Data virtualization: playing with Oracle 12c on Docker containers

Franck Pachot, dbi-services

This year will be about data virtualization. We can't continue to multiply our database storage for each preprod, test, dev, etc. environments we have to provision. We need to be agile in that provisioning. There are great tools for that (such as Delphix I wrote about in previous newsletter) and alternatives I'll write about soon (snapshot standby, Snap Clone, etc). For application environment provisioning, there is a rising tool to do that with agility: Docker. It leverages copy-on-write filesystem and linux containers to bring data containers. So why not use it for our databases?

Warning: This is not about something you can do now for your production. I'm not writing here about a mature solution but only some prototyping I've done to give an idea about what is feasible, and to help you do your own tests in your lab – so please keep me informed (@FranckPachot or Franck.Pachot@dbi-services.com)

I'll not describe what is docker. There is the www.docker.com website for that. I'll show only an exemple. I've installed Oracle in a Docker container. I've created a standby database in it, so that it is refreshed continuously from a source database. And I can provision new virtual databases that are very light: a new Docker container that shares the storage for data which is not updated.

Install Docker

You can install Docker in Linux, or use the 'boot2docker' VirtualBox VM.

I've installed boot2docker and I have VirtualBox installed on my Windows laptop:

```
boot2docker init
```

This creates a virtualbox VM and I have to share a directory as '/Users' in order to see the Windows directory from Docker. I'll use that to put the Oracle software installation files.

```
VBoxManage sharedfolder add boot2docker-vm -hostpath "F:\SOFTWARE" -
 name /Users –automount
```

Then I can start the VM and I'll be able to ssh to it

```
boot2docker start
Waiting for VM and Docker daemon to start...
.................................
Started.
Docker client does not run on Windows for now. Please use
"C:\Program Files\Boot2Docker for Windows\boot2docker.exe" ssh
to SSH into the VM instead.
```

Let's now ssh to it to continue:

```
boot2docker ssh
                        ##        .
                  ## ## ##       ==
               ## ## ## ##      ===
           /""""""""""""""""\___/ ===
      ~~~ {~~ ~~~~ ~~~ ~~~~ ~~ ~ /  ===- ~~~
           \______ o          __/
             \    \        __/
              \____\______/
 _                 _   ____     _            _
| |__   ___   ___ | |_|___ \ __| | ___   ___| | _____ _ __
| '_ \ / _ \ / _ \| __| __) / _` |/ _ \ / __| |/ / _ \ '__|
| |_) | (_) | (_) | |_ / __/ (_| | (_) | (__|   <  __/ |
|_.__/ \___/ \___/ \__|_____\__,_|\___/ \___|_|\_\___|_|
Boot2Docker version 1.4.1, build master : 86f7ec8 - Tue Dec 16
 23:11:29 UTC 2014
Docker version 1.4.1, build 5bc2ff8
docker@boot2docker:~$
```

I'll need to know my ip addres:

```
docker@boot2docker:~$ ip addr
…
  inet 192.168.59.104/24 brd 192.168.59.255 scope global
…
```

which is 192.168.59.104

Install Oracle

With Docker, you never start from scratch. You start from a container that you download (pull) from the repository. In the public repository, you can't find an image with Oracle already installed. The reason is licensing. Licensing prohibits to distribute a VM with the software already installed. There is only one exception to that. It's Oracle XE. And you can find Docker images with Oracle XE in the repository. But I want to use a full feature Oracle 12c and I'll setup the standby with DataGuard, so I have to install it myself.

Ok, I'll start from an image that is ready for Oracle. It's the 'breed85 / oracle-12c' and you can get it with:

```
docker@boot2docker:~$ docker pull breed85/oracle-12c
```

Here is it:

```
docker@boot2docker:~$ docker images
REPOSITORY         TAG        IMAGE ID     CREATED      VIRTUAL SIZE
breed85/oracle-12c latest     9d3b9aab053b 11 weeks ago 1.728 GB
breed85/oracle-12c preinstall 9d3b9aab053b 11 weeks ago 1.728 GB
```

Everything is there except Oracle. But you can download the software for oracle website (http://www.oracle.com/technetwork/database/enterprise-edition/downloads/) and get the two zip files: linuxamd64_12102_database_1of2.zip and linuxamd64_12102_database_2of2.zip. You have to put them in a directory that you can mount in the Docker container.
Because I'm using boot2docker, I share the windows directory in the boot2docker VM as '/Users' and boot2docker mounts it.

Then I run the 'breed85/oracle-12c' image with a tag '12101' and sharing the boot2docker /Users as /Users within the container:

```
docker@boot2docker:~$ docker run --name 12102 -ti -v /Users:/Users
 breed85/oracle-12c:preinstall /bin/bash
```

If you didn't already pull the 'breed85/oracle-12c' image, it downloads it.
Then I have to add 'unzip' as well as the 'preinstall' package just to be sure I have it:

```
[root@2cd790c5a069 /]# yum -y install oracle-rdbms-server-12cR1-
 preinstall unzip
```

I make some space in the yum cache and I create the /oracle directory where I'll put everything that have to be accessed by oracle:

```
[root@2cd790c5a069 /]# mkdir -p /oracle/install
[root@2cd790c5a069 /]# chown -R oracle:dba /oracle
```

Time to unzip here:

```
[root@2cd790c5a069 /]# unzip
 /Users/linuxamd64_12102_database_1of2.zip -d
 /oracle/install/linuxamd64_12102_database
[root@2cd790c5a069 /]# unzip
 /Users/linuxamd64_12102_database_2of2.zip -d
 /oracle/install/linuxamd64_12102_database
```

I'm ready to install it. I use silent mode, so I generate a response file:

```
[root@2cd790c5a069 /]# cat > /oracle/db_install.rsp <<-'CAT'
oracle.install.responseFileVersion=/oracle/install/rspfmt_dbinstall_
 response_schema_v12.1.0
oracle.install.option=INSTALL_DB_SWONLY
ORACLE_HOSTNAME=
UNIX_GROUP_NAME=oinstall
INVENTORY_LOCATION=/oracle/oraInventory
SELECTED_LANGUAGES=en
ORACLE_HOME=/oracle/product/12102
ORACLE_BASE=/oracle
oracle.install.db.InstallEdition=EE
oracle.install.db.DBA_GROUP=dba
oracle.install.db.OPER_GROUP=dba
oracle.install.db.BACKUPDBA_GROUP=dba
oracle.install.db.DGDBA_GROUP=dba
oracle.install.db.KMDBA_GROUP=dba
CAT
```

And install as the oracle user:

```
[root@2cd790c5a069 /]# su - oracle -c
 "/oracle/install/linuxamd64_12102_database/database/runInstaller -
 silent -ignoreSysPrereqs -ignorePrereq -silent -noconfig -
 responseFile /oracle/db_install.rsp"
```

I'm ignoring a lot of prerequisites because I'm not on a supported configuration anyway.

At that point just wait 5 minutes for the installation and then run the root.sh

```
[root@2cd790c5a069 /]# /oracle/oraInventory/orainstRoot.sh
[root@2cd790c5a069 /]# /oracle/product/12102/root.sh
```

I don't have a lot of space in my Docker VM, so I cleanup the unzipped files that I don't need anymore, as well as some others that I don't need (did I already say don't do that in production?)

```
[root@2cd790c5a069 /]# rm -rf /var/cache/yum/x86_64
 /oracle/install/linuxamd64_12102_database
 /oracle/product/12102/inventory/backup/*
 /oracle/product/12102/assistants/dbca/templates
```

Of course I could have created the Docker VM a bit larger, but that is just a test.

```
[root@2cd790c5a069 /]# exit
```

Ok, that was all the time consuming part – downloading, unzipping, installing – and I save that in a new image:

```
docker@boot2docker:~$ docker commit 12102 breed85/oracle-12c:12102
```

I could help you and just push that image to the docker repository, but I'm sorry I don't want to pay enterprise edition licences for all the CPUs that are behind ;)

So here are my images:

```
docker@boot2docker:~$ docker images
REPOSITORY          TAG         IMAGE ID     CREATED        VIRTUAL
 SIZE
breed85/oracle-12c 12102      9cc040ef1d6e 5 minutes ago 6.264 GB
breed85/oracle-12c latest     9d3b9aab053b 11 weeks ago  1.728 GB
breed85/oracle-12c preinstall 9d3b9aab053b 11 weeks ago  1.728 GB
```

And I can remove the container that I just have commited:

```
docker@boot2docker:~$ docker ps -as
CONTAINER ID IMAGE                      COMMAND     CREATED
 STATUS                    PORTS NAMES SIZE
2cd790c5a069 breed85/oracle-12c:latest "/bin/bash" 18 minutes ago
 Exited (0) 5 minutes ago        12102 4.536 GB

docker@boot2docker:~$ docker rm 12102
```

Duplicate the database

As my goal is to simulate the virtualization of an existing database, I' will duplicate an existing database with RMAN and make it a physical standby.
My source database is the most simple one created with dbca and all default options (except that I create it as a non-CDB to keep it simple).

Here is the source database:

```
RMAN> report schema;

using target database control file instead of recovery catalog
Report of database schema for database with db_unique_name DEMO111

List of Permanent Datafiles
===========================
File Size(MB) Tablespace           RB segs Datafile Name
---- -------- -------------------- ------- ------------------------
1    780      SYSTEM               YES     /u02/…/o1_mf_system_bf67nzw0_.dbf
3    600      SYSAUX               NO      /u02/…/o1_mf_sysaux_bf67mwls_.dbf
4    255      UNDOTBS1             YES     /u02/…/o1_mf_undotbs1_bf67pgdj_.dbf
6    5        USERS                NO      /u02/…/o1_mf_users_bf67pf5q_.dbf

List of Temporary Files
=======================
File Size(MB) Tablespace           Maxsize(MB) Tempfile Name
---- -------- -------------------- ----------- --------------------
1    60       TEMP                 32767       /u02/…/o1_mf_temp_bf67qnwz_.tmp
```

My source database is called DEMO111 and is on host 192.168.78.111 and it's SYS password is "oracle" and I've set dg_brokerstart=true as I'll create the standby with the DataGuard broker. My docker VM has ip address 192.168.59.104

So if you copy/paste you may have to change those values.

I run a shell in a container from the image I've created before. And call it with 'DEMO111_SBY' as hostname. I don't need the /Users filesystem anymore. But now I'm redirecting the 1521 port to 9000. That means that when I'll start the listener on port 1521 in the container (listening to //DEMO111_SBY:1521 into the container) I will be able to access to it externally with the address //192.168.59.104:9000

Here is my run command (-h is for the hostname, -ti is for an interactive terminal):

```
docker@boot2docker:~$ docker run --name "DEMO111_SBY" -ti -h
 "DEMO111_SBY" -p 9000:1521 breed85/oracle-12c:12102 su - oracle
```

I set my environment. Instance name will be DEMO111:

```
[oracle@DEMO111_SBY ~]$ export ORACLE_HOME=/oracle/product/12102
[oracle@DEMO111_SBY ~]$ export ORACLE_SID=DEMO111
```

and create a password file identical to the primary database:

```
[oracle@DEMO111_SBY ~]$ $ORACLE_HOME/bin/orapwd
 file=$ORACLE_HOME/dbs/orapwDEMO111 password=oracle
```

I need a listener with a static service. Let's keep it minimal:

```
[oracle@DEMO111_SBY ~]$ cat >
 $ORACLE_HOME/network/admin/listener.ora <<-CAT
LISTENER=(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=DEMO111_SBY)(PORT
 =1521)))
SID_LIST_LISTENER=(SID_LIST=(SID_DESC=(GLOBAL_DBNAME=DEMO111_SBY_DGM
 GRL)(ORACLE_HOME=$ORACLE_HOME)(SID_NAME=DEMO111)))
CAT
```

And start it:

```
[oracle@DEMO111_SBY ~]$ $ORACLE_HOME/bin/lsnrctl start
```

Here is the output that shows the endpoint and the static service:

```
...
Listener Parameter File
 /oracle/product/12102/network/admin/listener.ora
Listener Log File
 /oracle/diag/tnslsnr/DEMO111_SBY/listener/alert/log.xml
Listening Endpoints Summary...

  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=DEMO111_SBY)(PORT=1521)))
Services Summary...
Service "DEMO111_SBY_DGMGRL" has 1 instance(s).
  Instance "DEMO111", status UNKNOWN, has 1 handler(s) for this
 service...
The command completed successfully
```

I'll use OMF and create the directories:

```
[oracle@DEMO111_SBY ~]$ mkdir /oracle/oradata /oracle/recovery_area
```

I need an init.ora to start my instance

```
[oracle@DEMO111_SBY ~]$ cat > /tmp/init.ora <<-CAT
db_name='DEMO111'
compatible=12.1.0.2.
db_unique_name='DEMO111_SBY'
db_create_file_dest='/oracle/oradata'
db_recovery_file_dest_size='1G'
db_recovery_file_dest='/oracle/recovery_area'
dg_broker_start=true
CAT
```

Then create spfile from it and start the instance:

```
[oracle@DEMO111_SBY ~]$ $ORACLE_HOME/bin/sqlplus / as sysdba <<-
 'SQL'
create spfile from pfile='/tmp/init.ora';
startup nomount;
SQL
```

Let's test network connection:

```
[oracle@DEMO111_SBY ~]$ $ORACLE_HOME/bin/tnsping
 "//192.168.78.111:1521/DEMO111"
…
Attempting to contact
  (DESCRIPTION=(CONNECT_DATA=(SERVICE_NAME=DEMO111))(ADDRESS=(PROTOCO
 L=TCP)(HOST=192.168.78.111)(PORT=1521)))
OK (0 msec)

[oracle@DEMO111_SBY ~]$ $ORACLE_HOME/bin/tnsping
 "//192.168.59.104:9000/DEMO111_SBY_DGMGRL"
…
Attempting to contact
  (DESCRIPTION=(CONNECT_DATA=(SERVICE_NAME=DEMO111_SBY_DGMGRL))(ADDRE
 SS=(PROTOCOL=TCP)(HOST=192.168.59.104)(PORT=9000)))
OK (0 msec)
```

Everything is perfect. As I can connect to source and target, let's do an rman duplicate from active. Best practice is to declare the connection string in a tnsnames.ora but let's do it quickly with the ezconnect string (which works as long as it is less than 64 characters).

```
[oracle@DEMO111_SBY ~]$ $ORACLE_HOME/bin/rman

RMAN> connect target sys/oracle@//192.168.78.111:1521/DEMO111;
connected to target database: DEMO111 (DBID=2551031863)


RMAN> connect auxiliary
 sys/oracle@//192.168.59.104:9000/DEMO111_SBY_DGMGRL;
connected to auxiliary database: DEMO111 (not mounted)

RMAN> duplicate database for standby from active database;
Starting Duplicate…
…
Finished Duplicate Db…
```

Configure the standby

I'm using the DataGuard broker to setup quickly the standby:

```
[oracle@DEMO111_SBY ~]$ $ORACLE_HOME/bin/dgmgrl <<-DG
connect sys/oracle@"//192.168.78.111:1521/DEMO111"
create configuration 'DEMO111' as primary database is 'DEMO111'
 connect identifier is '192.168.78.111:1521/DEMO111';
add database 'DEMO111_SBY' as connect identifier is
 '//192.168.59.104:9000/DEMO111_SBY_DGMGRL ';
enable configuration;
edit database 'DEMO111_SBY' set state=apply-on;
DG
```

Don't forget that the goal is just to have a standby that is synchronized. We will never switchover to the Docker container ! Here is the output:

Ok, just wait a while and it is synchronized:

```
DGMGRL> connect /
Connected as SYSDBA.
DGMGRL> show database "DEMO111_SBY"
Database - DEMO111_SBY

  Role:               PHYSICAL STANDBY
  Intended State:     APPLY-ON
  Transport Lag:      0 seconds (computed 20 seconds ago)
  Apply Lag:          0 seconds (computed 20 seconds ago)
  Average Apply Rate: 30.00 KByte/s
  Real Time Query:    OFF
  Instance(s):
    DEMO111
```

And that's all. In order to keep that configuration, we can quit the interactive shell and save this image:

```
docker@boot2docker:~$ docker commit DEMO111_SBY breed85/oracle-
 12c:DEMO111_SBY
```

remove the container:

```
docker@boot2docker:~$ docker rm DEMO111_SBY
```

And check the new image that has additional 2.13GB for our database:

```
docker@boot2docker:~$ docker images
REPOSITORY          TAG         IMAGE ID     CREATED
 VIRTUAL SIZE
breed85/oracle-12c DEMO111_SBY 31068a2689fd About a minute ago 8.394
 GB
breed85/oracle-12c 12102       9cc040ef1d6e About an hour ago  6.264
 GB
breed85/oracle-12c latest      9d3b9aab053b 11 weeks ago       1.728
 GB
breed85/oracle-12c preinstall  9d3b9aab053b 11 weeks ago       1.728
 GB
```

Refresh the standby

Our goal is to have that standby refreshed continuously, so let's run a container that we will leave running. I can put it in background, but for testing I'll leave it running in one window.

It's the same command line except that I call the container 'refresh'

```
docker@boot2docker:~$ docker run --name "refresh" -ti -h
 "DEMO111_SBY" -p 9000:1521 breed85/oracle-12c:DEMO111_SBY su -
 oracle
```

And I run the minimum required: start the listener:

```
[oracle@DEMO111_SBY ~]$ export ORACLE_HOME=/oracle/product/12102
[oracle@DEMO111_SBY ~]$ export ORACLE_SID=DEMO111
[oracle@DEMO111_SBY ~]$ $ORACLE_HOME/bin/lsnrctl start
```

And the instance:

```
[oracle@DEMO111_SBY ~]$ $ORACLE_HOME/bin/sqlplus / as sysdba
 startup nomount;
 alter database mount standby database;
```

and I keep that running. I start docker into another window:

```
cd "C:\Program Files\Boot2Docker for Windows"
boot2docker ssh
docker@boot2docker:~$
```

I’ve the following images:

```
docker@boot2docker:~$ docker images
REPOSITORY          TAG          IMAGE ID     CREATED        VIRTUAL SIZE
breed85/oracle-12c DEMO111_SBY 31068a2689fd 41 minutes ago 8.394 GB
breed85/oracle-12c 12102       9cc040ef1d6e 2 hours ago    6.264 GB
breed85/oracle-12c latest      9d3b9aab053b 11 weeks ago   1.728 GB
breed85/oracle-12c preinstall  9d3b9aab053b 11 weeks ago   1.728 GB
```

each are build on top of the other, this is why I don’t need space to store the virtual sized that we see here.

My ‘refresh’ container is running.

```
docker@boot2docker:~$ docker ps -as
…IMAGE            COMMAND       PORTS                        NAMES    SIZE
…12c:DEMO111_SBY "su - oracle" 0.0.0.0:9000->1521/tcp    refresh
 1.793 GB

docker@boot2docker:~$ docker ps -as
CONTAINER ID IMAGE                              COMMAND       CREATED
 STATUS         PORTS                       NAMES    SIZE
9f2d6e5333bb breed85/oracle-12c:DEMO111_SBY "su - oracle" 20 minutes
 ago Up 20 minutes 0.0.0.0:9000->1521/tcp refresh 1.808 GB
```

We see the size (which is the copy-on-write storage over the base image) and the port redirection. Status is running.

now I save its state to another image:

```
docker@boot2docker:~$ docker commit refresh breed85/oracle-12c:vdb1
```

Then I have an additional image:

```
docker@boot2docker:~$ docker images
REPOSITORY          TAG          IMAGE ID     CREATED         VIRTUAL
 SIZE
breed85/oracle-12c vdb1         be0ba1450a98 42 seconds ago 10.2 GB
breed85/oracle-12c DEMO111_SBY  31068a2689fd 46 minutes ago 8.394 GB
breed85/oracle-12c 12102        9cc040ef1d6e 2 hours ago    6.264 GB
breed85/oracle-12c latest       9d3b9aab053b 11 weeks ago   1.728 GB
breed85/oracle-12c preinstall   9d3b9aab053b 11 weeks ago   1.728 GB
```

You see that I’ve now an additional virtual database server that has 10.2GB (sytem + oracle software + datafiles + archived logs) but I need only 1.79GB to store it – which is the size of the ‘vdb1’ layer.

Those are virtual sizes. The real physical size is visible on the AUFS filesystem used by docker:

```
docker@boot2docker:~$ df -h /mnt/sda1/var/lib/docker/aufs
Filesystem                Size      Used Available Use% Mounted on
/dev/sda1                18.2G     11.3G      5.9G  66% /mnt/sda1
```

What is the cost to provision another virtual database?

```
docker@boot2docker:~$ docker commit refresh breed85/oracle-12c:vdb2
Filesystem                Size      Used Available Use% Mounted on
/dev/sda1                18.2G     13.0G      4.2G  76% /mnt/sda1

docker@boot2docker:~$ docker images
REPOSITORY         TAG          IMAGE ID     CREATED            VIRTUAL SIZE
breed85/oracle-12c vdb2         61d372cc903d About a minute ago 10.2 GB
breed85/oracle-12c vdb1         be0ba1450a98 6 minutes ago      10.2 GB
breed85/oracle-12c DEMO111_SBY  31068a2689fd 51 minutes ago     8.394 GB
breed85/oracle-12c 12102        9cc040ef1d6e 2 hours ago        6.264 GB
breed85/oracle-12c latest       9d3b9aab053b 11 weeks ago       1.728 GB
breed85/oracle-12c preinstall   9d3b9aab053b 11 weeks ago       1.728 GB
```

The new database server has used 1.7 GB but we have another copy of a 10.2GB virtual size.

Opening the virtual database

So, what do we need in order to use the new virtual database? Just run the image in a new container and redirect the 1521 port to another one:

```
docker@boot2docker:~$ docker run --name "vdb1" -ti --rm -h
 "DEMO111_vdb1" -h "vdb1" -p 9001:1521 breed85/oracle-12c:vdb1 su -
 oracle
```

I’ve added the --rm’ option so that the container is removed when I quit the shell. I do that when I just want to use it for a short period of time.

I’ll be able to access to it using the redirected port 9001 so the connection string will be: //192.169.59.104:9001/DEMO111_SBY but for the moment, I have to start the listener. Note that I remove the listener.ora which was configured for another hostname:

```
[oracle@vdb1 ~]$ export ORACLE_HOME=/oracle/product/12102
[oracle@vdb1 ~]$ export ORACLE_SID=DEMO111
[oracle@vdb1 ~]$ rm $ORACLE_HOME/network/admin/listener.ora
[oracle@vdb1 ~]$ $ORACLE_HOME/bin/lsnrctl start
```

Then start the instance :

```
[oracle@vdb1 ~]$ $ORACLE_HOME/bin/sqlplus / as sysdba
 startup nomount;
 alter database mount standby database;
```

and activate the standby to become a primary in order to open it read write – operation which is called a failover.

```
[oracle@vdb1 ~]$ $ORACLE_HOME/bin/dgmgrl /
  failover to "DEMO111_SBY" immediate;

…
Performing failover NOW, please wait...
Failover succeeded, new primary is "DEMO111_SBY"
```

Of course, it can be a good idea to isolate the container when starting it so that we are sure that this new primary doesn't try to communicate with the actual primary and standby. We can also change the database name but containers can provide the isolation we need. I want only one thing to pass-through: the 9001:1521 port redirection.

The result is that I have now two containers running:

```
IMAGE            CREATED   STATUS  PORTS                NAMES    SIZE
12c:vdb1         9 minut   Up 9 m  …:9001->1521/tcp     vdb1     2.128 GB
12c:DEMO111_SBY  About a   Up Abo  …:9000->1521/tcp     refresh  1.813 GB
```

I've made a few updates in the 'vdb1' which is why the size has increased and the 'refresh' one is still applying redo from the primary.

I can run as many virtual databases as I want. They are managed as any Docker containers. And if you wonder about the resources I have on my Docker VM, here is the output from top:

```
Mem: 1889632K used, 166604K free, 0K shrd, 376K buff, 376K cached
CPU:  0.1% usr  0.0% sys  0.0% nic 99.8% idle  0.0% io  0.0% irq
 0.0% sirq
Load average: 0.20 0.32 0.41 2/411 8023
…
 5960  5908 54321    S     662m 32.8   5  0.0 {ora_pmon_demo11}
 ora_pmon_DEMO111
…
 7691  7631 54321    S     662m 32.8   1  0.0 {ora_pmon_demo11}
 ora_pmon_DEMO111
…
```

All the instances are running there (you see multiple pmon). It's all running on the same server but isolated in containers.

Connect to the virtual database

Then I can connect to my vdb1 from any client that has access to the docker VM through the network:

```
SQL> connect system/oracle@//192.168.59.104:9001/DEMO111_SBY
```

And check on which host it is running:

```
SQL> select host_name from v$instance;
HOST_NAME
----------------------------------------
DEMO111_vdb1
```

Which is the virtual hostname passed to the ‘run’ command through the ‘-h’ argument.

Conclusion

What’s the point writing an article about something we cannot do on production? I wanted to show the concepts of data virtualization and I like to learn by practice. All the underlying technologies (copy-on-write filesystem, virtualization, isolation) are there and you can use them with simple tools such as Docker an practice on your laptop.

Try it. Download Docker. Download Oracle zip files. And do the same which is probably as simple as copy-paste the commands and replace only ip addresses.

Of course, for your production you will have to go to robust and supported software and many vendors have nice solutions for data virtualization, easy snapshots and clones. But I’m sure that once you have practiced the concepts with that do-it-yourself unsupported solution, then you will have better bases for selecting and challenging the commercial solutions.

Contact:
Franck Pachot
franck.pachot@dbi-services.com

Franck Pachot is senior consultant at dbi services in Switzerland. He has 20 years of experience in Oracle databases, all areas from development, data modeling, performance, administration, and training. Oracle Certified Master and Oracle ACE, he tries to leverage knowledge sharing in forums, publications, presentations.