries/system" op monitor interval="60"
crm(live)configure# primitive p_fence_qvddb2 stonith:external/libvirt params hostlist=" ↩
    qvddb2" \
hypervisor_uri="qemu+ssh://aries/system" op monitor interval="60"
crm(live)configure# location l_fence_qvddb1 p_fence_qvddb1 -inf: qvddb1
crm(live)configure# location l_fence_qvddb2 p_fence_qvddb2 -inf: qvddb2
crm(live)configure# property stonith-enabled=true
crm(live)configure# commit
```

Test the fencing of each node with the following command (change the node name as appropriate):

```
# crm node fence qvddb2
# crm_mon -o -f
```

XenAPI Fencing

For a Xen Server cluster, you will need `stonith-xenapi-0.5-10.1.x86_64.rpm` and `fence-agents-3.1.11-2.4.x86_64.rpm`, which you can obtain here:

http://download.opensuse.org/repositories/home:/qvd:/fence-agents/

Before installing, you will need to add the `devel:languages:python` repository, or simply download and install the `python-suds` package from there as it is a dependency of `fence-agents`:

```
# rpm -ihv http://download.opensuse.org/repositories/devel:/languages:/python/SLE_11_SP2/ ↩
    x86_64/python-suds-0.4-13.1.x86_64.rpm
```

Now, add the `fence-agents` repository, and install the packages:

```
# zypper ar "http://download.opensuse.org/repositories/home:/qvd:/fence-agents/SLE_11_SP2/" ↩
    qvd\:fence-agents
# zypper in fence-agents stonith-xenapi
```

Test the `fence_xenapi` script:

```
# fence_xenapi -s https://aries -l root -p password -v -o status -n qvddb1
```

To configure Pacemaker, add the following commands using `crm configure`:

```
crm(live)configure# primitive p_fence_qvddb1 stonith:external/xenapi params hostlist=" ↩
    qvddb1" \
session_url="https://aries" login="root" passwd="pass" op monitor interval="60"
crm(live)configure# location l_fence_qvddb1 p_fence_qvddb1 -inf: qvddb1
crm(live)configure# primitive p_fence_qvddb2 stonith:external/xenapi params hostlist=" ↩
    qvddb2" \
session_url="https://aries" login="root" passwd="pass" op monitor interval="60"
crm(live)configure# location l_fence_qvddb2 p_fence_qvddb2 -inf: qvddb2
crm(live)configure# property stonith-enabled=true
crm(live)configure# commit
```

Create a ping and mailto resource

The following two resources can now be configured:

- **ping**, so that we can later add a location constraint to run the database only where we have ping to the router
- **ping** : with a ping resource, we can later add a location constraint that will only run the database on nodes which respond to ping
- **mailto**, so that whatever happens to the cluster we receive an email. Ensure that the root alias is forwarded

```
crm(live)configure# primitive p_ping ocf:pacemaker:ping \
 params host_list="172.20.68.1" multiplier="100" \
 op start interval="0" timeout="60" \
 op stop interval="0" timeout="20" \
 op monitor interval="10" timeout="60"
crm(live)configure# clone c_ping p_ping \
 meta interleave="true"
crm(live)configure# primitive p_mailto ocf:heartbeat:MailTo \
 params email="root" \
 op start interval="0" timeout="10" \
 op stop interval="0" timeout="10" \
 op monitor interval="10" timeout="10"
crm(live)configure# commit
```

This can be tested by downing the eth0 interface which should trigger a system email and be reported by crm_mon.

Create a postgresql resource

Create four primitives:

- p_drbd_postgres: DRBD primitive (we lower the promote and demote timeouts)
- p_fs_postgres: Filesystem primitive (we lower the start and stop timeouts, and enable the filesystem check with OCF_CHECK_LEVEL, this is not strictly necessary)
- p_ip_postgres: The IP address to which the nodes connect
- p_postgres: The postgres process, we update the pacemaker_monitor table every 10 sec to verify that everything is working
- g_postgres: Start the four listed resources in order (and stop them in the reverse order)
- ms_drbd_postgres: Only one DRBD master

- l_drbd_master_on_ping: Run the DRBD master only on nodes responding to ping
- o_postgres_on_drbd: Run g_postgres where DRBD is master
- o_drbd_before_postgres: Run g_postgres where the master resource is running

Add these by going into `crm configure` and entering the following:

```
primitive p_drbd_postgres ocf:linbit:drbd \
 params drbd_resource="postgres" \
 op start interval="0" timeout="240" \
 op stop interval="0" timeout="100" \
 op promote interval="0" timeout="60" \
 op demote interval="0" timeout="30" \
 op notify interval="0" timeout="30" \
 op monitor interval="10" role="Master" timeout="20" \
 op monitor interval="20" role="Slave" timeout="20"
primitive p_fs_postgres ocf:heartbeat:Filesystem \
 params device="/dev/drbd/by-res/postgres/0" directory="/var/lib/pgsql/data" fstype="xfs" \
 op monitor interval="20" timeout="40" OCF_CHECK_LEVEL="20" \
 op start interval="0" timeout="60" \
 op stop interval="0" timeout="30" \
 op notify interval="0" timeout="60"
primitive p_ip_postgres ocf:heartbeat:IPaddr2 \
 params ip="172.20.64.90" cidr_netmask="18" nic="eth0" \
 op start interval="0" timeout="10" \
 op stop interval="0" timeout="10" \
 op monitor interval="10" timeout="20"
primitive p_postgres ocf:heartbeat:pgsql \
 params pghost="qvddb" monitor_user="qvd" monitor_password="changeme" pgdb="qvddb" \
 monitor_sql="update pacemaker_monitor set last_update=current_timestamp;" \
 op monitor interval="10" timeout="10" \
 op start interval="0" timeout="30" \
 op stop interval="0" timeout="30"
group g_postgres p_fs_postgres p_ip_postgres p_postgres
ms ms_drbd_postgres p_drbd_postgres \
 meta clone-max="2" mast-max="1" notify="true" is-managed="true"
location l_drbd_master_on_ping ms_drbd_postgres \
 rule $id="l_drbd_master_on_ping-rule" $role="Master" -inf: not_defined pingd or pingd  ↩
     number:lte 0
colocation o_postgres_on_drbd inf: g_postgres ms_drbd_postgres:Master
order o_drbd_before_postgres inf: ms_drbd_postgres:promote g_postgres:start
```

Create the file `.pgpass` in the root directory (ensuring strict permissions). The format is `hostname:port:database:username:password`:

```
qvddb:5432:qvddb:qvd:changeme
```

To verify that everything that the postgres is checking you can run:

```
watch 'psql -t -h qvddb -U qvd -d qvddb -c "select * from pacemaker_monitor;"'
```

or

```
while : ;
do
  echo -n $(date +%H:%S:)
  psql -t -hqvddb -Uqvd -d qvddb -c "select * from pacemaker_monitor;" | egrep -v '^$'
  sleep 10
done
```

Chapter 11

Test Cluster Resources

What follows are some of the tests you can perform against your cluster resources. Use the `crm_mon -o -f` command to study the effects of these tests and have the console ready.

- Reboot non active node - it should come up normally
- Reboot the active node (init 6)
- Reboot active node (echo b > /proc/sysrq-trigger)
- Restart postgres (it should restart on the same node)
- Restart postgres again (the second "failure" should mean it is now moved to next node)
- Restart postgres (should restart again on the new node)
- Restart postgres (should fail, unless you wait for failtimeout)
- Stop the disk or umount manually (create a DRBD split, see http://www.drbd.org/users-guide/s-split-brain-notification-and-recovery.html for more on this)

Chapter 12

Configure Database IP

Once you are satisfied that the cluster resources are robust, it's time to configure a database IP. This is the node-neutral address to which external services on the lan will connect.

```
crm configure primitive DBIP ocf:heartbeat:IPaddr2 \
  params ip=172.20.64.90 cidr_netmask=24 \
  op monitor interval=30s
```

Chapter 13

Configure DRBD on the Cluster

Add the DRBD resource to the cluster:

```
crm configure primitive drbd_postgres ocf:linbit:drbd \
 params drbd_resource="postgres" \
 op monitor interval="15s"
```

Configure the primary and secondary node:

```
crm configure ms ms_drbd_postgres drbd_postgres \
 meta master-max="1" master-node-max="1" \
 clone-max="2" clone-node-max="1" \
 notify="true"
```

Configure the DRBD mountpoint and the filesystem:

```
crm configure primitive postgres_fs ocf:heartbeat:Filesystem \
 params device="/dev/drbd0" directory="/var/lib/pgsql" fstype="ext3"
```

Chapter 14

Configure Postgresql on the Cluster

Add the postgresql resource to the cluster:

```
crm configure primitive postgresql ocf:heartbeat:pgsql \
 op monitor depth="0" timeout="30" interval="30"
```

Group the database IP resource (DBIP), postgresql and the DRBD partition under the name postgres:

```
crm configure group postgres postgres_fs DBIP postgresql
```

Now, set the group *postgres* to run together with DRBD Primary node:

```
crm configure colocation postgres_on_drbd inf: postgres ms_drbd_postgres:Master
```

Configuring postgres to run after DRBD:

```
crm configure order postgres_after_drbd inf: ms_drbd_postgres:promote postgres:start
```

Chapter 15

Select a Preferential Node

It's a good idea to tell Pacemaker which node you *prefer* to run the database, so set one of your nodes as preferential:

```
crm configure location master-prefer-node1 DBIP 50: qvddb1
```

Part II

Maintenance and Tuning

Chapter 16

Managing the Cluster

Migrating a resource to another node can be achieved with the following command:

```
crm resource migrate postgres qvddb2
```

Similarly, to unmigrate the node:

```
crm resource unmigrate postgres
```

Stopping the postgresql service should be done through the cluster management tools rather than through traditional means:

```
crm resource stop postgresql
```

Likewise, starting the service:

```
crm resource start postgresql
```

Chapter 17

DRBD Verification

DRBD provides online verification, but it is not enabled for resources by default. To enable it, add it to the `/etc/drbd.conf` file on both nodes, and select an algorithm from `crc32c`, `sha1`, and `md5`. We will use `crc32c` here:

```
resource postgres
  net {
    verify-alg crc32c;
  }
  ...
}
```

That done adjust the resource on both nodes.

```
# drbdadm adjust postgres
```

Now you can run `drbdadm verify` on *one* of the nodes:

```
# drbdadm verify postgres
```

Automating the verification is as simple as adding a crontab file, for example `/etc/cron.d/drbdadm-verify`:

```
# Run drbdadm verify every Saturday at 3am
00 03 * * 06 root /sbin/drbdadm verify postgres
```

Note that `drbdadm verify <resource` does *not* in itself synchronize the nodes. To do that you will need to disconnect and reconnect the resource:

```
# drbdadm disconnect postgres
# drbdadm connect postgres
```

The resources should be checked at least weekly.

Chapter 18

Postgresql Backups

The postgresql database can be dumped using the `pg_dump` command, for example:

```
# pg_dump -h drbd -U qvd -d qvddb  | gzip -c > /var/lib/pgsql/backups/qvddb-$(date +\%F-\%T ↩
    ).sql.gz
```

You are advised to back up the database at least nightly.

Chapter 19

crm Backups

The crm configuration can similarly be backed up with a simple command:

```
# crm configure show | gzip -c > /var/backup/crm-$(date +\%F-\%T).gz
```

This should be backed up nightly.

Chapter 20

QVD tuning

The following table contains the recommended timers you should set in pacemaker to work with a typical QVD setup.

Table 20.1: Recommended Pacemaker Timers

resource	**start timeout**	**stop timeout**	**monitor interval/time-out**	**promote timeout**	**demote timeout**	**total**
drbd	240	100	10/20 (master) 20/20 (slave)	60	30	...
fs	60	30	20/40	0	0	...
ip	10	10	10/20	0	0	...
pg	30	20	10/10	0	0	...
ping	20	20	10/30	0	0	...

Max timeout is the max time to stop each resource and to start them (+ check inerval of datbase) (ping does not affect):

timeout: 10 + 10 (sql) = 20 start: 60 (promote) + 60 (fs) + 10 (ip) + 30 (pg) = 160 seg stop : 30 (demote) + 30 (fs) + 10 (ip) + 20 (pg) = 90 seg

Total: 20 + 160 + 90 = 270

Chapter 21

Troubleshooting

Due to the number of components involved, problems can arise from a few different situations in a DRBD setup. Please refer to the Troublshooting section in the SUSE Linux High Availability Guide https://www.suse.com/documentation/sle_ha/singlehtml/-book_sleha/book_sleha.html#sec.ha.drbd.trouble for a selection of problematic scenarios and recommended solutions.

Part III

Resources

The following resources were used to compile this document and may provide further information:

ARTICLES

- DRBD user guide http://www.drbd.org/users-guide/
- Pacemaker Postgres HowTo http://clusterlabs.org/wiki/PostgresHowto
- Pacemaker Configuration Explained http://clusterlabs.org/doc/en-US/Pacemaker/1.1/html/Pacemaker_Explained/index.html
- SLES11 HA guide https://www.suse.com/documentation/sle_ha/singlehtml/book_sleha/book_sleha.html
- QVD Installation Guide http://docs.theqvd.com/en/4.0/QVDInstallationGuide.html